

Efficient robotic manipulation skill learning

by

Jianlan Luo

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Alice Agogino, Co-chair

Professor Pieter Abbeel, Co-chair

Professor Roberto Horowitz

Summer 2020

Efficient robotic manipulation skill learning

Copyright 2020

by

Jianlan Luo

Abstract

Efficient robotic manipulation skill learning

by

Jianlan Luo

Doctor of Philosophy in Mechanical Engineering

University of California, Berkeley

Professor Alice Agogino, Co-chair

Professor Pieter Abbeel, Co-chair

In this thesis, we study how to efficiently learn robotic manipulation skills from various perspectives. We start by reviewing the current trend of applying learning-based methods in the domain of robotics; despite their great potentials, these methods still have limits on sample efficiency and the ability to generalize across domains. We aim to tackle these issues from roughly two aspects: introducing suitable policy priors into the learning process and robotic specific representations that are effective and generalizable. We first introduce an effective algorithm called residual reinforcement learning, which essentially regularizes policy learning in function space by superimposing its action with the control signal from a “prior controller”, the variance during policy learning is reduced at the cost of injecting certain amount of bias which can be quantified. We validate our method on a variety of contact-rich fine-grained robotic manipulation tasks. On the other perspective, we study how robot learning could benefit from an effective representation. We introduce an action space representation for reinforcement learning algorithms: an operational space force controller. When a RL agent is combined with this operational force controller, it tends to learn substantially faster and perform better than other action space in several high-precision assembly tasks. We then propose a representation that explicitly models gripper features in grasping tasks, this allows us to collect data entirely in simulation and transfer to real-world without any fine-tuning nor loss of performance. We conduct large-scale real world experiments on a variety of objects and environments to validate this method. On a similar fashion, we extend a method called domain randomization by explicitly modelling robot rigid transformations in the task of pose estimation, which again trains entirely on synthetic data and follows a zero-shot transfer to real world. Results show that our method could achieve significantly better accuracy. Finally, we conclude that injecting prior information in suitable forms such as control bias, state representation, simulation model, etc., will dramatically increase the performance in a broad range of hard robotic problems; however these inductive bias will make our solution less

general. We point future directions as to come up with principled way to design and inject such prior information to learning systems.

To My Family.

Contents

Contents	ii
List of Figures	iv
List of Tables	viii
1 Introduction	1
1.1 Deep learning and its application in robotics	1
1.2 Deep reinforcement learning and its application in robotics	3
1.3 Dissertation outline and contributions	5
2 Learning residual policies for robotic control	8
2.1 Introduction	8
2.2 Related work	11
2.3 Method	12
2.4 Experiments	16
2.5 Extension to visual inputs and natural rewards	22
2.6 Experiments	25
2.7 Interpretation as bias-variance trade-off	31
2.8 Discussion	35
3 Learning complex manipulation with action space representation	37
3.1 Introduction	37
3.2 Related work	38
3.3 Method	39
3.4 Experiments	41
3.5 Discussion	53
4 Learning scalable robotic grasping with action image representation	55
4.1 Introduction	55
4.2 Related work	57
4.3 Method	58
4.4 Experiments	65

4.5 Discussion	70
5 Learning active pose estimation with domain randomization	72
5.1 Introduction	72
5.2 Related work	74
5.3 Method	75
5.4 Experiments	78
5.5 Discussion	84
6 Conclusions	85
Bibliography	87

List of Figures

1.1	A common neural network architecture.	2
1.2	Reinforcement learning illustration, source: Sutton and Barto, Reinforcement learning: an introduction [23].	4
2.1	We train an agent directly in the real world to solve a model assembly task involving contacts and unstable objects. An outline of our method, which consists of combining hand-engineered controllers with a residual RL controller, is shown on the left. Rollouts of residual RL solving the block insertion task are shown on the right. Residual RL is capable of learning a feedback controller that adapts to variations in the orientations of the standing blocks and successfully completes the task of inserting a block between them. Videos are available at residualrl.github.io	9
2.2	Block assembly task in simulation (left) and real-world (right). The task is to insert a block between the two blocks on the table without moving the blocks or tipping them over. In the learning curves, we compare our method with RL without any hand-engineered controller [†] . In both simulation and real-world hardware experiments, we see that residual RL learns faster than RL alone, while achieving better performance than the hand-engineered controller.	16
2.3	Simulation results for different experiments. In each plot, the final average return obtained by running the method for various settings of a parameter is shown. Plot (a) shows that residual RL can adjust to noise in the environment caused by rotation of the blocks in a range of 0 to 0.3rad. In plot (b), residual RL finds robust strategies in order to reduce the effect of control noise, as the final average return is not greatly affected by the magnitude of noise. Plot (c) shows that residual RL can compensate for biased controllers and maintains good performance as control bias increases, while the performance of the hand-designed controller dramatically deteriorates with higher control bias.	19

2.4	Outcome of our residual RL method in different experiments during the block assembly task in the real-world. Success rate is recorded manually by human judgment of whether the blocks stayed upright and ended in the correct position. Plot (a) compares the insertion success of residual RL and hand-designed controller depending on the block orientation during run time. Plot (b) shows the success rate of the insertion process during training, where on every reset the blocks are randomly rotated: straight, tilted clockwise, or tilted counterclockwise ($\pm 20^\circ$) and plot (c) shows the increasing success rate of our method for biased controllers as well even as control bias increases.	20
2.5	Real-world block insertion results using residual RL for sim-to-real transfer. “Sim” indicates that the real-world policy was initialized by reinforcement learning in simulation. Residual RL with simulation initialization successfully solves the task with little environment experience required.	21
2.6	A close-up view of the three connector insertion tasks shows the contacts and low tolerances the agent must navigate to solve these tasks. These tasks require sub-millimeter precision without visual feedback.	22
2.7	We train policies directly in the real world to solve connector insertion tasks from raw pixel input and without access to ground-truth state information for reward functions. Left: a rollout from a learned policy that successfully completes the insertion task for each connector is shown. Right: a full view of the robot setup. Videos of the results are available at industrial-insertion-rl.github.io	23
2.8	Illustration of the robot’s cascade control scheme. The actions u_t are computed at a frequency of up to 10 Hz, desired joint angles are obtained by inverse kinematics, and a joint-space impedance controller with anti-windup PID control commands actuator torques at 1000 Hz.	25
2.9	Successful insertion on the Model-E connector. The image-based RL algorithms receives only receives the 32×32 grayscale image as the observation.	26
2.10	Resulting final mean distance during the vision-based training; the distance is calculated as pixel-wise L1 norm. One rollout amounts to 50 time steps. The comparison includes RL, residual RL, and RL with learning from demonstrations. Only residual RL manages to deal with the high-dimensional input and consistently solve all the tasks after the given amount of training. The other methods learn to move downwards, but often get stuck in the beginning of the insertion and fail to recover from unsuccessful attempts.	26
2.11	Average success rate on the USB insertion task. Residual RL and RL + LfD solve the task consistently. Moreover, residual RL stays robust under $\pm 1\text{mm}$ noise.	27
2.12	Plots of the final mean distance to the goal during the state-based training. Final distances greater than 0.01 m indicate unsuccessful insertions. Here, the residual RL approach performs noticeably better than pure RL and is often able to solve the task during the exploration in the early stages of the training.	28

2.13	Learning curves for the training of the USB insertion task with a sparse reward. Success rate is shown; higher is better. One rollout amounts to 50 time steps. Residual RL and RL with learning from demonstrations both solve the task relatively quickly, while RL alone takes about twice as long to solve the task at the same performance.	29
2.14	We report average success out of 25 policy executions after training is finished for each method. For noisy goals, noise is added in form of ± 1 mm perturbations of the goal location. Residual RL, particularly with SAC, tends to be the best performing method across all three connectors. For the Model-E connector, only residual RL solves the task in the given amount of training time.	30
3.1	Experimental setup for peg insertion task.	42
3.2	Assembly modes for rigid peg and deformable material.	43
3.3	(a) Successful insertion. (b) Failure mode where the policy is unable to locate the hole.	44
3.4	Learning curve in three experiments starting from different positions.	45
3.5	Learning control policies for assembly tasks.	46
3.6	Neural Network Architecture	48
3.7	Snapshots of experimental runs for the four studied assembly tasks.	50
3.8	Action computed by learned policy during one successful insertion. Solid line shows computed action mean, and error bar for computed variance.	51
3.9	Six degree of freedom force torque measurements from a successful Task 2 insertion.	52
4.1	Real-world grasp success evaluation of policies trained only in simulation using a cluster of 8 mobile manipulators.	56
4.2	The procedure for performing end-to-end grasping using an <i>Action Image</i> representation. Our mobile manipulator first approaches the workspace and then grasp candidates are uniformly sampled from the workspace. Next, a preprocessor generates pairs of sensed-image and action-image data corresponding to crops centered around the grasp proposals. Then, an <i>Action Image</i> network evaluates the candidate pairs and if a grasp proposal exceeds the desired probability of success the grasp will be executed by the robot. Otherwise, the grasp candidate distribution is updated with the top 10% of elite proposals and the process is repeated.	59
4.3	The description of the <i>Action Image</i> preprocessor. Given a sample pose, the 3D position of the gripper features is calculated and projected into camera-space producing the <i>Action Image</i> representation. The number of these features determines the expressiveness of the representation and at least 3 points are required to estimate the pose. The values of the projected points can carry additional information, such as depth in camera frame. Next, the resulting <i>Action Image</i> and sensor-image data are adaptively cropped to a $128 \times 128 \times 3$ square centered at the proposed grasp candidate and then are resized to $256 \times 256 \times 3$	60

4.4	Left: We represent a gripper by 3 feature points: two features are placed on the gripper finger tips and the third one is placed on the point where the fingers are attached to the wrist. Right: These feature points can be easily projected into camera-space through known kinematic transformations available from the robot. The generated <i>Action Image</i> is shown in the bottom-right image.	62
4.5	Representations for the <i>Action Image</i> networks used in this section were (a) action RGB (b) action D and (c) action RGB-D. All networks were composed solely of convolutional layers. These networks were compared to a baseline (d) action vector network where the grasp candidates were fed to the network as a 4D vector of floats $[x, y, \psi, \alpha]$	63
4.7	Set of 48 simulated objects used in the training of all models.	66
4.8	In total 172 objects are used in the real robot evaluation, including 80 familiar (left) and 92 novel (right) objects.	66
4.9	Workspace and random base footprints for real-world (right) and simulated (left) scenarios.	68
4.10	Snapshots of the grasping attempts on novel objects in the scenes large, medium, and small (from top to bottom). Images correspond to the initial scene (left) and the time of grasp (right).	70
5.1	Inverse Transform Domain Randomization: We show that we can improve the accuracy of real world pose prediction with multiple images of a scene with known geometric transformations between object poses in the scenes.	73
5.2	Simulation to reality transfer with active reference object tabletop movement . .	78
5.3	Experiment setups: (a) estimate pose relative to movable reference object. (b) estimate pose relative to robot gripper.	79
5.4	Simulation to reality transfer with active gripper motion. Left: simulated images with domain randomization. Right: real images. Active perception here is based on moving the robot gripper.	81
5.5	Different transform applied to the reference object as described in Section 5.4. The green cylinder is the reference object and we are estimating the pose of the black peg. As seen from these figures we can transform the position of the green cylinder to four different positions and use multiple images to improve pose estimation .	82
5.6	Different camera angles of the same setting with target object boxed up. We want to estimate the position and orientation of the target object relative table corner. We can use multiple observing angles to explore the geometric properties of the target object and achieve a better pose estimation.	83
5.7	Distribution of the error in horizontal direction, vertical direction, and θ with respect to the orientation of the object(θ). Using one image, we see a high prediction error when the peg is orientated away from the camera (θ around 3.14) due to the asymmetric shape of the peg. When using three images, the prediction for this previous difficult object orientation is significantly improved.	83

List of Tables

3.1	Comparison of success rate, with different distance to the hole.	45
3.2	Comparison of success rates for different tasks. Baseline 1 refers to kinematics only; baseline 2 refers to iLQG with direct torque control; baseline 3 refers to iLQG with direct torque control, augmented state space, our method w/ augmented refers augmented state space in our method.	49
3.3	Comparison of success rate in generalization capability	53
4.1	A list of simulation datasets and their size. On-policy data #1 and #2 were collected from policies trained on Dataset A and B, respectively.	67
4.2	Evaluation results in simulation. Success rate for each model is based on +5K grasp evaluations.	68
4.3	Quantitative evaluation of on-robot testing. These results are based on 2160 grasps in total. The success rate for each scenario was calculated from, on average, 83 grasps, except for the random policy where only, on average, 28 grasps were performed.	69
5.1	Table showing the mean prediction error for pose prediction for the moving reference object scenario described in Section 5.4. This table shows that performance of pose prediction transferred from simulation to reality can be significantly improved by using the robot actively to modify the scene being considered. Prediction error is low when the reference object is placed at all four corners or on diagonal corners rather than along an edge of the table. This indicates that multiple images with known transformations do help reduce pose prediction error, and the choice of which transforms to use has a significant effect on the prediction error.	79
5.2	Table showing the mean prediction error for pose prediction for moving gripper scenario described in Section 5.4. We see that using multiple images with known geometric transformations is able to significantly reduce the angle error and provide some improvements in the estimation of the offset of the gripped object as well.	82
5.3	Table showing the mean prediction error for pose prediction for the moving camera scenario described in Section 5.4. We see that using multiple images taken from different point of view is able to reduce both the coordinates and angle error. . .	84

Acknowledgments

It is an unforgettable journey of the past five years for me. My personal experience at Berkeley has shaped me to what I am today that I could not have ever imagined five years ago. I want to first thank Professor Homayoon Kazerooni for introducing me to Berkeley. I want to express my deep gratitude to my advisor Professor Alice Agogino for giving me the research freedom, providing constant support and advice throughout my PhD study; which made this thesis possible. I am also extremely grateful to Professor Pieter Abbeel, who generously serves as the chair of my master thesis in computer science and co-chair of this thesis. Pieter is the very best in the research fields of machine learning, robotics and their intersections; I entered this field of robot learning by reading Pieter's pioneering work. I am really fortunate to interact with Pieter during my PhD, his positive responses always encourages me to move forward.

I want to also thank Professor Roberto Horowitz for both serving in my qual committee and thesis committee. I would like to thank Professor Anant Sahai, Professor Somayeh Sojoudi and Professor Mark Mueller serving in my qual committee.

I have also had the great fortune working with a lot of people during my PhD. I would like to thank Ashvin Nair, Tobi Johannik, Shikhar Bahl, Aviv Tamar, Gerrit Schoettler, Xinyi Ren, Jeff Bingham, Mohi Khansari, Daniel Kappler. I also want to thank Stefan Schaal, Ravi Kolluri as my internship hosts at Google, which provides me a wonderful experience; and Jon Scholz from Deepmind offering me the opportunity to join his team. There are also a lot of people I interact at Berkeley, Google, and research conferences from which I learn a lot, I could not list them all, but I want to thank them as well.

I want to thank awesome colleagues and friends which make my experience both at Berkeley and Google more enjoyable: Hezheng Yin, An Ju, Andrew Sabelhaus, Eldon Schoop, Jianyu Chen, Zhi Chen, Xiangyu Yue, Bodi Yuan, Chang Li, Zining Wang, Teng Zeng, Ying Xu, Chuyuan Fu, Xuchen Han, Hongxu Ma, Xinyue Li.

This thesis is dedicated to my parents and all my family members for their unconditional love and support.

Chapter 1

Introduction

1.1 Deep learning and its application in robotics

Deep learning has demonstrated remarkable performance in a variety range of domains such as computer vision, natural language process, speech synthesis and others [1]. Compared to classical machine learning algorithms where hand-crafted features are often needed, the expressive power of deep neural networks and massive amount of data available today makes deep learning algorithms discover those features from data automatically [2]. The key component in deep learning is a deep neural network that is trained on some training datasets; if the training data comes with pairs of inputs and labels, we often refer it as supervised learning; if no labels are provided in the training dataset, we often use deep neural networks to extract features from the data themselves, which we refer as unsupervised learning. Such deep neural network models are typically trained with first-order methods such as gradient descent, an algorithm called backpropagation can efficiently compute the gradient w.r.t all parameters in the neural network by using chain rules [3]. Figure 1.1 shows a classical neural network architecture: input data are passed through the input layer, several hidden layers connected to each other apply linear transformation to inputs then followed by some nonlinear operations which we call “activation functions”, finally outputs from hidden layers will be consolidated into the output layer with desired dimension. Outputs from neural network will be compared to provided labels in the training dataset, the resulting errors will be used to find gradient directions to minimize such discrepancies. An important aspect of the design of such neural network is to select a proper cost function. Usually, we simply choose the maximum likelihood principle to design cost functions, this way the burden of choosing specific cost function for each individual model is removed since the neural network is entirely parameterized. More rigorously, the neural network usually models a parameterized conditional distribution $p(\mathbf{y}|\mathbf{x};\theta)$, where θ represents parameters in our neural network. Of course, there are also several other generative models aiming for recovering the complete marginal distribution in unsupervised, or semi-supervised ways; however, in this thesis we focus mostly on supervised learning where we merely want to predict some statistics of $\mathbf{y}$

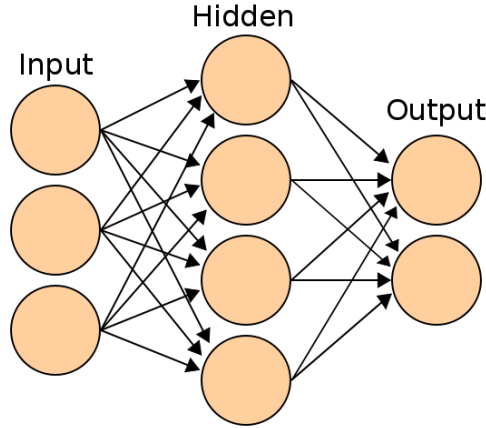


Figure 1.1: A common neural network architecture.

conditioned on $\mathbf{x}$. As we described in the last paragraph, we usually adopt the negative log-likelihood as cost function, equivalently as the cross-entropy between the training data and the neural network model distributions. We can write down this cost function as following:

$$J(\theta) = -\mathbb{E}_{x,y \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{y}|\mathbf{x}) \quad (1.1)$$

Depending on the specific form $p_{\text{model}}(\mathbf{y}|\mathbf{x})$, equation 1.1 could exhibit different forms. For instance, in regression problems, $p_{\text{model}}(\mathbf{y}|\mathbf{x}) = \mathcal{N}(f(\mathbf{x}; \theta), \mathbf{I})$, where f represents operations applied by the neural network; plug this into equation 1.1, we can easily find equation 1.1 reduces to mean-squared error:

$$J(\theta) = \frac{1}{2} \mathbb{E}_{x,y \sim \hat{p}_{\text{data}}} \|\mathbf{y} - f(\mathbf{x}; \theta)\|^2 + \text{constant}, \quad (1.2)$$

the constant term depends on the variance of the Gaussian, which we usually don't choose to parameterize. With this cost function in hand, we can then use gradient-based methods to optimize it:

$$\theta_{k+1} = \theta_k + \alpha \nabla J(\theta_k), \quad (1.3)$$

where k is the optimization iteration, α is the step size, and we can calculate this gradient very effectively by chain rule, the associated algorithm is called backpropagation as we mentioned before, we refer reader to [2], [3] for more details.

Recently, deep learning methods have gained a lot of attraction in the robotics community. Its significant advancement in computer vision opens a new door for many visuomotor control problems. Almost every single advancement in any sub-domain of computer vision could find a direct application in robotics: pose estimation [4–8], object detection [9–12], classification [2], [13], [14], instance segmentation [15] and etc. And there are also work trying to tie perception with control in the same loop [16–20]. The success of these methods heavily

rely on the availability of high-quality labeled training data, which comes at the price of significant manual labeling effort; furthermore, such datasets may be not available at all in many application scenarios such as pose estimation, visuomotor control, whose labels are not as simple as classification tags or bounding boxes. Additionally, these end-to-end methods do not effectively exploit certain existing properties of robotic problems, injecting these priors will make the learning process more structured and effective. We aim to solve these issues from two perspectives: 1) effective use of simulation data with simulation-to-real transfer, this way we theoretically get unlimited amount of data we can train our neural network on; 2) effective use of well-understood structures of certain robotic problems, we leverage principles of rigid transformations in chapter 4 to improve accuracy of pose estimation, and we explicitly model grasping-invariant features in chapter 3 to get a good grasping representation that could generalize better.

1.2 Deep reinforcement learning and its application in robotics

Reinforcement learning is about learning what to do, i.e., how to map situations to actions, so as to maximize a numerical reward signal [21]. The agent is not told which action to take, instead, it must discover those actions with highest reward by trying them out. Reinforcement learning is different from supervised learning which we discussed in the previous section: supervised learning requires a training dataset with labeled data which can be provided by an external supervisor; however, reinforcement learning is guided by a reward signal which can only be acquired through interaction with the environment. Figure 1.2 gives an illustration of such process.

We often study reinforcement learning in the context of Markov decision process (MDP), for its proper abstraction of problems. An MDP consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, and the state transition probability $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty)$; where p represents the probability distribution of the next state s_{t+1} given the current state s_t and action taken a_t ; and there is often a reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ associated with such MDP process. We often denote an MDP with the tuple $(\mathcal{S}, \mathcal{A}, p, r)$, and the resulting policy $\pi(a_t|s_t)$; a trajectory distribution $p(\tau)$ can be induced by applying this policy, where we denote $\tau = (s_0, a_0, s_1, a_1, \dots)$. Usually, the policy $\pi(a|s)$ is parameterized somehow, let's denote it as $\pi_\theta(a_t|s_t)$; if θ is a deep neural network, we often refer the associated problem as deep reinforcement learning.

The goal is often to learn such a policy $\pi(a_t|s_t)$ to maximize accumulated expected rewards under its distribution; which we can define using quantities as above $\sum_t \mathbb{E}_{(s_t, a_t) \sim p_\pi} [r(s_t, a_t)]$. We can solve this problem generally from two perspectives: value-based methods and policy-based methods. A value function is defined as total expected rewards if we act according to some policy π at a specific time step: $V_\pi(s) = \sum_{k=t}^{\infty} [r_k | s_t = s]$. Similarly, we define a Q-function as the total expected rewards if we take a specific action at a specific time step: $Q_\pi(s, a) = \sum_{k=t}^{\infty} [r_k | s_t = s, a_t = a]$. These two quantities obey the so-called Bellman backup

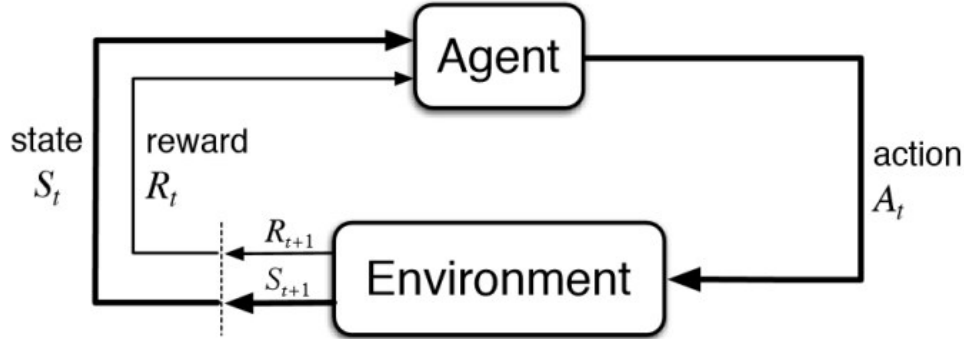


Figure 1.2: Reinforcement learning illustration, source: Sutton and Barto, Reinforcement learning: an introduction [23].

as follows:

$$V_{k+1}(s) = \mathbb{E}_{\pi}[r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s] \quad (1.4)$$

$$Q_{\pi}(s, a) = \mathbb{E}[r_{t+1} + \gamma V_{\pi}(s_{t+1}) | s_t = s, a_t = a], \quad (1.5)$$

where $\gamma \in [0, 1]$ is a discount factor, V_k is the estimation of value function at the k -th iteration. With these equations, we can solve the aforementioned MDP by recursively applying them to get estimates of value function. We can extract the desired policy $\pi(a|s)$ by performing maximization operation over the value function: in discrete case, this can be done analytically; in continuous case, this could be approximated by some sampling methods or we could use a neural network to learn such maximization. On the other side, without resorting to use these action-value methods, we can solve for a policy by directly optimizing its parameters, these methods are usually called policy-based methods. Note our objective is to maximize total return $J(\theta) = \sum_t \mathbb{E}_{(s_t, a_t) \sim p_{\pi_{\theta}}}[r(s_t, a_t)]$ under current policy π_{θ} , and this utility is only a function of policy parameter θ , therefore we can optimize $J(\theta)$ by gradient ascent on the parameter θ :

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \quad (1.6)$$

Taking the full gradient $\nabla J(\theta_t)$ can be prohibitive since we don't have access to all state-action marginals; however, we could estimate this quantity by Monte Carlo sampling at each iteration, these are so called policy gradient methods. Since sampling may lead to high variance in our gradient estimator, many policy gradient style methods use variance deduction techniques such as inserting baselines, constraining parameter update to a trust region [22], [23].

As we described, deep reinforcement learning(DRL) has the advantage to acquire complex sensorimotor skills by constantly interacting with the environment rather than engineering complicated heuristic; and the use of general purpose deep neural network also enables DRL

algorithms to handle complex sensory inputs [20], [24]–[26]. Due to such nature of deep reinforcement learning, it has emerged as a promising direction to tackle many hard robotic problems such as manipulation and locomotion where handcrafted controller design can be very difficult. There have been several attempts applying deep reinforcement learning in learning locomotion policies both in simulation and real world [27]–[30], fine-grained manipulation in real world [19], [20], [31]–[35]. However, challenges still remain in scaling DRL up in the domain of robotics. The first is sample efficiency, typically these methods require hundreds of thousands of trials in real world, which can be prohibitive; secondly, reinforcement learning is often guided by a reward signal, it might require enormous engineering effort to design such reward functions. In this thesis, we aim to address these two issues in the domain of robotics: we dramatically increase the sample efficiency by what we call residual reinforcement learning – RL algorithms is supposed to learn a residual policy to some base policy which can be easily obtained, which later we show in fact is a bias-variance trade-off; we also empirically validate that our method can be trivially extended to adopt to visual inputs, which users can specify goals and reward functions by just providing a snapshot of desired outcomes.

1.3 Dissertation outline and contributions

Though end-to-end learning has proven to be tremendously successful in many domains such as computer vision, natural language processing, playing video games, etc., it still remains challenging to apply such algorithms directly in real world robotic applications for reasons we discussed above. In this thesis, we study how to efficiently learn robotic skills by best exploiting structures and properties of different robotic problems. This has several benefits: there are already many mature methods handling control of kinematics, dynamics or other robotic specific problems, injecting these knowledge as priors will abstract the learning problem in a more structured way thus greatly improving sample efficiency of these learning algorithms; moreover it modulates the problem into a interpretable form so that learned skills has a better ability to transfer across tasks. Specifically, we organize this thesis as following:

- In chapter 2, we proposes a simple yet effective algorithm which we call residual reinforcement learning. We decompose a difficult control problem in real world into a part that can be solved efficiently by a easily obtained 'prior controller', and the residual part is solved with RL. The final control policy is a superposition of both signals. We evaluate our method on contact-rich robotic manipulation tasks such as inserting tight-fitting connectors and high-variant blocks; results show that our method can solve these tasks which are unsolvable by neither the prior controller nor RL alone. Furthermore, we extend our method to adopt to visual inputs and what we call natural rewards, the user will only need to specify his intention by taking a snapshot of desired outcome, our method can solve it directly in pixel space with a proper prior controller. Finally we show in theory that our method is in fact a bias-variance trade-off: prior controller injects bias in the final solution, however, reducing the variance to a

quantifiable extent, we can also guarantee stability of the final controller if the prior controller satisfy certain properties.

This chapter is composed of two previously published papers:

Residual reinforcement learning for robot control. Tobias Johannink*, Shikhar Bahl*, Ashvin Nair*, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, Sergey Levine. ICRA 2019.

Deep Reinforcement Learning for Industrial Insertion Tasks with Visual Inputs and Natural Rewards. Gerrit Schoettler*, Ashvin Nair*, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, Sergey Levine. IROS 2019.

- In chapter 3, we present a better action space for RL algorithm in learning high-precision robotic assembly skills. This is motivated by human heuristically mapping perceived force to control actions, which results in completing high-precision tasks in a fairly easy manner. Our approach combines RL with force/torque information by incorporating a proper operational space force controller; it greatly reduces the search space for RL, making the reinforcement learning process more structured and interpretable. Our method can be used in both inherently compliant and non-compliant robots; we tackle two specific use-cases: 1)deformable object manipulation 2)the open-source Siemens Robot Learning Challenge; both of them requires precise and delicate force-controlled robotic behavior. Results show that our method not only learns fast, also is the only method that solves high-precision assembly tasks we study in this chapter.

This chapter is composed of two previously published papers:

Deep reinforcement learning for robotic assembly of mixed deformable and rigid objects. Jianlan Luo, Eugen Solowjow, Chengtao Wen, Juan Aparicio Ojea, Alice M Agogino. IROS 2018.

Reinforcement learning on variable impedance controller for high-precision robotic assembly. Jianlan Luo, Eugen Solowjow, Chengtao Wen, Juan Aparicio Ojea, Alice M Agogino, Aviv Tamar, Pieter Abbeel. ICRA 2019.

- In chapter 4, we propose a method of learning large-scale grasping policies efficiently. We explicitly consider the gripper feature as an effective representation in robotic grasping, which we call the *Action Image* representation, this is built based on the idea that object features are invariant to translation in image space. *Action Image* represents a grasp proposal that can take multi-modal visual inputs and uses a deep convolution neural network to infer grasp quality. We show that *Action Image* representation can effectively extract reusable, salient features of grasping tasks that generalize across different objects and environments. The neural network is trained entirely with synthetic data generated from simulation, however could transfer to real world without loss of performance. We achieve 84% success rate on a variety of objects across different environments.

This chapter is composed of one previously published paper:

Action Image Representation: Learning Scalable Deep Grasping Policies with Zero

Real World Data. Mohi Khansari, Jianlan Luo, Daniel Kappler, Jeff Bingham, Mrinal Kalakrishnan. ICRA 2020.

- In chapter 5, we extend domain randomization, a method transfers simulation policies to real world, with an active learning component by explicitly considering geometric transformations in solving pose estimation tasks. Domain randomization based pose estimation trains a neural network to directly infer Cartesian poses from a 2D image of the scene. The network is trained on randomly generated synthetic images from a simulator with different varying factors such as lighting, textures, object scale etc. Our main idea is that active perception – moving the robot to get a better estimate of pose – can be trained in simulation and transferred to real using domain randomization. In our approach, the robot trains in a domain-randomized simulation how to estimate pose from a *sequence* of images. We show that our approach can significantly improve the accuracy of standard pose estimation in several scenarios: when the robot holding an object moves, when reference objects are moved in the scene, or when the camera is moved around the object.

This chapter is composed of one previously published papers:

Domain randomization for active pose estimation. Xinyi Ren, Jianlan Luo, Eugen Solowjow, Juan Aparicio Ojea, Abhishek Gupta, Aviv Tamar, Pieter Abbeel. ICRA 2019.

Finally, in chapter 6, we make our conclusions and propose future research directions.

Chapter 2

Learning residual policies for robotic control

2.1 Introduction

Robots in today’s manufacturing environments typically perform repetitive tasks, and often lack the ability to handle variability and uncertainty. Commonly used control algorithms, such as PID regulators and the computed torque method, usually follow predefined trajectories with little adaptive behavior. Many manufacturing tasks require some degree of adaptability or feedback to the environment, but significant engineering effort and expertise is required to design feedback control algorithms for these industrial robots. The engineering time for fine tuning such a controller might be similar in cost to the robot hardware itself. Being able to quickly and easily design feedback controllers for industrial robots would significantly broaden the space of manufacturing tasks that can be automated by robots.

Why is designing a feedback controller for many tasks hard with classical methods? While conventional feedback control methods can solve tasks such as path following efficiently, applications that involve contacts between the robot and its environment are difficult to approach with conventional control methods. Identifying and characterizing contacts and friction is difficult—even if a physical model provides reasonable contact behavior, identifying the physical parameters of a contact interaction accurately is very hard. Hence, it is often difficult to achieve adaptable yet robust control behavior, and significant control tuning effort is required as soon as these elements are introduced. Another drawback of conventional control methods is their lack of behavior generalization. Thus, all possible system behaviors must be considered a priori at design time.

Reinforcement learning (RL) methods hold the promise of solving these challenges because they allow agents to learn behaviors through interaction with their surrounding environments and ideally generalize to new scenarios that differ from the specifications at the control design stage. And RL can handle control problems that are difficult to approach with conventional controllers because the control goal can be specified indirectly as a term in a reward function

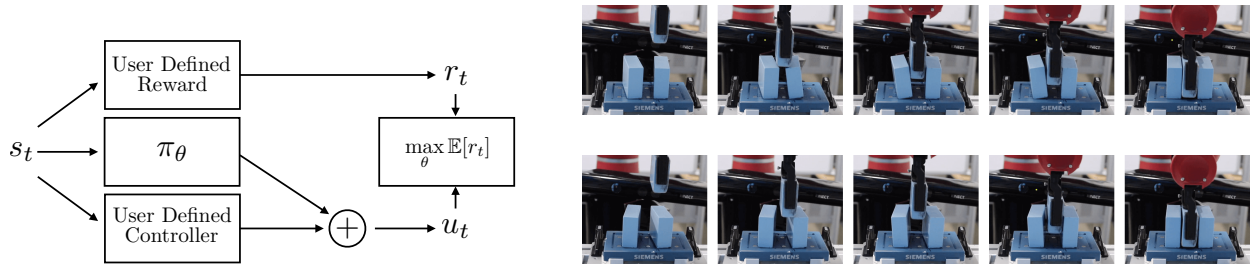


Figure 2.1: We train an agent directly in the real world to solve a model assembly task involving contacts and unstable objects. An outline of our method, which consists of combining hand-engineered controllers with a residual RL controller, is shown on the left. Rollouts of residual RL solving the block insertion task are shown on the right. Residual RL is capable of learning a feedback controller that adapts to variations in the orientations of the standing blocks and successfully completes the task of inserting a block between them. Videos are available at residualrl.github.io

and not explicitly as the result of a control action. All of these aspects are considered enablers for truly autonomous manufacturing systems and important for fully flexible lot-size one manufacturing [36].

However, standard RL methods require the robot learn through interaction, which can be unsafe initially, and collecting the amount of interaction that is needed to learn a complex skill from scratch can be very time consuming. Moreover, deep RL has thus far not seen wide adoption in the automation community due to several practical obstacles. Sample efficiency is one obstacle as mentioned above: tasks must be completed without excessive interaction time or wear and tear on the robot. Progress in recent years on developing better RL algorithms has led to significantly better sample efficiency, even in dynamically complicated tasks [37, 38], but remains a challenge for deploying RL in real-world robotics contexts. Another often-underappreciated major obstacle is goal specification: while prior work in RL assumes a reward signal to guide the optimization, it is often carefully shaped to allow the system to learn [39–41]. Obtaining such densely-shaped reward signals can be very challenging in most cases, as one must build additional such reward shaping models; and building these systems may require considerable domain knowledge and manual effort. In an ideal RL system, the agent should learn from rewards that are natural and easy for human to specify; this way both the RL agent and human can benefit from each other: human can express his intention in its simplest form to the RL agent while the RL agent can gain enough learning signals from such specified goals.

To address the first issue, we introduce a method we call residual reinforcement learning which essentially injects inductive bias into the RL learning process by introducing suitable control priors. Such problems possess structure that can be partially handled with conventional feedback control. The residual part of the control task, which is the part that must consider

contacts and external object dynamics, is solved with RL. The outputs of the conventional controller and RL are superposed to form the commanded control. This method combines conventional feedback control with deep RL methods and is illustrated in Figure 2.1. Our main motivation is a control approach that is suitable for real-world control problems in manufacturing, where the exploratory behavior of RL is a safety concern and the data requirements of deep RL can be expensive. We provide a thorough evaluation of our method on a block assembly task in simulation and on physical hardware. When the initial orientation of the blocks is noisy, our hand-designed controller fails to solve the task, while residual RL successfully learns to perform the task in under 3 hours. This suggests that we could usefully apply our method to practical manufacturing problems.

To address the second issue, we extend the aforementioned method to adopt to visual inputs and binary reward signal. In this case, one can specify his intention by showing a goal image to the robot, and construct a simple binary mechanism tells the robot if it's a success or failure during the learning process. This method gives human a great deal of flexibility and convenience to specify their intention while maintaining sample efficiency for the learning agent. In our experiments, we show that we can successfully complete real-world tight tolerance assembly tasks, such as inserting various industrial connectors and sockets with our method as shown in Figure 2.7. These reward signals require no extra engineering and are easy to specify and can be as simple as binary signals from successful electric connections. Additionally, beyond showing the feasibility of RL for solving these tasks, we evaluate multiple RL algorithms across three tasks and study their robustness to imprecise positioning and noise.

We also include an interpretation of our method as bias-variance trade off between the injected control prior and the RL controller. This chapter is organized as follows:

- Section 2.1 and 2.2 gives introduction and related work.
- Section 2.3 details our method.
- Section 2.4 describes experiments of our methods without visual inputs, instead full state information is handed to the robot.
- Section 2.5 extends our method to take visual inputs and natural rewards.
- Section 2.6 describes additional experiments on visual inputs and natural rewards.
- Section 2.7 proves that our method is actually a bias-variance trade-off between prior policy and RL policy.
- Section 2.8 concludes this chapter and points to direction of future work.

2.2 Related work

Reinforcement learning for robotics holds the promise of greater autonomy and reliability, which could be vital to improving our manufacturing processes beyond its current limitations. RL methods have been difficult to apply in robotics because of sample efficiency, safety, and stability issues. Still, RL has been used to allow robots to learn tasks such as learning robotic locomotion policies [27–29], playing table tennis [42], swinging up a cartpole and balancing a unicycle [43], grasping [44, 45], opening a door [33], and general manipulation tasks [20, 31, 32, 37]. The use of large function approximators, such as neural networks, in RL has further broadened the generality of RL [46]. Such techniques, called “deep” RL, have further allowed robots to be trained directly in the real world to perform fine-grained manipulation tasks from vision [20], open doors [47], play hockey [48], stack Lego blocks [49], use dexterous hands [50], and grasp objects [51]. RL, particularly deep RL, tends to be data-hungry; even learning simple tasks can require many hours of interaction. To bring these methods into factories and warehouses, they must be able to consistently solve complex tasks, multi-step tasks. One way to enable these methods to solve these complex tasks is to introduce prior human knowledge into the learning system, as our method does.

Prior work in RL has incorporated human prior knowledge for solving tasks in various ways. One such way is reward shaping [39], where additional rewards auxiliary to the real objective are included in order to guide the agent towards the desired behavior. Reward shaping can effectively encode a policy. For example, to train an agent to perform block stacking, each step can be encoded into the reward [40].

Often, intensive reward shaping is key for RL to succeed at a task and prior work has even considered reward shaping as part of the learning system [41]. Reward shaping in order to tune agent behavior is a very manual process and recovering a good policy with reward shaping can be as difficult as specifying the policy itself. Hence, in our method we allow for human specification of both rewards and policies—whichever might be more practical for a particular task.

Further work has incorporated more specialized human knowledge into RL systems. One approach is to use trajectory planners in RL in order to solve robotics tasks [52]. However, since the method optimizes trajectory following instead of the task reward, generalization can be difficult when aspects of the environment change. Other work has focused on human feedback [53–57] to inform the agent about rewards or to encourage safety. However, in many robotics tasks, providing enough information about the task through incremental human feedback is difficult.

Another way to include prior knowledge in RL is through demonstrations [34, 58–62]. This could be in the form of initializing a policy [34, 58], inferring reward functions using inverse reinforcement learning [63, 64] or improving the policy throughout the learning procedure [59, 60, 62]. These methods require multiple demonstrations, which can be difficult to collect, especially for assembly tasks, although learning a reward function by classifying goal states [65] may partially alleviate this issue. More recently, manually specifying

a policy and learning the residual task has been proposed [66], [67]. Demonstrations can substantially simplify the exploration problem as the agent begins training having already received examples of high-reward transitions and therefore knows where to explore [68]. However, providing demonstrations requires humans to be able to teleoperate the robot to perform the task. In contrast, our method only requires a conventional controller for motion, which ships with most robots.

Prior knowledge can also be induced through neural network architecture choices. Deep residual networks with additive residual blocks achieved state of the art results in many computer vision tasks by enabling training of extremely deep networks [69]. In RL, structured control nets showed improvement on several tasks by splitting the policy into a linear module and a non-linear module [70]. Prior work in adaptive flight control has also considered compensating linearized controllers with neural networks [71]. Most closely related to our work, residual policy learning concurrently and independently explores training a residual policy in the context of simulated long-horizon, sparse-reward tasks with environment variation and sensor noise [67]. Our work instead focuses on achieving practical real-world training of contact-intensive tasks.

Previous work has also tackled high precision assembly tasks, especially insertion-type tasks. One line of work focuses on obtaining high dimensional observations, including geometry, forces, joint positions and velocities [32], [72]–[74], but this information is not easily procured, increasing complexity of the experiments and the supervision required. Other work relies on external trajectory planning or very high precision control [73], [74], but this can be brittle to error in other components of the system, such as perception. We show how our method not only solves insertion tasks with much less information about the environment, but also does so under noisy conditions.

2.3 Method

Preliminaries

In this section, we set up our problem and summarize the foundations of classical control and reinforcement learning that we build on in our approach.

Problem Statement - System Theoretic Interpretation

The class of control problems that we are dealing with in this section can be viewed from a dynamical systems point of view as follows. Consider a dynamical system that consists of a fully actuated robot and underactuated objects in the robot’s environment. The robot and the objects in its environment are described by their states s_m and s_o , respectively. The robot can be controlled through the control input u while the objects cannot be directly controlled. However, the robot’s states are coupled with the objects’ states so that indirect control of s_o is possible through u . This is for example the case if the agent has large inertia

and is interacting with small parts as is common in manufacturing. The states of agent and objects can either be fully observable or they can be estimated from measurements.

The time-discrete equations of motion of the overall dynamical system comprise the robot and objects and can be stated as

$$s_{t+1} = \begin{bmatrix} s_{m,t+1} \\ s_{o,t+1} \end{bmatrix} = \begin{bmatrix} A(s_{m,t}) & 0 \\ B(s_{m,t}, s_{o,t}) & C(s_{o,t}) \end{bmatrix} \begin{bmatrix} s_{m,t} \\ s_{o,t} \end{bmatrix} + D \begin{bmatrix} u_t \\ 0 \end{bmatrix}, \quad (2.1)$$

where the states can also be subject to algebraic constraints, which we do not state explicitly here.

The type of control objectives that we are interested in can be summarized as controlling the agent in order to manipulate the objects while also fulfilling a geometric objective such as trajectory following. It is difficult to solve the control problem directly with conventional feedback control approaches, which compute the difference between a desired and a measured state variable. In order to achieve best system performance feedback control methods require well understood and modeled state transition dynamics. Finding the optimal control parameters can be difficult or even impossible if the system dynamics are not fully known.

In (2.1) the state transition matrices although $A(s_m)$ and $C(s_o)$ are usually known to a certain extent, because they represent rigid body dynamics, the coupling matrix $B(s_m, s_o)$ is usually not known. Physical interactions such as contacts and friction forces are the dominant effects that $B(s_m, s_o)$ needs to capture, which also applies to algebraic constraints, which are functions of s_m and s_o as well. Hence, conventional feedback control synthesis for determining u to control s_o is very difficult, and requires trial and error in practice. Another difficulty for directly designing feedback controllers is due to the fact that, for many control objectives, the states s_o need to fulfill conditions that cannot be expressed as deviations (errors) from desired states. This is often the case when we only know the final goal rather than a full trajectory.

Instead of directly designing a feedback control system, we can instead specify the goal via a reward function. These reward functions can depend on both s_m and s_o , where the terms that depend on s_m are position related objectives.

Interpretation as a Reinforcement Learning Problem

In reinforcement learning, we consider the standard Markov decision process framework for picking optimal actions to maximize rewards over discrete timesteps in an environment E . At every timestep t , an agent is in a state s_t , takes an action u_t , receives a reward r_t , and E evolves to state s_{t+1} . In reinforcement learning, the agent must learn a policy $u_t = \pi(s_t)$ to maximize expected returns. We denote the return by $R_t = \sum_{i=t}^T \gamma^{(i-t)} r_i$, where T is the horizon that the agent optimizes over and γ is a discount factor for future rewards. The agent's objective is to maximize expected return from the start distribution $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi} [R_0]$.

Unlike the previous section, RL does not attempt to model the unknown coupled dynamics of the agent and the object. Instead, it finds actions that maximize rewards, without making

any assumptions about the system dynamics. In this section, we use value-based RL methods. These methods estimate the state-action value function:

$$Q^\pi(s_t, u_t) = \mathbb{E}_{r_t, s_t \sim E, u_t \sim \pi} [R_t | s_t, u_t] \quad (2.2)$$

$$= \mathbb{E}_{r_t, s_{t+1} \sim E} [r_t + \gamma \mathbb{E}_{u_{t+1} \sim \pi} [Q^\pi(s_{t+1}, u_{t+1})]] \quad (2.3)$$

Equation 2.3 is a recursive version of Equation 2.2, and is known as the Bellman equation. The Bellman equation allows us to estimate Q via approximate dynamic programming. Value-based methods can be learned off-policy, making them very sample efficient, which is vital for real-world robot learning.

As mentioned in introduction part in this thesis, extracting an optimal action in aforementioned Bellman equation requires taking a maximum over Q values. In discrete case, this could be achieved by directly taking the *argmax* from the action vector; however in continuous case this could become intractable. Continuous RL algorithms such as deep deterministic policy gradients (DDPG) [75] additionally train a neural network to learn the maximizing action. In our study we specifically consider two related reinforcement learning algorithms that lend themselves well to real-world learning as they are sample efficient, stable, and require little hyperparameter tuning.

Twin Delayed Deep Deterministic Policy Gradients (TD3). Like DDPG, TD3 optimizes a deterministic policy [22] but uses two Q -function approximators to reduce value overestimation [76] and delayed policy updates to stabilize training.

Soft Actor Critic (SAC). SAC is an off-policy value-based reinforcement learning method which puts an additional entropy regularization term to encourage diverse solutions. [37].

Method: Residual Reinforcement Learning

Based on the analysis in Sec. 2.3, we introduce a control system that consists of two parts. The first part is based on conventional feedback control theory and maximize all reward terms that are functions of s_m . An RL method is superposed and maximizes the reward terms that are functions of s_o .

In most robotics tasks, we consider rewards of the form:

$$r_t = f(s_m) + g(s_o). \quad (2.4)$$

The term $f(s_m)$ is assumed to be a function, which represents a geometric relationship of robot states, such as a Euclidean distance or a desired trajectory. The second term of the sum $g(s_o)$ can be a general class of functions. Concretely, in our model assembly task, $f(s_m)$ is the reward for moving the robot gripper between the standing blocks, while $g(s_o)$ is the reward for keeping the standing blocks upright and in their original positions.

The key insight of residual RL is that in many tasks, $f(s_m)$ can be easily optimized a priori of any environment interaction by conventional controllers, while $g(s_o)$ may be easier

to learn with RL which can learn fine-grained hand-engineered feedback controllers even with friction and contacts. To take advantage of the efficiency of conventional controllers but also the flexibility of RL, we choose:

$$u = \pi_H(s_m) + \pi_\theta(s_m, s_o) \quad (2.5)$$

as the control action, where $\pi_H(s_m)$ is the human-designed controller and $\pi_\theta(s_m, s_o)$ is a learned policy parametrized by θ and optimized by an RL algorithm to maximize expected returns on the task.

Inserting (2.5) into (2.1) one can see that a properly designed feedback control law for $\pi_H(s_m)$ is able to provide exponentially stable error dynamics of s_m if the learned controller π_θ is neglected and the sub statespace is stabilizable. This is equivalent to maximizing (2.4) for the case f represents errors between actual and desired states.

The residual controller $\pi_\theta(s_m, s_o)$ can now be used to maximize the reward term $g(s_o)$ in (2.4). Since the control sequence (2.5) enters (2.1) through the dynamics of s_m and s_m is in fact the control input to the dynamics of s_o , we cannot simply use the a-priori hand-engineered feedback controller to achieve zero error of s_m and independently achieve the control objective on s_o . Through the coupling of states we need to perform an overall optimization of (2.5), whereby the hand-engineered feedback controller provides internal structures and eases the optimization related to the reward term $f(s_m)$.

Algorithm 1 Residual reinforcement learning

Require: policy π_θ , hand-engineered controller π_H .

- 1: **for** $n = 0, \dots, N - 1$ episodes **do**
 - 2: Initialize random process $\mathcal{N}$ for exploration
 - 3: Sample initial state $s_0 \sim E$.
 - 4: **for** $t = 0, \dots, H - 1$ steps **do**
 - 5: Get policy action $u_t = \pi_\theta(s_t) + \mathcal{N}_t$.
 - 6: Get action to execute $u'_t = u_t + \pi_H(s_t)$.
 - 7: Get next state $s_{t+1} \sim p(\cdot \mid s_t, u'_t)$.
 - 8: Store (s_t, u_t, s_{t+1}) into replay buffer $\mathcal{R}$.
 - 9: Sample set of transitions $(s, u, s') \sim \mathcal{R}$.
 - 10: Optimize θ using RL with sampled transitions.
-

Our method is summarized in Algorithm 1. The key idea is to combine the flexibility of RL with the efficiency of conventional controllers by additively combining a learnable parametrized policy with a fixed hand-engineered controller.

As our underlying RL algorithm, we use a variant of twin delayed deep deterministic policy gradients (TD3) as described in [22]. TD3 is a value-based RL algorithm for continuous control based off of the deep deterministic policy gradient (DDPG) algorithm [75]. We have found that TD3 is stable, sample-efficient, and requires little manual tuning compared to DDPG. We used the publicly available `rlkit` implementation of TD3 [77]. Our method is

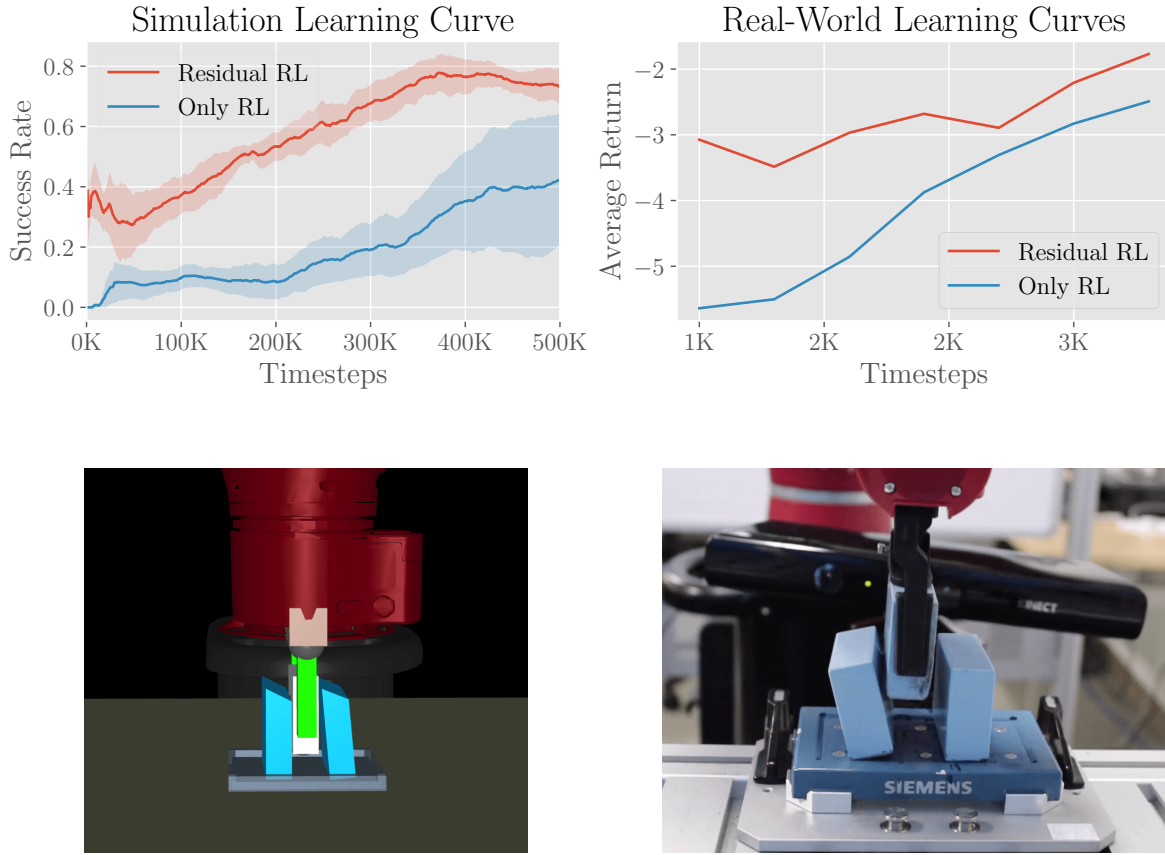


Figure 2.2: Block assembly task in simulation (left) and real-world (right). The task is to insert a block between the two blocks on the table without moving the blocks or tipping them over. In the learning curves, we compare our method with RL without any hand-engineered controller^[1]. In both simulation and real-world hardware experiments, we see that residual RL learns faster than RL alone, while achieving better performance than the hand-engineered controller.

independent of the choice of RL algorithm, and we could apply residual RL to any other RL algorithm.

2.4 Experiments

We evaluate our method on the task shown in Figure 2.2, both in simulation and in the real world. This section introduces the details of the experimental setup and provides an overview of the experiments.

Simulated Environment

We use MuJoCo [78], a full-featured simulator for model-based optimization considering body contacts, to evaluate our method in simulation. This environment consists of a simulated Sawyer robot arm with seven degrees of freedom and a parallel gripper. We command the robot with a Cartesian-space position controller.

Two blocks each with 3-DOF and one angled corner on the top are placed on a defined platform on the table in front of the robot. To allow block insertion between the standing blocks, a sufficiently large gap is defined (this gap represents the goal position). Both standing blocks can slide in x - and y -direction and topple around the y -axis. The agent receives the end effector position, the end effector forces and torques in relation to the x -, y - and z -axes, all block positions, and the goal position as the observation. In the robot's initial position one block for the insertion process is already picked up by the gripper claws and the gripper is located above the blocks on the tabletop.

We use a reward function

$$r_t = -\|x_g - x_t\|_2 - \lambda(\|\theta_l\|_1 + \|\theta_r\|_1) \quad (2.6)$$

where x_t is the current block position, x_g is the goal position, θ_l , θ_r are the angles with respect to the table (in the y -axis) of the left and right blocks, respectively. λ is a hyperparameter.

Real-World Environment

The real-world environment is largely the same as the simulated environment, except for the controller, rewards, and observations. We command the robot with a compliant joint-space impedance controller we have developed to be smooth and tolerant of contacts. The positioning of the block being inserted is similar to the simulation but the observation is estimated from a camera-based tracking system as we do not have access to ground truth position information.

Due to the blocks' slight weight and their capability of sliding in the plane (x, y) , the Sawyer is not able to measure contact forces regarding these axes. Therefore, we only add the end effector forces in z -direction to the observation space instead of observing the end effector forces and torques regarding to the x -, y - and z -axes. The reward function was slightly different, being defined as:

$$\begin{aligned} r_t = & -\|x_g - x_t\|_2 - \lambda(\|\theta_l\|_1 + \|\theta_r\|_1) \\ & -\mu\|X_g - X_t\|_2 - \beta(\|\phi_l\|_1 + \|\phi_r\|_1) \end{aligned} \quad (2.7)$$

where x_t is the current end effector position, x_g is the goal position, X_t describes the current position of both standing blocks, X_g their desired positions, θ_l , θ_r are the angles of the current orientation with respect to the table (in the y -axis) and ϕ_l and ϕ_r are the angles of the current orientation with respect to the z -axis of the left and the right block respectively λ , μ , and β are hyperparameters.

Overview of Experiments

In our experiments we evaluate the following research questions:

1. Does incorporating a hand-designed controller improve the performance and sample-efficiency of RL algorithms, while still being able to recover from an imperfect hand-designed controller?
2. Can our method allow robots to be more tolerant of variation in the environment?
3. Can our method successfully control noisy systems, compared to classical control methods?

Sample Efficiency of Residual RL

In this section, we compare our residual RL method with the human controller alone and RL alone. The following methods are compared:

1. Only RL: using the same underlying RL algorithm as our method but without adding a hand-engineered policy
2. Residual RL: our method which trains a superposition of the hand-engineered controller and a neural network policy, with RL

First, we compare residual RL and pure RL without a hand-engineered controller on the insertion task. Figure 2.2 shows in simulation and real-world that residual RL achieves a better final performance and requires less samples than RL alone, both in simulation and on physical hardware. Unlike residual RL, the pure RL approach needs to learn the structure of the position control problem from scratch, which explains the difference in sample efficiency. As samples are expensive to collect in the real world, residual RL is better suited for solving real-world tasks. Moreover, RL shows a broader spatial variance during training and needs to explore a wider set of states compared to residual RL, which can be potentially dangerous in hardware deployments.

Effect of Environment Variation

In automation, environments can be subject to noise and solving manufacturing tasks become more difficult as variability in the environment increases. It is difficult to manually design feedback controllers that are robust to environment variation, as it might require significant human expertise and tuning. In this experiment, we vary the initial orientation of the blocks during each episode and demonstrate that residual RL can still solve the task. We compare its performance to that of the hand-engineered controller.

To introduce variability in simulation, on every reset we sampled the rotation of each block independently from a uniform distribution $U[-r, r]$, $r \in \{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$.

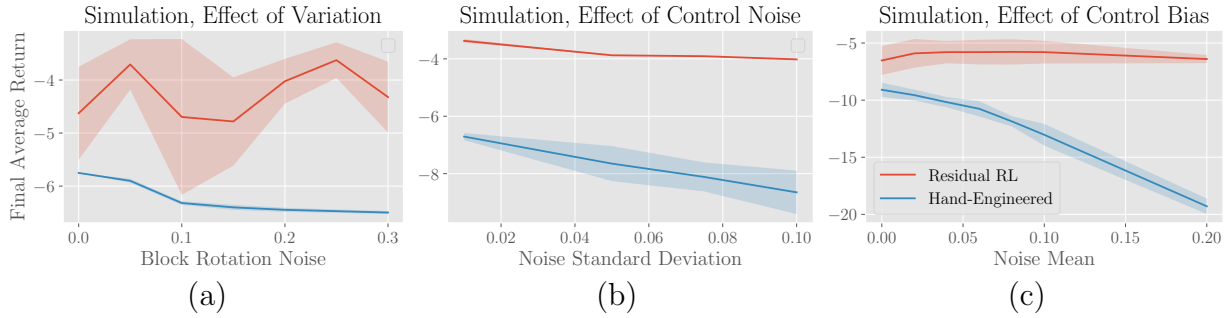


Figure 2.3: Simulation results for different experiments. In each plot, the final average return obtained by running the method for various settings of a parameter is shown. Plot (a) shows that residual RL can adjust to noise in the environment caused by rotation of the blocks in a range of 0 to 0.3 rad. In plot (b), residual RL finds robust strategies in order to reduce the effect of control noise, as the final average return is not greatly affected by the magnitude of noise. Plot (c) shows that residual RL can compensate for biased controllers and maintains good performance as control bias increases, while the performance of the hand-designed controller dramatically deteriorates with higher control bias.

Similarly, in the real world experiments, on every reset we randomly rotated each block to one of three orientations: straight, tilted clockwise, or tilted counterclockwise (tilt was $\pm 20^\circ$ from original position).

In previous set of experiments, both standing blocks were placed in their initial position without any position or orientation error. In this case, the hand-engineered controller performs well, as both blocks are placed such that there is a sufficiently large defined gap for insertion. However, once the initial orientation of the blocks is randomized, the gap between the blocks and the goal position does not afford directly inserting from above. Therefore, the hand-engineered controller struggles to solve the insertion task, succeeding in only 2/20 trials, while residual RL still succeeds in 15/20 trials. These results are summarized in Figure 2.4(a). Rollouts from the learned policy are included in 2.1. In this experiment, the agent demonstrably learns consistent small corrective feedback behaviors in order to slightly nudge the blocks in the right direction without tipping them over, a behavior that is very difficult to manually specify.

The result of this experiment showcases the strength of residual RL. Since the human controller specifies the general trajectory of the optimal policy, environment samples are required only to learn this corrective feedback behavior. The real-world learning curve for the experiment in Figure 2.4(b) shows that this behavior is gradually acquired over the course of eight thousand samples, which is only about three hours of real-world training time.

We further studied the effect of the block orientation changing after every reset in simulation. The results are shown in Figure 2.3(a). The simulation results show that the performance of the hand-engineered controller decreases as the block rotation angle increases,

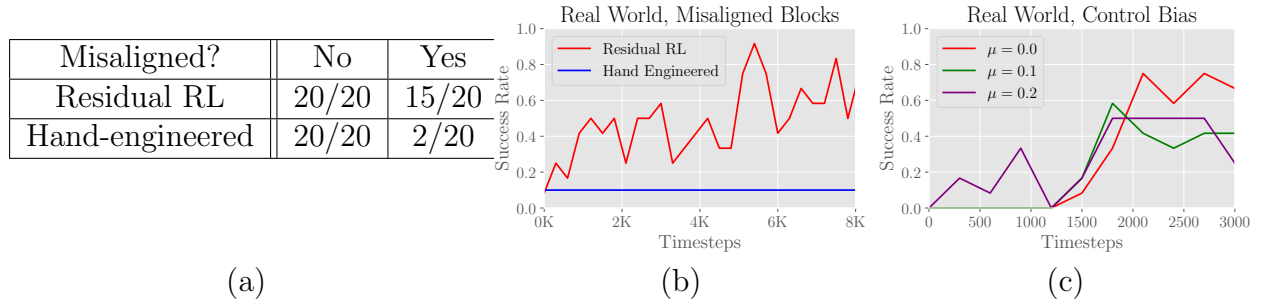


Figure 2.4: Outcome of our residual RL method in different experiments during the block assembly task in the real-world. Success rate is recorded manually by human judgment of whether the blocks stayed upright and ended in the correct position. Plot (a) compares the insertion success of residual RL and hand-designed controller depending on the block orientation during run time. Plot (b) shows the success rate of the insertion process during training, where on every reset the blocks are randomly rotated: straight, tilted clockwise, or tilted counterclockwise ($\pm 20^\circ$) and plot (c) shows the increasing success rate of our method for biased controllers as well even as control bias increases.

whereas our control method maintains a constant average performance over different variations.

Recovering from Control Noise

Due to a host of issues, such as defective hardware or poorly tuned controller parameters, feedback controllers might have induced noise. Conventional feedback control policies are determined a priori and do not adapt their behavior from data. However, RL methods are known for their ability to cope with shifting noise distributions and are capable of recovering from such issues.

In this experiment, we introduce a control noise, including biased control noise, and demonstrate that residual RL can still successfully solve the task, while a hand-engineered controller cannot. The control noise follows a normal distribution and is added to the control output of the system at every step:

$$u'_t = u_t + \mathcal{N}(\mu, \sigma^2) \quad (2.8)$$

To test tolerance to control noise, we set $\mu = 0$ and vary $\sigma \in [0.01, 0.1]$. In theory, RL could adapt to a noisy controller by learning more robust solutions to the task which are less sensitive to perturbations.

Furthermore, to test tolerance to a biased controller, we set $\sigma = 0.05$ and vary $\mu \in [0, 0.2]$. To optimize the task reward, RL can learn to simply counteract the bias.

In this experiment, we observe that residual RL is able to cope with actuator noise, including biased actuator noise. Quantitative results for simulation are shown in Figure 2.3(b)

and (c). In Figure 2.3(c) our method keeps the average return constant and correct for biased controllers even as control bias increases, whereas the hand-engineered controller cannot compensate biased input and its performance deteriorates as control bias increases. The same applies for adding control noise to the control output as shown in Figure 2.3(b).

For the hardware experiments, only biased actuator noise is investigated. These results are shown in Figure 2.4(c). These learning curves show that even as more control bias is introduced, training in the real world proceeds without significant issues. This result suggests the potential for RL to address practical issues in automation such as sensor drift.

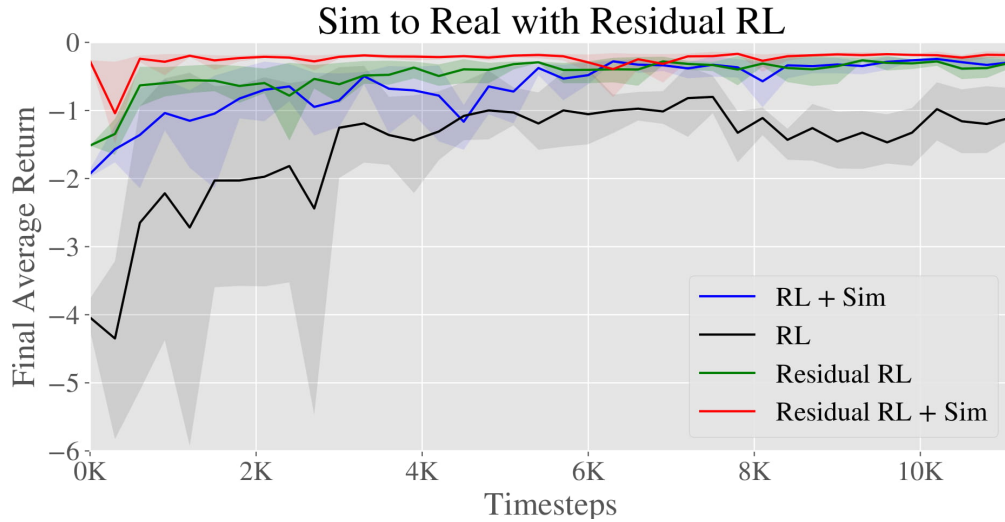


Figure 2.5: Real-world block insertion results using residual RL for sim-to-real transfer. “Sim” indicates that the real-world policy was initialized by reinforcement learning in simulation. Residual RL with simulation initialization successfully solves the task with little environment experience required.

Sim-to-Real with Residual RL

As an alternative to analytic solutions of real-world control problems, we can often instead model the forward dynamics of the problem (i.e., a simulator). With access to such a model, we can first find a solution to the problem with our possibly inaccurate model, and then use residual RL to find a realistic control solution in the real world.

In this experiment, we attempt the block insertion task with the side blocks fixed in place. The hand-engineered policy π_H in this case comes from training a parametric policy in simulation of the same scenario (with deep RL). We then use this policy as initialization for residual RL in the real world.

The result of the sim-to-real experiment is shown in Figure 2.5. In this experiment, each setting was run with three random seeds. Adding policy initialization from simulation

significantly speeds up both RL and residual RL. In particular, residual RL with policy initialization from simulation successfully solves the task extremely quickly: in under one thousand timesteps of environment interaction. This method poses a highly sample efficient, practical way to solve robotics problems with difficult contact dynamics.

2.5 Extension to visual inputs and natural rewards

As we mentioned in the introduction part of this chapter, another concerning question in reinforcement learning is to conveniently specify goal and associated rewards. To this end, we can extend our method to fulfill these requirements by adding visual inputs and natural rewards. In this case, we only need to specify the goal by taking a snapshot image of what we want the robot to achieve, and the reward could be calculated by means of measuring how far the current visual observation is from the desired one. This could be challenging since the pixel space of an image could be high-dimensional, hence directly calculating some sort of discrepancy between images could be intractable in most cases. However, in our method, the use of control prior largely mitigates this issue.

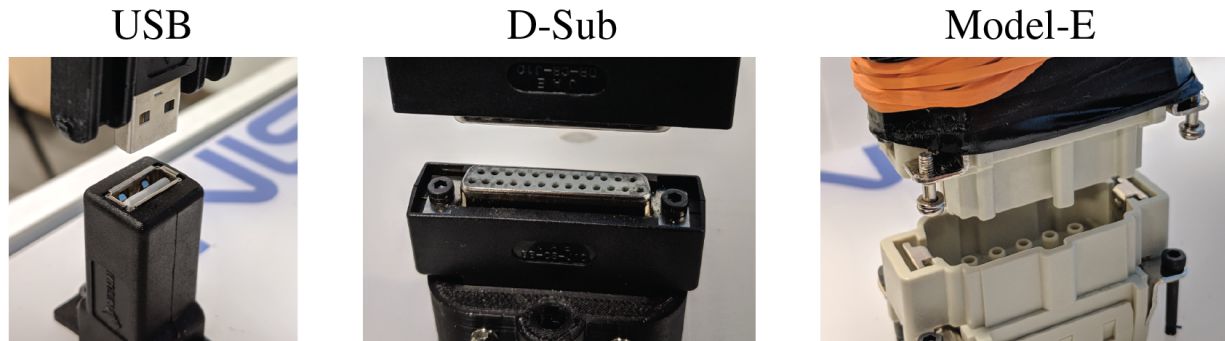


Figure 2.6: A close-up view of the three connector insertion tasks shows the contacts and low tolerances the agent must navigate to solve these tasks. These tasks require sub-millimeter precision without visual feedback.

We empirically evaluate learning methods on a set of electric connector assembly tasks, pictured in Figure 2.6. Connector plug insertions are difficult for two reasons. First, the robot must be very precise in lining up the plug with its socket. As we show in our experiments, errors as small as ± 1 mm can lead to consistent failure. Second, there is significant friction when the connector plug touches the socket, and the robot must learn to navigate under this constraint in order to insert the plug. Image sequences of successful insertions are shown in Figure 2.7, where it is also possible to see details of the gripper setup that we used to ensure a failure free, fully automated training process. In our experiments, we use a 7 degrees of freedom Sawyer robot with end effector control. The action signal u_t provides the relative end

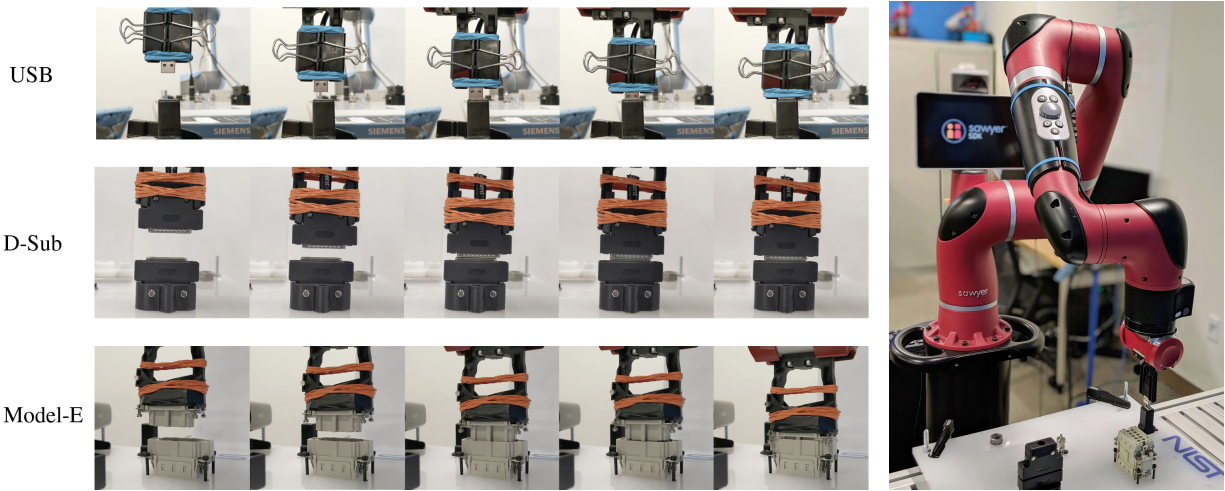


Figure 2.7: We train policies directly in the real world to solve connector insertion tasks from raw pixel input and without access to ground-truth state information for reward functions. Left: a rollout from a learned policy that successfully completes the insertion task for each connector is shown. Right: a full view of the robot setup. Videos of the results are available at industrial-insertion-rl.github.io

effector movement in Cartesian coordinates, the action space is 3-dimensional. The robot’s underlying internal control pipeline is illustrated in Figure 2.8.

To comprehensively evaluate connector assembly tasks, we experiment on a variety of connectors. Each connector offers a different challenge in terms of required precision and force to overcome friction. We chose to benchmark the controllers performance on the insertion of a USB connector, a U-Sub connector, and a waterproof Model-E connector manufactured by MISUMI. All the explored use cases were part of the IROS 2017 Robotic Grasping and Manipulation Competition [79], included as part of a task board developed by NIST to benchmark the performance of assembly robots.

Connectors

In the following we describe the used connectors, USB, D-Sub, and Model-E. The observed difficulty of the insertion increases in that order.

USB. The USB connector is a ubiquitous, widely-used connector and offers a challenging insertion task. Because the connector becomes smoother and therefore easier to insert over time due to wear and tear, we periodically replace the connector. Of the three tested connectors, the USB connector is the easiest.

D-sub. Inserting this connector requires aligning several pins correctly, and is therefore more sensitive than inserting the USB connector. It also requires more downward force due to a tighter fit.

Model-E. This connector is the most difficult of the three tested connectors as it contains several edges and grooves to align and requires significant downward force to successfully insert the part.

Experimental Settings

We consider three settings in our experiments in order to evaluate how plausible it is to solve these tasks with more convenient state representations and reward functions and to evaluate the performance of different algorithms changes as the setting is modified.

Visual In this experiment, we evaluate whether the RL algorithms can learn to perform the connector assembly tasks from vision without having access to state information. The state provided to the learned policy is a 32×32 grayscale image, such as shown in Figure 2.9. For goal specification, we use a goal image, avoiding the need for state information to compute rewards. The reward is the pixelwise ℓ_1 distance to the given goal image. Being able to learn from such a setup is compelling as it does not require any extra state estimation and many tasks can be specified easily by a goal image.

Sparse In this experiment, the reward is obtained by directly measuring whether the connection is alive and transmitting:

$$r = \begin{cases} 1, & \text{if insertion signal detected} \\ 0, & \text{else.} \end{cases} \quad (2.9)$$

This is the exact true reward for the task of connecting a cable, and can be naturally obtained in many manufacturing systems. As state, the robot is given the Cartesian coordinates of the end effector x_t , relative to the assumed goal location, and the vertical force f_z that is acting on the end effector. The state space is 4-dimensional. Providing relative position measurements allows to apply the same policy at a different goal location after recalibration.

Dense In this experiment, the robot receives a manually shaped reward based on the distance to the target location x^* . We use the reward function

$$r_t = \alpha \cdot \|x_t - x^*\|_1 + \frac{\beta}{(\|x_t - x^*\|_2 + \varepsilon)} + \varphi \cdot f_z, \quad (2.10)$$

where $0 < \varepsilon \ll 1$. The hyperparameters are set to $\alpha = -100$, $\beta = 0.002$, and $\varphi = -0.1$. When an insertion is indicated through a distance measurement, the sign of the force term flips, so that $\varphi = 0.1$ when the connector is inserted. This rewards the agent for pressing down after an insertion and showed to improve the learning process. The force measurements are calibrated before each rollout to account for measurement bias and to decouple the measurements from the robot pose. The state space is the same as in the sparse setting.

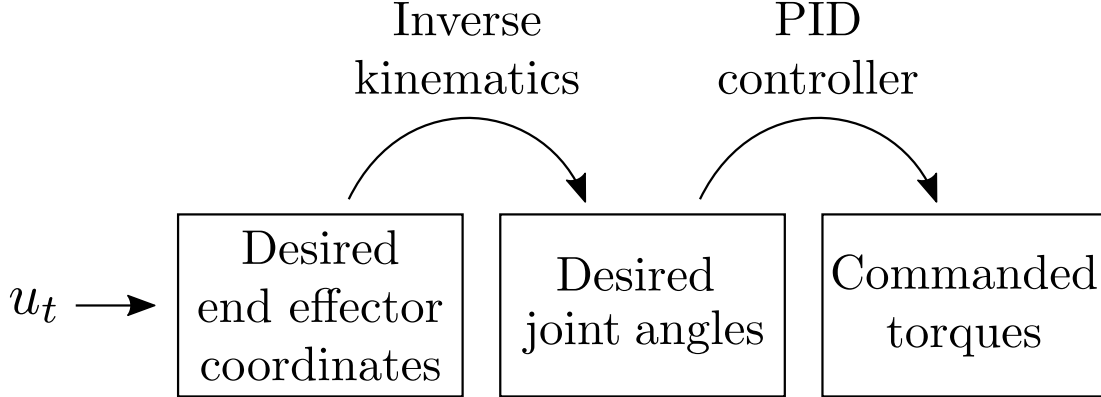


Figure 2.8: Illustration of the robot’s cascade control scheme. The actions u_t are computed at a frequency of up to 10 Hz, desired joint angles are obtained by inverse kinematics, and a joint-space impedance controller with anti-windup PID control commands actuator torques at 1000 Hz.

2.6 Experiments

We evaluate our method, which combines residual RL with easy-to-obtain reward signals, on a variety of connector assembly tasks performed on a real robot. In this section, we consider two types of natural rewards that are intuitive for users to provide: an image directly specifying a goal and a binary sparse reward indicating success. For both cases, we report success rates on tasks they solve. We aim to answer the following questions: (1) Can such trained policies provide comparable performance to policies that are trained with densely-shaped rewards? (2) Are these trained policies robust to small variations and noise?

Vision-based Learning

For the vision-based learning experiments, we use only raw image observations and ℓ_1 distance between the current image and goal image as the reward signal. Sample images that the robot received are shown in Figure 2.9. We evaluate this type of reward on all three connectors. In our experiments, we use 32×32 grayscale images.

The learning curves of the vision-based experiment are presented in Figure 2.10, the graphs show the final distance at the end of each 50 step long episode, plotted over the total number of environment steps throughout the training. Our experiments show that a successful and consistent vision-based insertion policy can be learned from relatively few samples using residual RL. In the early stages of the training, the human-engineered controller in residual RL guides the robot towards the point of insertion and due to the RL algorithm’s exploration noise, successful insertions are experienced much earlier in the training than with

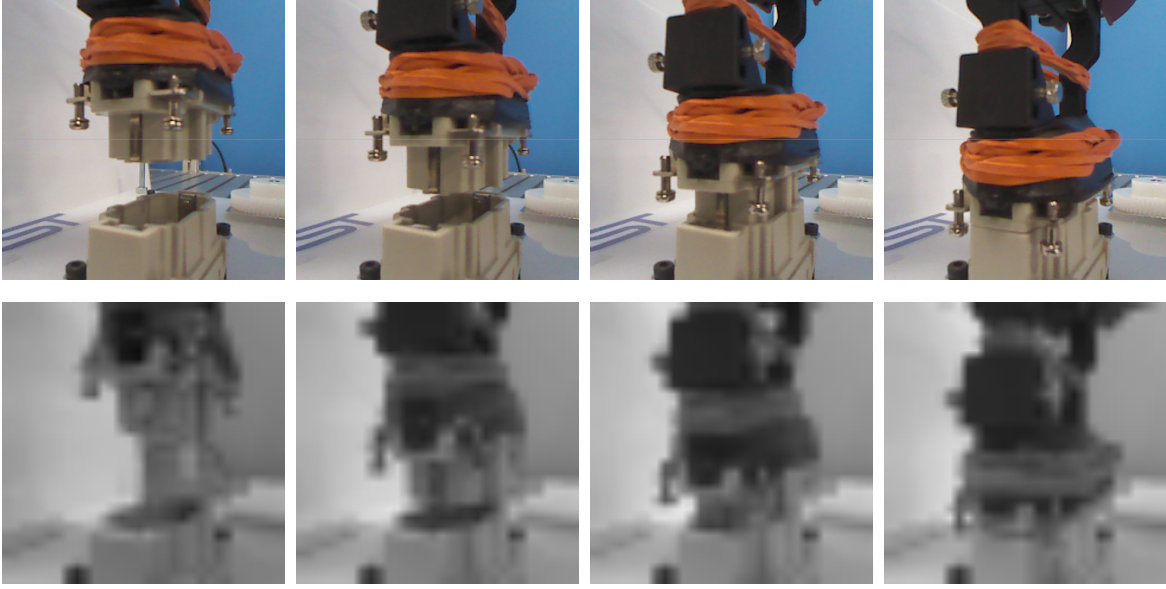


Figure 2.9: Successful insertion on the Model-E connector. The image-based RL algorithms receives only receives the 32×32 grayscale image as the observation.

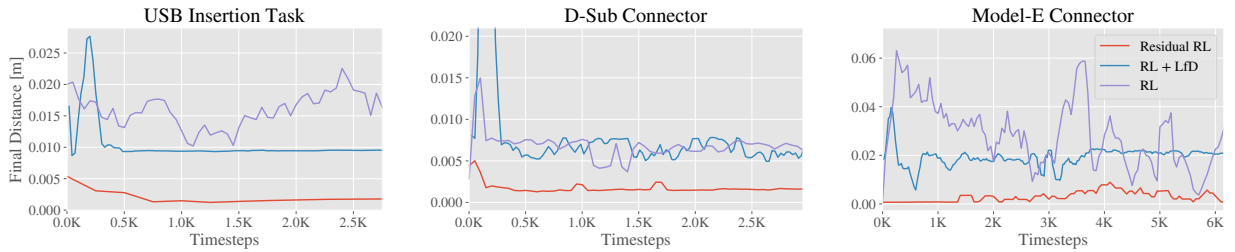


Figure 2.10: Resulting final mean distance during the vision-based training; the distance is calculated as pixel-wise L1 norm. One rollout amounts to 50 time steps. The comparison includes RL, residual RL, and RL with learning from demonstrations. Only residual RL manages to deal with the high-dimensional input and consistently solve all the tasks after the given amount of training. The other methods learn to move downwards, but often get stuck in the beginning of the insertion and fail to recover from unsuccessful attempts.

standard RL. During the training on the USB connector, the residual RL policy consistently improved in performance. On the D-Sub and Model-E connector, the residual RL policy became too deterministic and decreased in performance at certain times, but managed to recover and solved the task consistently at the end of the training. This result suggests that goal-specification through images is a practical way to solve these types of industrial tasks. Although image-based rewards are often very sparse and hard to learn from, in this case the

USB Connector		Goal	
		Perfect	Noisy
Pure RL	Dense	28%	20%
	Sparse, SAC	16%	8%
	Sparse, TD3	44%	28%
	Images, SAC	36%	32%
	Images, TD3	28%	28%
RL + LfD	Sparse	100%	32%
	Images	88%	60%
Residual RL	Dense	100%	84%
	Sparse, SAC	88%	84%
	Sparse, TD3	100%	36%
	Images, SAC	100%	80%
	Images, TD3	0%	0%
Human	P-Controller	100%	60%

Figure 2.11: Average success rate on the USB insertion task. Residual RL and RL + LfD solve the task consistently. Moreover, residual RL stays robust under $\pm 1\text{mm}$ noise.

distance between images corresponds to a relatively dense reward signal which is sufficient to distinguish the different stages of the insertion process.

Interestingly, during training with standard RL, the policy would sometimes learn to “hack” the reward signal by moving down in the image in front of or behind the socket. In contrast, the stabilizing human-engineered controller in residual RL provides sufficient horizontal control to prevent this. The initial controller also scaffolds the learning process, by providing a very strong initialization that requires the reinforcement learning algorithm to only learn the final phase of the insertion. This produces substantially better performance in conjunction with vision-based rewards.

Learning from Sparse Rewards

In the sparse reward experiment, we use the binary signal of the connector being electrically connected as the reward signal. This experiment is most applicable to electronic manufacturing settings where the electrical connection between connectors can be directly measured. We only evaluate the sparse reward setting on the USB connector, as it was straightforward to obtain the electrical connection signal.

In this experiment, we compare these methods on the USB insertion task with sparse rewards. The success rates during the training process are reported in Figure [2.13](#). All methods are able to achieve similar performance at the end of the training in the sparse setting. Learning from demonstrations showed to provide a very good initialization and,

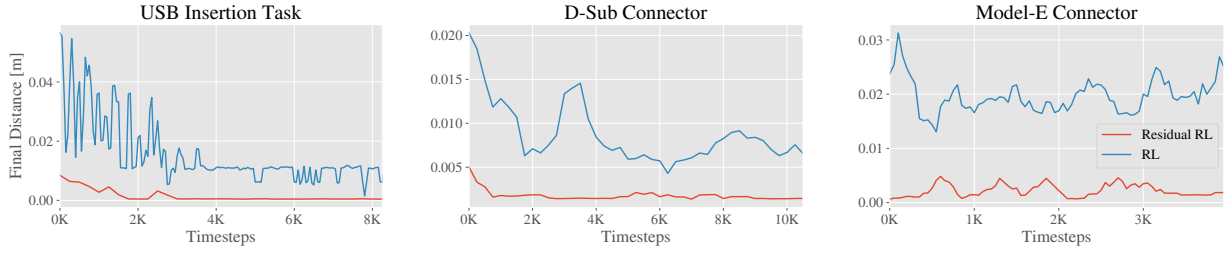


Figure 2.12: Plots of the final mean distance to the goal during the state-based training. Final distances greater than 0.01 m indicate unsuccessful insertions. Here, the residual RL approach performs noticeably better than pure RL and is often able to solve the task during the exploration in the early stages of the training.

after a short decrease in performance due to exploration, managed to reach a continuously succeeding policy in the least amount of training, shortly followed by Residual RL. After twice the amount of time, standard RL also achieved a high success rate during the training. This result shows that we can learn precise industrial insertion tasks from sparse rewards, which can often be obtained much more easily than a dense, shaped reward. In fact, prior work has found that the final policy for sparse rewards can outperform the final policy for dense rewards as it does not suffer from a misspecified objective [80].

Perfect State Information

After evaluating the tasks in the above settings, we further evaluate with full state information with a dense and carefully shaped reward signal, given in Eq. 2.10, that incorporates distance to the goal and force information. Evaluating in this setting gives us an “oracle” that can be compared to the previous experiments in order to understand how much of a challenge sparse or image rewards pose for various algorithms.

The results of the experiment with perfect state information and dense rewards is shown in Figure 2.12. In this case, residual RL outperforms standard RL by a large margin, although the shaped reward function allows standard RL to partially solve the task early in the training. However, the hand-designed shaped reward function makes it harder for the policy to actually perform the full insertion, potentially because the more complex reward landscape provides other competing goals to the policy. The final performance with sparse rewards on the USB insertion task is substantially better.

Robustness

For safe and reliable future usage, it is required that the insertion controller is robust against small measurement or calibration errors that can occur when disassembling and reassembling a mechanical system. To test robustness towards small goal perturbations, we artificially

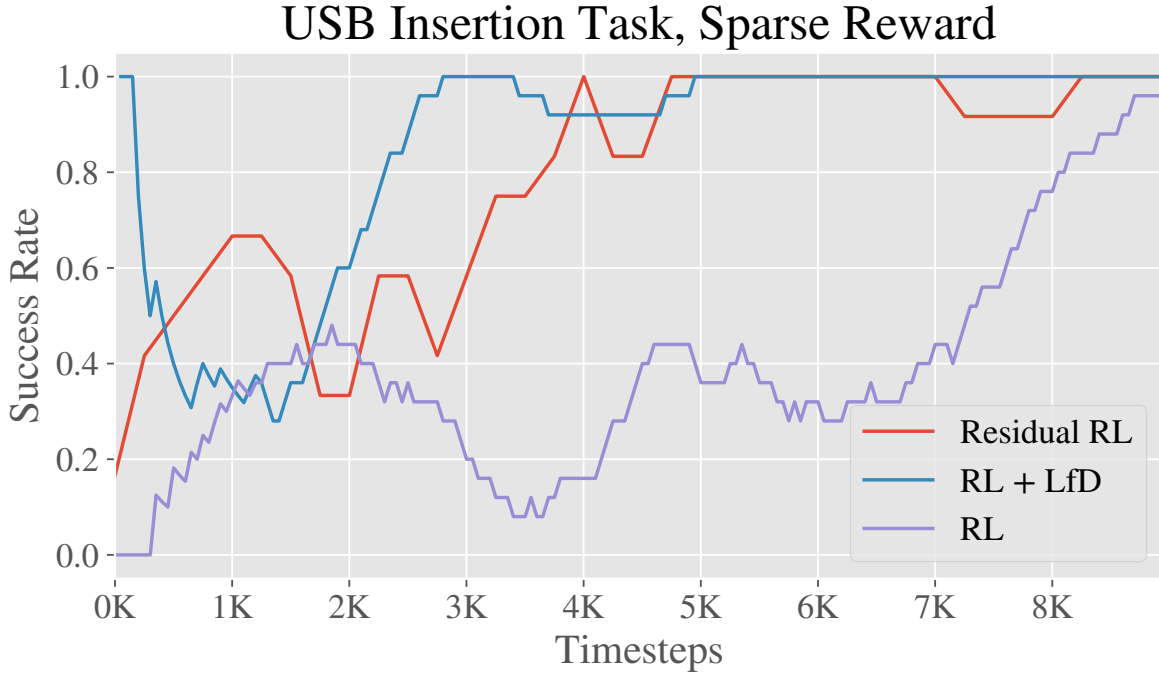


Figure 2.13: Learning curves for the training of the USB insertion task with a sparse reward. Success rate is shown; higher is better. One rollout amounts to 50 time steps. Residual RL and RL with learning from demonstrations both solve the task relatively quickly, while RL alone takes about twice as long to solve the task at the same performance.

perturb the goal by ± 1 mm in x and in y direction, such that the perturbed goals lie on the edges and corners of a square with side length 2 mm, centered on the exact goal location. This distribution results in 8 perturbed goal locations, on which we evaluate the trained policies. We measure the success rate over 25 policy executions, during which each of the perturbed goal locations is sampled at least 3 times. In addition to translational errors, rotational errors are very likely to occur as well when modifying a robot setup. However, since we calibrate the rotation of the goal location before each experiment, we did not notice rotational errors as the reason for unsuccessful insertions. As our policies only output translational movement and no rotations, a study of the robustness towards rotational errors is left out. In the previous set of experiments, the goal locations were known exactly. In this case, the hand-engineered controller performs well. However, once noise is added to the goal location, the deterministic P-controller struggles. All results of our robustness evaluations are listed in Figure 2.14 and Figure 2.11. In the presence of a ± 1 mm perturbation, the residual RL controller succeeds more often on the USB and D-Sub tasks, and significantly more often on the Model-E task, than the other methods. Unlike the P-controller, residual RL consistently solved the Model-E

task and overcame goal perturbations in 19/25 trials. On the USB task, residual RL was able to achieve successful insertions in 21/25 trials in the presence of goal perturbations. The agent demonstrably learns small but consistent corrective feedback behaviors in order to move in the right direction during the descent motion, a behavior that is very difficult to specify manually. This behavior illustrates the strength of residual RL. Since the human controller already specifies the general trajectory of the optimal policy, environment samples are only required to learn this corrective feedback behavior.

D-Sub Connector		Goal	
		Perfect	Noisy
Pure RL	Dense	16%	0%
	Images, SAC	0%	0%
	Images, TD3	12%	12%
RL + LfD	Images	52%	52%
Residual RL	Dense	100%	60%
	Images, SAC	100%	64%
	Images, TD3	52%	52%
Human	P-Controller	100%	44%
Model-E Connector		Goal	
		Perfect	Noisy
Pure RL	Dense	0%	0%
	Images, SAC	0%	0%
	Images, TD3	0%	0%
RL + LfD	Images	20%	20%
Residual RL	Dense	100%	76%
	Images, SAC	100%	76%
	Images, TD3	0%	0%
Human	P-Controller	52%	24%

Figure 2.14: We report average success out of 25 policy executions after training is finished for each method. For noisy goals, noise is added in form of ± 1 mm perturbations of the goal location. Residual RL, particularly with SAC, tends to be the best performing method across all three connectors. For the Model-E connector, only residual RL solves the task in the given amount of training time.

Exploration Comparison

One advantage of using reinforcement learning is the exploratory behavior that allows the controller to adapt from new experiences unlike a deterministic control law. The two RL algorithms we consider in this section, SAC and TD3, explore differently. SAC maintains a stochastic policy, and the algorithm also adapts the stochasticity through training. TD3

has a deterministic policy, but uses another noise process (in our case Gaussian) to inject exploratory behavior during training time. We compare the two algorithms, as well as when they are used in conjunction with residual RL, in order to evaluate the effect of the different exploration schemes.

All experiments were also performed using TD3 instead of SAC. The final success rates of these experiments are included in Figure 2.14 and Figure 2.11. Without added goal perturbations, SAC and TD3 perform comparably. However, TD3 is often substantially less robust. These results are likely explained by the exploration strategy of the two algorithms. TD3 has a deterministic policy and fixed noise during training, so once it observes some high-reward states, it quickly learns to repeat that trajectory. SAC adapts the noise to the correct scale, helping SAC stay robust to small perturbations, and because SAC learns the value function for a stochastic policy, it is able to handle some degree of additive noise effectively. In combination with residual RL, TD3 only managed to solve the sparse reward settings and did not train well in the vision-based setting. Here found that the outputted action of TD3 approaches the extreme values at the edge of the allowed action space, while SAC executed less extreme actions, which showed to produce better training results.

We analyze the performance of policies learned with residual RL, as well as other methods, based on their ability to achieve the task goal, as well as the distance of the final object location to the goal pose over the course of training. To study the robustness of the learned policies, we also evaluate them in conditions where the goal connector position is perturbed, in order to understand the tolerance of RL policies to imprecise object placement.

2.7 Interpretation as bias-variance trade-off

In this section, we analyze the proposed algorithm in more depth. We will show the learned policy can be interpreted as a bias-variance trade-off between the prior policy and the RL policy; the prior policy serves as a function regularization for RL policy. Our analysis is heavily built on the work [81].

Background and problem formulation

Let's start by analyzing the variance of a standard policy gradient RL algorithm. Denote a stochastic RL policy $\pi_\theta(a|s) : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, parameterized by θ . We want to maximize the expected accumulated reward $J(\theta)$:

$$J(\theta) = \sum_{t=1}^N \mathbb{E}_{(s_t, a_t) \sim p_{\pi_\theta}} [r(s_t, a_t)]. \quad (2.11)$$

Policy gradient style methods maximize the reward in equation 2.11 by estimating its

gradient from sampled trajectories:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(\tau) Q^{\pi_{\theta}}(\tau)] \quad (2.12)$$

$$\approx \sum_{i=1}^N \sum_{t=1}^T [\nabla_{\theta} \log \pi_{\theta}(s_{i,t}, a_{i,t}) Q^{\pi_{\theta}}(s_{i,t}, a_{i,t})], \quad (2.13)$$

where $Q^{\pi_{\theta}}(s, a) = \mathbb{E}_{\tau \sim \pi_{\theta}} [\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) | s_t = s, a_t = a]$, τ is a trajectory whose actions and states are sampled from the policy distribution $\pi_{\theta}(a|s)$ and the environment $\tau = (s_0, a_0, s_1, a_1, \dots)$.

The resulting policy gradient estimator can be a low-bias estimator if we can obtain a good estimate of Q function, however it can have high variance since we can only use a small number of sampled trajectories to calculate the expectation in (2.12).

We usually update the policy as follows:

$$\theta_{k+1} = \theta_k + \alpha \nabla_{\theta} J(\theta_k), \quad (2.14)$$

this update will bring the high variance from previous policy to the new policy.

We apply Tylor's expansion around π_{θ_k} to approximate $\pi_{\theta_{k+1}}$:

$$\pi_{\theta_{k+1}} = \pi_{\theta_k} + \alpha \frac{d\pi_{\theta_k}}{d\theta} \nabla_{\theta} J(\theta_k) + \mathcal{O}(\Delta\theta^2), \quad (2.15)$$

variance of policy at next time step can be seen below:

$$\text{var}[\pi_{\theta_{k+1}}] \approx \alpha^2 \frac{d\pi_{\theta_k}}{d\theta} \text{var}[\nabla_{\theta} J(\theta_k)] \frac{d\pi_{\theta_k}}{d\theta}^T, \quad (2.16)$$

this holds when $\alpha \ll 1$ which is almost always true when we choose learning rate. Note we are evaluating variance of updated policy with respect to its parameters θ , not the noise for exploration, so the variance of π_{θ_k} is zero.

Now let's write our mixed policy between prior policy and RL policy:

$$u_k(s) = \frac{1}{1+\lambda} u_{\theta_k}(s) + \frac{\lambda}{1+\lambda} u_{\text{prior}}(s), \quad (2.17)$$

where u_k is the mixed policy, u_{prior} is a control prior, π_{θ_k} is the control sampled from our stochastic RL policy $\pi_{\theta_k}(a|s)$.

Bias-variance trade-off

We now analyze bias and variance of the mixed policy.

Theorem 2.7.1. *The variance in (2.16) of mixed policy from equation (2.17) is reduced by a factor of $(\frac{1}{1+\lambda})^2$ when compared to a pure RL policy. However, it will introduce inductive bias proportional to the sub-optimality of the policy prior. If we denote the optimal policy to be*

π_{opt} , and $D(\cdot, \cdot)$ to be some divergence measure between two policies. We first measure the sub-optimality $D_{sub} = D_{TV}(\pi_{opt}, \pi_{prior})$, where $D_{TV}(\cdot, \cdot)$ means the total variation divergence. Then, the policy bias $D_{TV}(\pi_k, \pi_{opt})$ is bounded as follows:

$$D_{TV}(\pi_k, \pi_{opt}) \geq D_{sub} - \frac{1}{1+\lambda} D_{TV}(\pi_{\theta_k}, \pi_{prior}) \quad (2.18)$$

$$D_{TV}(\pi_k, \pi_{opt}) \leq \frac{\lambda}{1+\lambda} D_{sub} \quad \text{as } k \rightarrow \infty \quad (2.19)$$

Proof: Let a_{k+1} be the stochastic action sampled from pure RL policy $\pi_{\theta_{k+1}}(a|s)$, from equation 2.16, we can have following:

$$\text{var}[a_{k+1}|s] \approx \alpha^2 \frac{d\pi_{\theta_k}}{d\theta} \text{var}[\nabla_{\theta} J(\theta_k)] \frac{d\pi_{\theta_k}}{d\theta}^T; \quad (2.20)$$

plug equation 2.17 into equation 2.20, we get following:

$$\text{var}[\pi_{k+1}] = \text{var}_{\theta} \left[\frac{1}{1+\lambda} a_{k+1} + \frac{\lambda}{1+\lambda} u_{prior} | s \right] \quad (2.21)$$

$$= \frac{1}{(1+\lambda)^2} \text{var}_{\theta} [a_{k+1} | s] \quad (2.22)$$

$$\approx \frac{\alpha^2}{(1+\lambda)^2} \frac{d\pi_{\theta_k}}{d\theta} \text{var}[\nabla_{\theta} J(\theta_k)] \frac{d\pi_{\theta_k}}{d\theta}^T, \quad (2.23)$$

compare equation 2.23 with equation 2.20, we can clearly see the variance is reduced by the factor of $(\frac{1}{1+\lambda})^2$, this completes proof of the first part of our theorem.

Let's now analyze the bias of the mixed policy. We can obtain following relation by applying triangle inequality of metric:

$$D_{TV}(\pi_k, \pi_{opt}) \geq D_{TV}(\pi_{prior}, \pi_{opt}) - D_{TV}(\pi_{prior}, \pi_k), \quad (2.24)$$

We calculate the last term $D_{TV}(\pi_{prior}, \pi_k)$ as follows:

$$D_{TV}(\pi_{prior}, \pi_k) = \sup_{s,a} \left| \pi_{prior} - \frac{1}{1+\lambda} \pi_{\theta_k} \right| \quad (2.25)$$

$$= \frac{1}{1+\lambda} \sup_{s,a} |\pi_{\theta_k} - \pi_{prior}| \quad (2.26)$$

$$= \frac{1}{1+\lambda} D_{TV}(\pi_{\theta_k}, \pi_{prior}). \quad (2.27)$$

Then we can obtain the lower bound:

$$D_{TV}(\pi_k, \pi_{opt}) \geq D_{sub} - \frac{1}{1+\lambda} D_{TV}(\pi_{\theta_k}, \pi_{prior}). \quad (2.28)$$

Now we switch to the upper bound part, it's been proven that for certain classes of function approximators, RL can achieve asymptotic convergence to optimal policy [21]. So, as $k \rightarrow \infty$, $\pi_{\theta_k} \rightarrow \pi_{opt}$, we use this property to get an upper bound:

$$D_{TV}(\pi_{opt}, \pi_k) = \sup_{s,a} \left| \pi_{prior} - \frac{1}{1+\lambda} \pi_{\theta_k} \right| \quad (2.29)$$

$$= \frac{1}{1+\lambda} \sup_{s,a} |\pi_{opt} - \pi_{prior}| \quad \text{as } k \rightarrow \infty \quad (2.30)$$

$$= \frac{1}{1+\lambda} D_{TV}(\pi_{opt}, \pi_{prior}) \quad \text{as } k \rightarrow \infty \quad (2.31)$$

$$= \frac{\lambda}{1+\lambda} D_{sub} \quad \text{as } k \rightarrow \infty. \quad (2.32)$$

But in reality, π_{θ_k} will be influenced by π_{prior} , so it will not necessarily reach π_{opt} , hence the equality in equation 2.32 becomes inequality, we can therefore get an upper bound:

$$D_{TV}(\pi_k, \pi_{opt}) \leq \frac{\lambda}{1+\lambda} D_{sub} \quad \text{as } k \rightarrow \infty \quad (2.33)$$

□

Control regularization

In this section, we will look at the mixed policy from another perspective, we will show that the mixed policy can be actually formulated as a regularized optimization.

Lemma 2.7.2. *The mixed policy in equation 2.17 can be seen as the solution to the following regularized optimization problem:*

$$u_k(s) = \underset{u}{\operatorname{argmin}} \|u(s) - u_{\theta_k}\|_{\Sigma} + \lambda \|u(s) - u_{prior}(s)\|_{\Sigma} \quad (2.34)$$

Proof: From definition, $\pi_{\theta_k}(a|s) \sim \mathcal{N}(u_{\theta_k}, \Sigma)$, and the distribution of mixed policy is shifted by the prior policy as follow:

$$\pi_k(a|s) = \mathcal{N}\left(\frac{1}{1+\lambda} u_{\theta_k} + \frac{\lambda}{1+\lambda} u_{prior}, \Sigma\right) \quad (2.35)$$

$$= \frac{1}{c_N} \mathcal{N}(u_{\theta_k}, (1+\lambda)\Sigma) \cdot \mathcal{N}(u_{prior}, \frac{1+\lambda}{\lambda}\Sigma), \quad (2.36)$$

where c_N is normalizing constant. Denote Σ -Norm as $\|u_1 - u_2\|_{\Sigma} = (u_1 - u_2)^T \Sigma^{-1} (u_1 - u_2)$. Let's re-write equation 2.36 in terms of PDF.

$$\mathbb{P}(X(s)) = -c_1 \exp\left(-\frac{1}{2(1+\lambda)}\|X(s) - u_{\theta_k}(s)\|_{\Sigma}\right) \times \quad (2.37)$$

$$-c_1 \lambda^{\frac{k}{2}} \exp\left(-\frac{\lambda}{2(1+\lambda)}\|X(s) - u_{prior}(s)\|_{\Sigma}\right), \quad (2.38)$$

where c_1 is normalizing constant., $\mathbb{P}(X(s))$ represents the probability of taking action X from state s under policy in equation [2.17](#). We can further break down the PDF as following:

$$\mathbb{P}(X(s)) = c_2 \exp(-\|X(s) - u_{\theta_k}(s)\|_{\Sigma} - \lambda\|X(s) - u_{prior}(s)\|_{\Sigma}) \quad (2.39)$$

where c_2 is normalizing constant.

Usually, the RL policy is single-mode, so we choose the action when the PDF is maximized; which can be obtained when the argument of exponential is minimized. So the action can be seen as solution to the following optimization problem:

$$u_k(s) = \underset{u}{\operatorname{argmin}} \|u(s) - u_{\theta_k}\|_{\Sigma} + \lambda\|u(s) - u_{prior}(s)\|_{\Sigma}. \quad (2.40)$$

So we can conclude the mixed policy is the solution to equation [2.34](#) □

Insights from the bias-variance trade-off

From the analysis above, we can actually gain several interesting and useful insights. We can see the policy prior actually serves as function regularization to the pure RL policy and bias the final solution to it. If λ is too high, then the explorable set for RL can be constrained so that it could not reach (local) optimal policy. On the other side, if the (local) optimal policy is within explorable region, then the final policy can reach the optimal solution at a faster speed.

From the bias-variance trade-off analysis, we can see that we could lower the variance during learning by a certain factor; however at the cost injecting bias which can be quantified. This gives us guarantee on the performance of final learned controller.

2.8 Discussion

In this section we study the combination of conventional feedback control methods with deep RL. We presented a control method that utilizes conventional feedback control along with RL to solve complicated manipulation tasks involving friction and contacts with unstable objects. We believe this approach can accelerate learning of many tasks, especially those where the control problem can be solved in large part by prior knowledge but requires some model-free reasoning to solve perfectly. Our method extends conventional feedback control and allows us to automatically solve problems from data that would otherwise require human experts

to design solutions for. In this sense, our method is related to differentiable programming [82] or automated algorithm design [83]. Our results demonstrate that the combination of conventional feedback control and RL can circumvent the disadvantages of both and provide a sample efficient controller that can cope with contact dynamics.

We also showed that we can learn insertion policies with raw image observations with either binary outcome-based rewards, or rewards based on goal images. We conducted a series of experiments for various connector type assemblies, and demonstrated the feasibility of our method under challenging conditions, such as noisy goal specification and complex connector geometries. Reinforcement learning algorithms that can automatically learn complex assembly tasks with easy-to-specify reward functions have the potential to automate a wide range of assembly tasks, making this technology a promising direction forward for flexible and capable robotic manipulators.

Future direction includes a principled way to design prior controllers for different problems, and investigating properties of the final controller when the prior controller satisfy certain criterion.

Chapter 3

Learning complex manipulation with action space representation

3.1 Introduction

Today, industrial robots deployed across various industries are mostly doing repetitive tasks. The overall task performance hinges on the accuracy of their controllers to track pre-defined trajectories. To this end, endowing these machines with a greater level of intelligence to autonomously acquire skills is desirable. The main challenge is to design adaptable, yet robust, control algorithms in the face of inherent difficulties in modeling all possible system behaviors and the necessity of behavior generalization. Reinforcement learning (RL) methods hold promises for solving such challenges, because they promise agents to learn behaviors through interaction with their surrounding environments and ideally generalize to new unseen scenarios [20], [21], [84], [85].

In this chapter, we study how an RL algorithm will benefit from a better action space representation in settings involving high-precision robotic assembly. Specifically, we tackle two such problems: 1) assembly of a rigid peg into a deformable hole, where the diameter of the hole is smaller than that of the peg; and the robot is non-compliant which only provides position and velocity control interface; 2) assembly of high-precision work-pieces, in this case robot is compliant, we can access torque control interface.

In both tasks, the required precision exceeds the robot position controller’s accuracy. In real manufacturing, human labor can accomplish such high-accuracy complex tasks in a fairly easy manner. For example, a peg in hole insertion is achieved by “feeling” the contacts. This can be achieved with heuristics based on force feedback, for instance by probing the hole before inserting or moving the peg around the surface to search for the insertion point. Hence, designing robust strategies by properly processing observations is more desirable than estimating perfect physical dynamics. RL allows to find control policies automatically for problems where traditionally heuristic have been used. The question arises *how do we properly integrate observed force information into reinforcement learning process to produce*

desirable behaviors?

RL is a method for learning such reactive policies automatically, through trial and error interaction in the domain, guided only by a reward signal that specifies how well the robot is performing the task. In practice, RL requires an informative reward signal to work effectively, which can be hard to design automatically. With sparse reward that just specifies successful task completion, RL is prone to getting stuck in local optima. However operational space control could mitigate this problem by specifying high-level goals in task-space. [86]–[89]. One particular way to achieve this is to combine RL with a proper operational force controller, this has the benefit of implicitly shaping control actions so that policies only search the space where a “good” solution exists for our problems.

Furthermore, combining RL with such force controllers is highly relevant to industrial applications if we want such robots to learn contact-rich manipulation skills. Currently, most policy search algorithms for contact-rich assemblies are implemented on inherently compliant robot arms such as the PR2, the Kuka iiwa or Rethink Robotics’ Sawyer [90]–[92]. These robot arms have either passive compliance through spring mechanism in motors or have the ability to measure and command joint torques. These properties enable safe physical interaction of the robot with its environment, and joint torque readings can be encoded as features in learning algorithms to describe contact situations. Unfortunately, this is only of limited use for industrial applications because industrial robots are in general not compliant and offer only velocity and position control, but no torque control. However, they can often be equipped with a wrist force-torque sensor. While this still does not provide the ability to command joint torques, it opens the possibility for admittance force-torque control in task space, which introduces a principled way for such non-compliant robots to learn contact-rich tasks.

The remainder of this paper is organized as follows: section 3.2 introduces related work; section 3.3 describes proposed method; section 3.4 presents our experiments; section 3.5 discuss limits and future work.

3.2 Related work

Recent advances in RL have gained great success in solving a variety of problems from playing video games [24], [75], [93] to robotic locomotion [23], [27], [94]–[96], manipulation [20], [42], [73], [97]–[102]. Reinforcement learning can be distinguished in model-based methods and model-free methods [21], [84], [85]. While model-based policy search is computationally more expensive than model-free methods, it requires less data to solve a task. Recent progresses in the area of Deep Neural Networks suggest to also deploy them for parametrizing policies and other functions in RL methods [20], [23], [103]. This is often referred to as Deep Reinforcement Learning (DRL).

A recently developed model-based reinforcement learning algorithm called guided policy search (GPS) provided new insights into training end-to-end policy for solving contact-rich manipulation problems [20], [48], [95], [103], [104]; however; this method is not suitable for

this high-precision setting because it has no means of avoiding local optima by its formulation. There are also approaches tackling this problem by explicitly modeling contact dynamics [105]–[109]. Inoue et al. [110] use LSTM to learn two separate policies for finding and inserting a peg into a hole; however, their methods require several pre-defined heuristics, and also the action space is discrete.

Thomas et al. [52] combine RL with a motion planner to shape state cost in high-precision settings. This method essentially learns a trajectory following torque controller, and assumes access to a trajectory planner that could roughly avoid local optima. They also encode such planned reference into a neural network with attention mechanism, they show good generalization results in simulation.

3.3 Method

Preliminaries

We consider all tasks here that can be described as moving already-grasped objects to their goal position. This is the most common setting in today’s manufacturing. The success of such tasks can be measured as minimizing the distance between objects and their goal positions. We make no particular assumptions about encountered dynamics during tasks especially contacts. These need to be learned by the robot from various interaction with its environment. Let $\mathbf{x}_t$ and $\mathbf{u}_t$ denote robot states and actions respectively ; $\ell(\mathbf{x}_t, \mathbf{u}_t)$ be the cost function related to the task, T be the time horizon of a task. Our problem can be formulated as

$$\begin{aligned} \min_{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_T} \quad & \sum_{t=1}^T \ell(\mathbf{x}_t, \mathbf{u}_t) \\ \text{s.t.} \quad & \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \quad t = 1, 2, \dots, T-1, \end{aligned}$$

where f governs (unknown) system dynamics, $\mathbf{x}$, $\mathbf{u}$ can also be subject to other algebraic constraints.

We consider our control action $\mathbf{u}$ to be operational force controller $\mathcal{F}_{tip} = [F_x, F_y, F_z, M_x, M_y, M_z]$. They represent desired force/torque or impedance in operational space, our goal is to optimize them through reinforcement learning. We make no particular assumption about the implementation of such controller; neither the specific reinforcement learning algorithm.

Operational Space Force Controller

There are various ways to implement such controller in operational space with or without a wrist force/torque sensor [86] [88] [111]. We present two such force controllers depending on whether a wrist force/torque sensor is available.

For setup in Figure 3.1, a wrist force/torque sensor is mounted on the robot’s end-effector; we feed Cartesian-space force/torque sensor signals to the robot’s Cartesian-space velocity control loop. Specifically, denote τ as the desired force/torque vector in tool space; this includes three forces and three torques in three corresponding Cartesian axes. $\hat{\tau}$ denotes the measured force/torque vector from the robot’s wrist sensor. We apply a PD feedback controller to the difference between τ and $\hat{\tau}$, and feed the resulting control to the Cartesian-space velocity command interface of the robot. This involves calculating the inverse Jacobian matrix of the robot and converting Cartesian-space velocity into joint-space velocity. We set this admittance controller as output of reinforcement learning algorithm.

For setup in Figure 3.5, no force/torque sensor is available; however we have access to joint torque readings. We can convert operational space force into joint space using robot Jacobian, then close the control loop in joint space using joint torque feedback. Let $\mathcal{F}_{tip}$ be the desired wrench on the end-effector, we can then express the control law in joint space as

$$M(q)\ddot{q} + c(q, \dot{q})\dot{q} + g(q) + J^T(q)\mathcal{F}_{tip} = \tau, \quad (3.1)$$

where q represents joint angles in generalized coordinates, $M(q)$ is the inertia, $c(q, \dot{q})$ is the Coriolis matrix, $g(q)$ are gravitational forces, $J(q)$ is the Jacobian, τ is the torque vector applied to manipulators’ joints. In many force control tasks, robots move slowly, hence we ignore acceleration and velocity terms in Eq. 3.1. For a 7-DOF Sawyer manipulator arm that we consider in this paper, we can also project torques to its non-empty nullspace. Denoting the nullspace torque vector as τ_{null} , joint space control law is:

$$\tau = g(q) + J^T(q)\mathcal{F}_{tip} + [I - J^T(q)J^{T\dagger}(q)]\tau_{null}, \quad (3.2)$$

where $J^{T\dagger}(q)$ is the pseudo-inverse of $J^T(q)$. The control law in Eq. 3.2 is appealing and simple, but it would generate undesirable and dangerous motion without enough resistance provided by the environment. In our experimental setup, we do not assume in-contact situations of objects being assembled, there is a relative open free-space that the robot needs to move through; directly applying Eq. 3.2 would result in continuous acceleration. To mitigate this issue, in all our experiments, we wrap a position loop with small gains around the controller in Eq. 3.2:

$$\tau = g(q) + J^T(q)\mathcal{F}_{tip} + [I - J^T(q)J^{T\dagger}(q)]\tau_{null} \quad (3.3)$$

$$+ K_{qp}(q - q^*) + K_{qd}(\dot{q} - \dot{q}^*), \quad (3.4)$$

where K_{qp} and K_{qd} are diagonal gain matrices with small entries, q and $\dot{q}$ are current joint positions and velocities, q^* and $\dot{q}^*$ are the desired ones obtained via inverse kinematics from end-effector pose. Aforementioned $\mathcal{F}_{tip}$ will be calculated by an RL controller.

Model-Based Reinforcement Learning Algorithm: iLQG

The specific model-based reinforcement learning algorithm that we consider here is iterative Linear-Quadratic-Gaussian (iLQG) [95], [113]. It is sample efficient and convenient second-order methods are available to solve it quickly [114]. Let $\omega = \{\mathbf{x}_1, \mathbf{u}_1, \dots, \mathbf{x}_T, \mathbf{u}_T\}$ denote

a trajectory, such that $p(\omega) = p(\mathbf{x}_1) \prod_{t=1}^T p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)p(\mathbf{u}_t|\mathbf{x}_t)$, $\ell(\omega) = \sum_{t=1}^T \ell(\mathbf{x}_t, \mathbf{u}_t)$ denotes the cost along a single trajectory ω ; where $\mathbf{x}$ typically consists of joint angles, end-effector pose and their time derivatives. We wish to minimize this cost; the goal is to minimize the expectation $E_{p(\omega)}[\ell(\omega)]$ over trajectory ω by iteratively optimizing linear-Gaussian controllers and re-fitting linear-Gaussian dynamics. This algorithm iteratively linearizes the dynamics around the current nominal trajectory, constructs a quadratic approximation of the cost, computes the optimal actions with respect to this approximation of the dynamics and cost by dynamic programming, and forward runs resulting actions to obtain a new nominal trajectory. We use subscripts, e. g. $\ell_{\mathbf{xut}}$ to denote derivatives with respect to vector $[\mathbf{x}_t; \mathbf{u}_t]$. Under the dynamics model and the cost function described in this section, we can write the Q-function and value function as

$$\begin{aligned} V(\mathbf{x}_t) &= \frac{1}{2} \mathbf{x}_t^\top V_{\mathbf{x}, \mathbf{x}t} \mathbf{x}_t + \mathbf{x}_t^\top V_{\mathbf{x}t} + \text{const} \\ Q(\mathbf{x}_t, \mathbf{u}_t) &= \frac{1}{2} [\mathbf{x}_t, \mathbf{u}_t]^\top Q_{\mathbf{xu}, \mathbf{xut}} [\mathbf{x}_t; \mathbf{u}_t] + [\mathbf{x}_t; \mathbf{u}_t]^\top Q_{\mathbf{xut}} + \text{const}. \end{aligned}$$

We can solve for V and Q with a recurrence that can be computed backwards through time starting from the last time step $t = T$:

$$\begin{aligned} Q_{\mathbf{xu}, \mathbf{xut}} &= \ell_{\mathbf{xu}, \mathbf{xut}} + f_{\mathbf{xut}}^\top V_{\mathbf{x}, \mathbf{x}t+1} f_{\mathbf{xut}} \\ Q_{\mathbf{xut}} &= l_{\mathbf{xut}} + f_{\mathbf{xut}}^\top V_{\mathbf{x}t+1} \\ V_{\mathbf{x}, \mathbf{x}t} &= Q_{\mathbf{x}, \mathbf{x}t} - Q_{\mathbf{u}, \mathbf{x}t}^\top Q_{\mathbf{u}, \mathbf{u}t}^{-1} Q_{\mathbf{u}, \mathbf{x}t} \\ V_{\mathbf{x}t} &= Q_{\mathbf{x}t} - Q_{\mathbf{u}, \mathbf{x}t}^\top Q_{\mathbf{u}, \mathbf{u}t}^{-1} Q_{\mathbf{u}t}. \end{aligned} \tag{3.5}$$

This actually leads to the maximum entropy LQR objective whose solution is a time-varying linear-Gaussian controller $p(\mathbf{u}_t|\mathbf{x}_t)$:

$$\min_{p(\mathbf{u}_t|\mathbf{x}_t)} \sum_{t=1}^T E_{p(\mathbf{u}_t|\mathbf{x}_t)} [\ell(\mathbf{x}_t, \mathbf{u}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))]$$

The entropy term in the objective function above encourages exploration to acquire diverse samples for fitting dynamics. This objective can be optimized by setting $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t, \mathbf{C}_t)$, and $\mathbf{C}_t = Q_{\mathbf{u}, \mathbf{u}t}^{-1}$. We brief the overall scheme in Alg. [2](#).

3.4 Experiments

Non-Compliant Robot arm with Deformable Objects

Problem statement

Consider a robot arm without the possibility of joint torque sensing or joint torque control as in Fig. [3.1](#). The robot has no inherent compliance and permits only position and velocity

Algorithm 2 Force-based RL controllers

- 1: **for** iteration $k \in \{1, \dots, K\}$ **do**:
 - 2: Train local RL controller using iLQG, where $\mathbf{u}_t$ is set as operational space force controller
 - 3: Project calculated desired force to joint space according to Chapter 3.3 depending on robot type
-

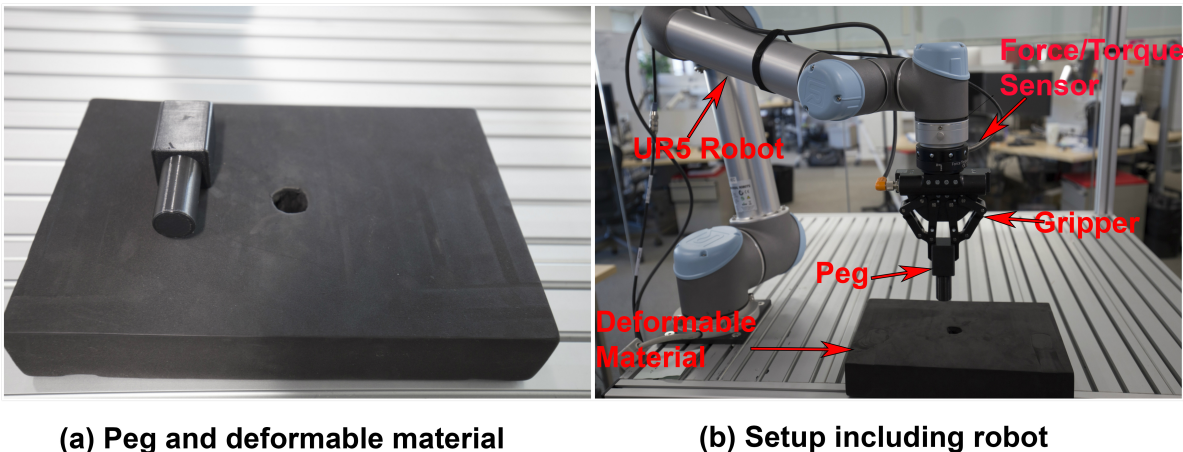


Figure 3.1: Experimental setup for peg insertion task.

control. However, the robot arm is equipped with a wrist mounted force-torque sensor, whose signals are available. Moreover, the joint angles of the robot are accessible as well. The robot end-effector holds a rigid peg whose position and orientation relative to the end-effector are known with a certain degree of uncertainty. Moreover, the grasp is not tight and the peg exhibits some bounded motion relative to the end-effector whenever the peg is in contact with the environment. A work piece with a hole is placed in the work space of the robot. The work piece consists of a deformable material with nonlinear material properties. The diameter of the hole is smaller than the diameter of the peg. The position of the whole relative to the robot frame is known, but also exhibits uncertainty.

The goal of the peg-in-hole problem is to control the robot in order to insert the rigid peg into the deformable hole. It is challenging to design a feedback control law for achieving this task due to the unknown contact mechanics. Figure 3.2 illustrates the assembly situation. In Figure 3.2 (a) the peg is successfully inserted into the elastic hole. The deformation poses challenges to the insertion tasks, because unlike rigid pairings, material yields if the robot pushes the peg into it as shown in Fig Fig. 3.2 (b). The key to escape from this local optimum is to properly adjust force/torque on the end-effector promptly.

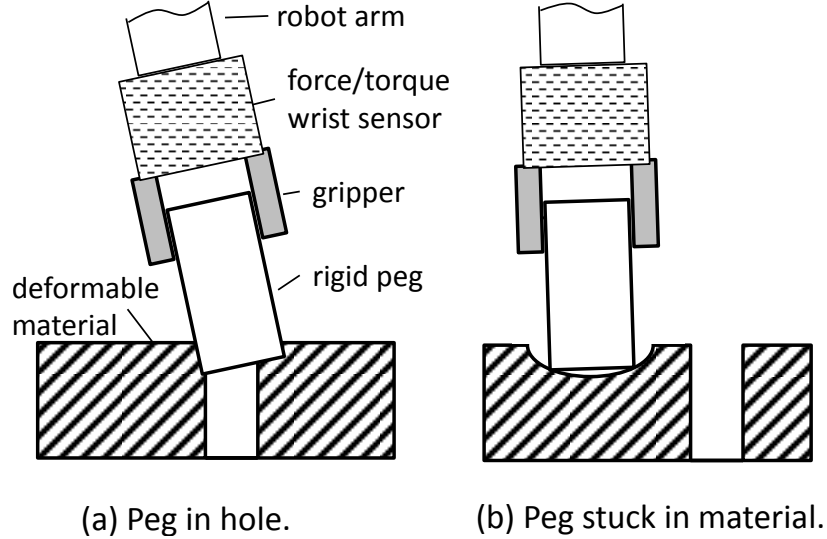


Figure 3.2: Assembly modes for rigid peg and deformable material.

Experimental Setup Details

As shown in Figure 3.1, we use a UR5 robot manufactured by Universal Robots, Polyscope version 3.5 with the motherboard in the control box having been upgraded to CB3.1. We use the force/torque sensor FT300 from Robotiq with a range of measurements of 300 N for forces and 30 Nm for torques. The noise levels (estimated standard deviations) for F_x, F_y are 1.2 N; F_z is 0.5 N; M_x, M_y are 0.02 Nm; and M_z is 0.03 Nm. The data output frequency is 100 Hz. We use a 85mm adaptive gripper from Robotiq. We remove the fingertips from the gripper, and fixed the peg directly to the gripper. However, the gripper connection allows bounded motions of the peg, which introduces some uncertainty. The hole is part of a work piece, which is made of ethylene-vinyl acetate. Its diameter is 20 mm while the peg's diameter is 25 mm. The cycle time for the UR5 is 8 ms, the robot does not provide access to its real-time control loop, also the force/torque sensor rate is 100 Hz. The robot neither has joint compliance nor does it offer joint torque control. Hence, it is well suited to benchmark our approach.

We vary the robot starting configuration for training local policies. The algorithm will be iteratively executed by deterministic reset to the same configuration after sampling trajectories. We collect five samples for each starting configuration in every iteration. We send control commands at 25 HZ; and each trajectory is 3.5 seconds long.

The robot state space consists of joint angles, joint velocities, the end-effector pose, which is represented by three Cartesian points and the velocities of these points. For an initial condition (robot configuration), we specify their target end-effector pose by three points in end-effector plane, the cost function is given by $r_\ell(d) = wd^2 + v\log(d^2 + \alpha)$, with

$\alpha = 10^{-5}$, $v = 0.01$ and $w = 1.0$. This shaped cost function performs better than an l_2 distance cost function because the second term encourages precise placement near the target position. In order not to damage the material, in all our experiments, we locked rotations in three Cartesian axes, thus the policy outputs an action $\mathbf{u}_t = [F_x, F_y, F_z]$. Denote the forces measured by F/T sensor as $\hat{\mathbf{u}}_t$, the commanded joint velocity $\mathbf{v}_t$ to the robot can be calculated as following:

$$K_p * (\mathbf{u}_t - \hat{\mathbf{u}}_t) = J_t * \mathbf{v}_t$$

where K_p is the feedback gain, and J_t represents robot Jacobian matrix.

Results

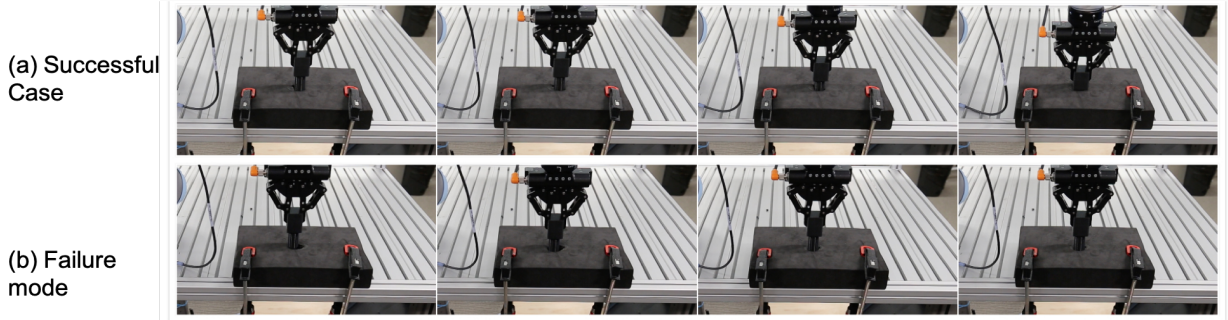


Figure 3.3: (a) Successful insertion. (b) Failure mode where the policy is unable to locate the hole.

We design our experiments by varying the starting positions of the peg, near to far; to test the if our method could learn policies effectively searching for the hole. Since the robot can only be position or velocity controlled, 1) it is not capable of running RL algorithm using torque control, which will provide compliant behaviors; 2) it is not capable of running RL algorithm directly commanding velocity or position either, it will stick to pre-calculated trajectories thus yielding unexpected large force or moments when facing contact. So, we consider only one baseline that includes several way-points, and use robot position controller to track them; since the hole is smaller than the peg, this baseline almost failed every single time because of safety stop from the robot itself. We report following results in Figure 3.4, three different starting positions are tried, the distance is regarded as difference of horizontal projection between center of peg and hole; we can see the policy training is able to reach a reliable convergence in roughly five iterations. Success rate comparison of our method and aforementioned baseline is listed on Table 3.1.

Figure 3.3(a)(b) shows some of the successes and failures. Several learned strategies can be observed. In the case that the initial position is slightly off the hole, the robot tends to apply large vertical forces to “slide” into the hole. While for the case that the initial position is further off the hole, the robot often first gets stuck in the material, then slightly lifts its

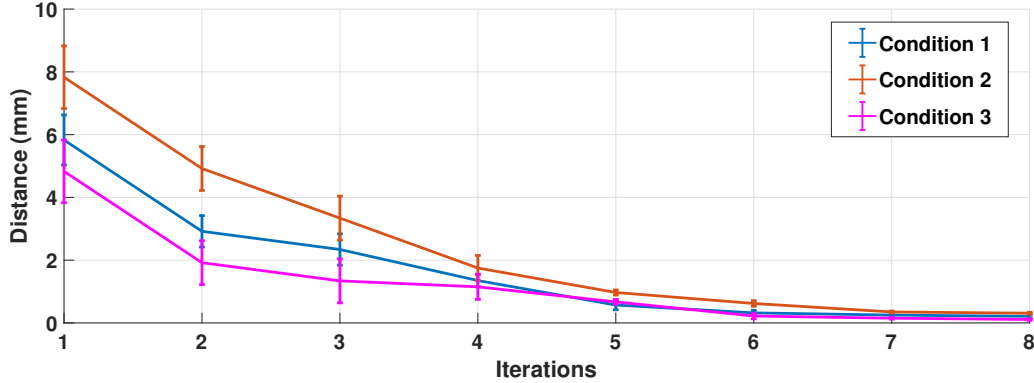


Figure 3.4: Learning curve in three experiments starting from different positions.

Table 3.1: Comparison of success rate, with different distance to the hole.

distance to hole	1 mm	6 mm	8 mm
our method	5/5	5/5	3/5
baseline	3/5	0/5	0/5

arm to reduce the exerted force from the material, and moves its arm quickly after the lifting to find the hole.

To summarize our contributions: We introduce a principled way for extending non-compliant robots to learning contact-rich manipulation skills. Currently, most policy search algorithms for contact-rich assemblies are implemented on inherently compliant robot arms such as the PR2, the iiwa or Rethink Robotics’ Sawyer [90]–[92]. These robot arms have either passive compliance through spring mechanism in motors or have the ability to measure and command joint torques. These properties enable safe physical interaction of the robot with its environment, and joint torque readings can be encoded as features in learning algorithms to describe contact situations. Unfortunately, this is only of limited use for industrial applications because industrial robots are in general not compliant and offer only velocity and position control, but no torque control. However, they can often be equipped with a wrist force-torque sensor. While this still does not provide the ability to command joint torques, it opens the possibility for admittance force-torque control in task space. We show how this can be exploited for GPS usage even if joint torques cannot be directly commanded, but only positions and velocities. To the best of our knowledge force-torque signals have not been incorporated in variants of the GPS algorithm yet.

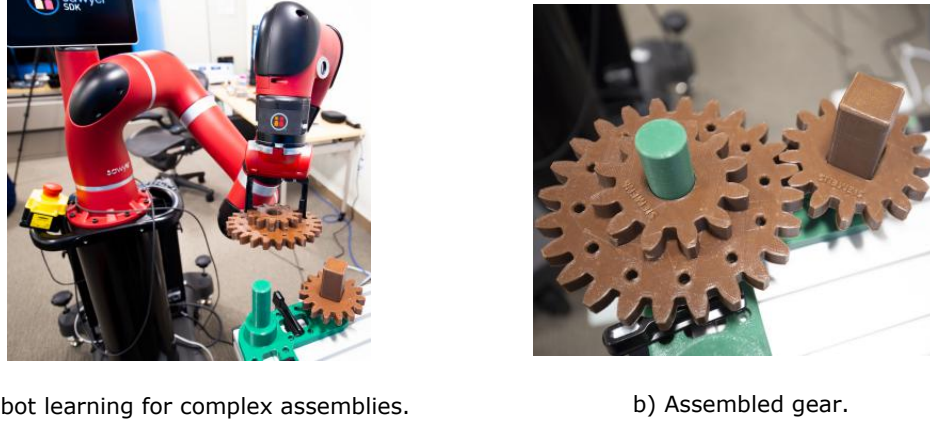


Figure 3.5: Learning control policies for assembly tasks.

Compliant Robot Arm with Rigid Objects

Problem statement

Consider the task of assembling the gear set shown in Fig. 3.5. The gear model was introduced by Siemens Corporation as a benchmark task for robotic assembly www.usa.siemens.com/robot-learning. The overall assembly task consists of four sequential steps, which are illustrated in Fig. 3.7: first the robot needs to insert a cylindrical peg into its matching hole; then the large brown gear should be inserted through the gear shaft; then the small brown gear with the squared hole should be assembled; lastly the gear wheels need to be matched by aligning the corresponding gear teeth. In general the tolerances are tight. For example, step two requires tolerances tighter than 0.1 mm, which is beyond most deployed industrial robots' accuracy today. Additionally, in step two, the peg can freely rotate at contact, the gear must be precisely oriented to match the squared peg; in step three, the small brown gear must be rotated by the large brown gear properly so that they can align with each other. This poses additional challenges: since none of these pegs or gears are fixed during assembly, this added uncertainty makes assembly even more difficult.

Interpretation as learning Pfaffian constraints

Our method could be regarded as generating Pfaffian constraints for each task. At contact, we can formulate holonomic and nonholonomic constraints enforced by the rigid environment as Pfaffian constraints:

$$A(q)\mathcal{V} = 0, \quad (3.6)$$

where $\mathcal{V}$ is the operational space twist in $SE(3)$ that $\mathcal{V} \in \mathbb{R}^6$, $A(q) \in \mathbb{R}^{k \times 6}$, k is the number of natural constraints. In motion control part, we can write down operational space dynamics

of the robot as:

$$\mathcal{F} = \Lambda(q)\dot{\mathcal{V}} + \eta(q, \mathcal{V}), \quad (3.7)$$

where $\Lambda(q) = J^{-T}(q)M(q)J^{-1}(q)$, $\eta(q, \mathcal{V}) = J^{-T}(q)c(q, J^{-1}\mathcal{V}) - \Lambda(q)J(\dot{q})J^{-1}(q)\mathcal{V}$. This is essentially the same motion dynamics without $\mathcal{F}_{tip}$ expressed in Eq. 3.1, but calculated in operational space.

By combining Eq. 3.6 and Eq. 3.7, we can express constrained dynamics for this hybrid motion/force controller as

$$\mathcal{F} = \Lambda(q)\dot{\mathcal{V}} + \eta(q, \mathcal{V}) + \underbrace{A^T(q)\lambda}_{\mathcal{F}_{tip}}, \quad (3.8)$$

where $\lambda \in \mathbb{R}^k$; and in this case, requested wrenches $\mathcal{F}_{tip}$ must lie in the column space of $A^T(q)$. In general, constraints $A(q)$ come from the environment the robot is interacting with. Abstracting this type of constraint for each individual manipulation task can be time-consuming and prone to errors. Instead, if we let $\mathcal{F}_{tip}$ be learned by RL through continuous interactions; we can roughly regard this process as iteratively improving $A(q)$ to generate increasingly accurate description of tasks, which is a key ingredient in high-precision settings.

Training neural network controller using MDGPS

We introduce a novel neural network architecture to process noisy force/torque readings from wrist sensors or other sources, which provide measurements in tool space. The neural network is shown in Fig. 3.6. Force/torque information is filtered with a low-pass filter (LFP), then concatenated in the second last layer of the neural network. Intuitively, we would like to provide most direct haptics information to the neural network as principle features; also avoiding the neural network to establish unreasonable correspondence between external force/torque readings and robot internal states. Also, the robot needs to move in free space before it is in contact; and the learned policy should be robot-configuration-dependent rather than force-dependent in free space. Since F/T readings are noisy, the learned policy dependent on robot state and coupled F/T generates random motions, if we directly treat F/T as an input to the first layer of the neural network.

For training this neural network, we adopt the mirror descent guided policy search (MDGPS) algorithm in [104]. Denote the neural network parameterized by θ as $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, the goal is to minimize the KL-divergence between linear-Gaussian controller learned via iLQG:

$$\min \mathbb{D}_{KL}(\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)||p(\mathbf{u}_t|\mathbf{x}_t)) \quad \forall \mathbf{x}_t, \mathbf{u}_t, t,$$

this can be implemented by supervised learning. These guided policy search methods also have some mechanism to enforce agreement between distributions of local policy and global policy by adding an additional KL-divergence cost [20], [103], [104].

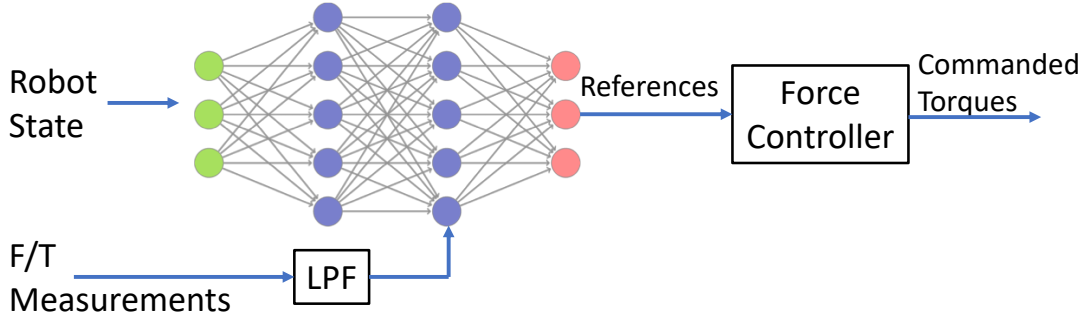


Figure 3.6: Neural Network Architecture

Experimental setup details

We evaluate our methods on four assembly tasks, which are shown in Fig. 3.2. We use a Rethink Robotics Sawyer robot. Sawyer offers an interface to query its wrist force/torque measurement, the noise levels (estimated standard deviations) for F_x, F_y are 2.0 N; F_z is 0.5 N; M_x, M_y are 0.5 Nm; and M_z is 0.1 Nm. Sawyer is commanded via ROS at 20 Hz. During training, we take four roll-outs per iteration. Typically, it takes three iterations to achieve successful behaviors, five iterations for convergence. We define a plane by three points in end-effector space, the cost function is a weighted mixture of the ℓ_1 and ℓ_2 norms of the differences between the current plane and the target plane as specified by the three aforementioned points.

Assembly performance results

We compare our method with the following baselines:

- **Kinematics Only:** For task 1 and task 2, we only specify target poses; for the more difficult task 3 and task 4, we also introduce several way-points. Note that for task 1 and 2, we should get the same result every single time since robot kinematic controller is deterministic as well as these tasks. But for task 3 and 2, the peg and gear can move freely, so it is hard to specify the desired trajectory.
- **iLQG with torque control:** This is the main baseline for comparison. The control actions from iLQG are directly the seven joint torques. For comparison, we use the same cost function as in our method, i.e., sparsely-defined target end-effector pose, no intermediate way-points are introduced.
- **iLQG with torque control, augmented state space:** We augment the state space with the F/T vector such that $\tilde{\mathbf{x}}_t = [\mathbf{x}_t, \mathbf{f}_t]$, where $\mathbf{f}_t$ are F/T measurements. We apply

direct torque control. The purpose of this is to verify if other formulation other than what we proposed could also actively use this additional information.

- **Our Method:** We refer to the operational space controller with iLQG.
- **Our Method with augmented state space:** Additional to operational space controller, we augment the state space to $\tilde{\mathbf{x}}_t = [\mathbf{x}_t, f_t]$. This experiment is for verifying if our method can be further improved.

A success is considered if an object is being assembled to a desired pose with defined tolerance. We report success rates for each individual task separately, because we train an individual policy for each task. However, it would be straightforward to report overall success rate by multiplying individual success rates together since policies are trained independently. We execute learned policies after training to calculate the success rates. Table 3.2 presents aforementioned success rates for four different tasks.

Table 3.2: Comparison of success rates for different tasks. Baseline 1 refers to kinematics only; baseline 2 refers to iLQG with direct torque control; baseline 3 refers to iLQG with direct torque control, augmented state space, our method w/ augmented refers augmented state space in our method.

	Task 1	Task 2	Task 3	Task 4
baseline 1	0/5	0/5	0/5	0/5
baseline 2	1/5	0/5	0/5	0/5
baseline 3	0/5	0/5	0/5	0/5
our method	5/5	5/5	2/5	4/5
our method w/ augmented	5/5	5/5	3/5	3/5

We interpret these results several fold: (1) kinematics baseline fails consecutively, this confirms the required accuracy and complexity for the gear set; (2) an iLQG with torque control, but without extensive cost shaping fails; the single success we observed is due to Gaussian noise in the controller, which generated some lucky motion to insert, and it is on the easiest task. (3) we did not find reliable improvement by augmenting state space with F/T information. Since F/T signals are not Markovian, fitting a time-correlated dynamics model to them does not produce meaningful information.

We made several interesting observations during the experiments. During task one, the robot moves quickly in free-space to reach in-contact status, then it reduces its speed to slowly probe around, trying to "feel" the surface; once it has a level of confidence of the hole's position, it becomes aggressive towards the goal it predicted, resulting in quick motions followed by a large downward force to complete insertion. The most interesting experiment



a) Task 1: Round peg in round hole.



b) Task 2: Gear wheel on shaft.



c) Task 3: Squared hole on squared shaft.



d) Task 4: Alignment of gear wheel teeth.

Figure 3.7: Snapshots of experimental runs for the four studied assembly tasks.

is task 3, where the added uncertainty comes from a rotating peg. The robot first brings the small gear in contact with the peg, while applying a downward force so that small gear would not fall into free-space again; but this amount of downward force also allows room for applying additional rotating torque to the peg and gear aligning them with each other roughly; then the downward force increases to try insertion, if not successful, downward forces decrease but small horizontal force are also observed to fine-tune poses, this procedure iterates until the peg is fully inserted. This kind of behavior roughly aligns with humans' heuristics when facing such tasks. These behaviors can be found in the supplemental video¹

Fig. 3.8 shows computed actions (only desired forces in x-directions and y-directions are plotted) for task 2 during one successful insertion. It is interesting to observe how the variance computed by the policy changes over time. Initially, there is a certain level of variance for

¹<https://www.youtube.com/watch?v=kqA1t1PaT8E>

exploration to search for the target position; once the policy is confident about the goal, the variance reduces dramatically, the robot aggressively moves the object towards the goal; finally during the insertion phase, a certain level of noise is again injected for fine-tuning the gear’s pose to overcome friction. This force-based insertion pattern is automatically discovered through interactions by the algorithm, and matches a human’s intuition on such tasks well. Fig. 3.9 presents F/T measurements during a successful insertion. We can observe peaks both in force and torque data, indicating some critical phases, e.g., contact. This motivates explicit use of F/T measurements, because of its informative nature.

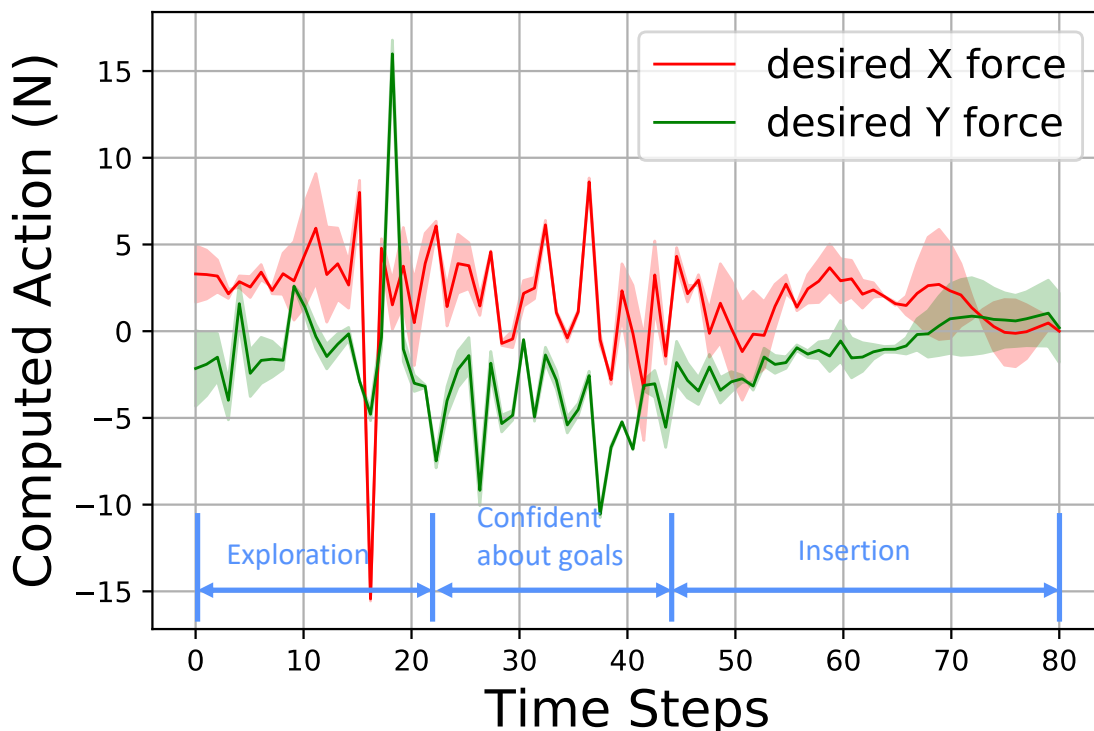


Figure 3.8: Action computed by learned policy during one successful insertion. Solid line shows computed action mean, and error bar for computed variance.

Generalization results

We only train the neural network controller based on a single instance of the local linear-Gaussian controller. Hence, our purpose is not to see if the trained neural network controller can interpolate between multiple local controllers as in original guided policy search methods [20]. Instead, we are interested in examining if the proposed architecture can effectively use F/T information and adapt to environment variations. We compare our proposed neural network against two baselines: one is to directly input F/T information into the first layer

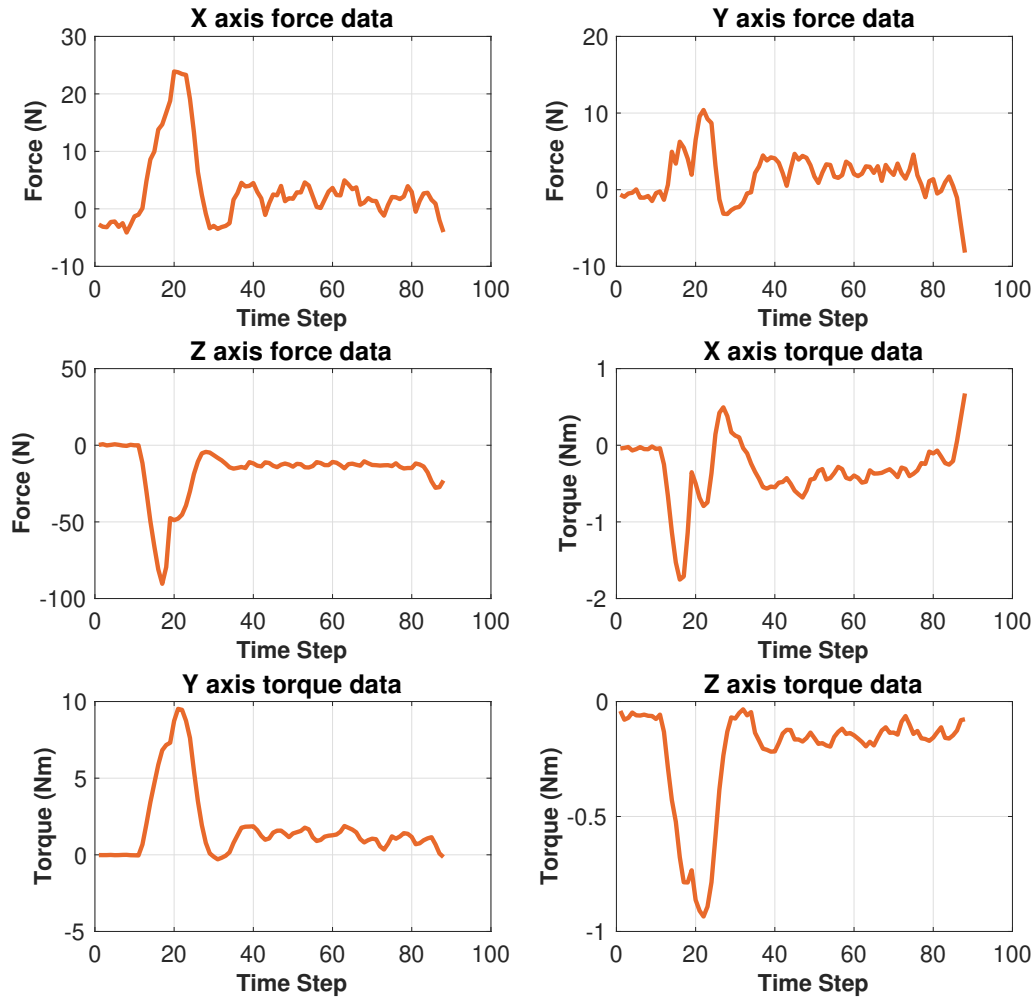


Figure 3.9: Six degree of freedom force torque measurements from a successful Task 2 insertion.

of the neural network; second is the iLQG controller that the neural network controller is trained on. We again only consider task 2 in these generalization experiments. We design our experiments as following: we train these three policies towards with the goal fixed, i.e., base does not move; after all policies are trained, we move base to slightly different positions but policies will be kept unchanged; then we count success rates for these variations of different base positions. The intuition is that even base is moved, the proposed neural network controller should still be able to find the hole if F/T information is actively used, since variations in base positions bear same force pattern in terms of peg-hole insertions. The comparison with iLQG baseline is due to the fact that slight variations in the goal position could also result in low cost in LQR. We want to distinguish between this and active F/T information.

We test these three methods in three different settings where the base is moved 1cm, 2cm, 5cm respectively. Success rates are reported in Table 3.3

Table 3.3: Comparison of success rate in generalization capability

	1cm	2cm	5cm
iLQG	8/10	5/10	6/10
our method	9/10	8/10	6/10
F/T input to first layer	0/10	0/10	0/10

For neural networks that input F/T data to their first layer, the training was often aborted due to large KL-divergence between iLQG and neural network; for few scenarios we could successfully train a policy, it often generates random, undesirable motions in free-space before insertion; it behaved poorly on three experiments. This validates our hypothesis for not establishing correspondence between external F/T readings and robot internal states.

For the comparison of the proposed neural network controller against iLQG controller, we found that neural network controller produced slightly better results; but at the mean time, the variance of the result was also high. The neural network controller did not outperform iLQG consistently. Getting a more accurate six axes F/T controller might mitigate this issue.

3.5 Discussion

In this chapter we combine RL with an operational space force controller to solve the problem of high-precision robotic assembly. We specifically exploited one of the model-based RL algorithm, iLQG, compared with several ablations. We evaluated our method with two use-cases: deformable object manipulation with a non-compliant robot, high-precision assembly with a compliant robot. Results show our method can not only perform well in high-precision

settings, but offer a principled way for industrial non-compliant robot learning contact-rich assembly skills.

One future direction is to add raw vision and tactile inputs to the architecture in Figure 3.6, thus becoming an end-to-end multi-modal neural network. It would be interesting to see if such learned policy can succeed from arbitrary starting positions in free space, and tactile sensing could further improve policy performance. Another interesting direction is to explicitly model contact, and encode such information as priors for a more structured Pfaffian constraint matrix, this would further reduce sample complexity and serve as a general primitive for policy transfer.

Chapter 4

Learning scalable robotic grasping with action image representation

4.1 Introduction

Grasping is a fundamental task that is a prerequisite for robots to manipulate the environment and interact with the world. Common household tasks, such as tidying a room, setting a table, or preparing a meal all require grasping as a fundamental capability. Due to its importance for manipulation, nearly every facet of grasping has been studied in the field of robotics. In this section, we build on previous work on inferring grasp poses through models and propose a new representation, *Action Image*, to predict successful grasps from perceptual observations.

A common practice for data-driven grasping is a two-stage method of first using perception to detect objects and/or estimate their poses in a scene, and then synthesizing a feasible grasp using a planner or a learned policy in the robot’s task-space. We refer to these types of methods as perception-dependent grasp synthesis approaches, see [115] for an extensive literature review. More recently, there has been a shift towards end-to-end grasping, inferring grasps directly from sensory information such as images and/or depth.

End-to-end approaches compare favorably to two-stage perception-dependent methods, since they are jointly optimized and the overall performance is not bound by the performance of each individual stage. This improvement is in large part due to the difficulty in recovering from errors. Multistage methods benefit from abstractions that allow solving a collection of easier sub-problems, but the downside is each stage increases the possibility of irrecoverable failure. As an example, if the perception stage poorly estimates the pose of an object, then all subsequent grasp synthesis will accumulate this error. In contrast, end-to-end grasping policies learn implicit object-centric representations where object-detection and pose estimation become a fused part of the network and multiplicative errors are avoided. The cost of a combined representation is typically the need for massive datasets to produce quality results and the increased complexity of optimization during training. While these large



Figure 4.1: Real-world grasp success evaluation of policies trained only in simulation using a cluster of 8 mobile manipulators.

datasets are possible to obtain in simulation it is often unfeasible to obtain the same scale of data in the real-world. A further concern is that most end-to-end methods do not generalize well to data distribution shifts, which is unfortunately very common when transferring from synthetic to real-world data.

End-to-end approaches for grasping are roughly formulated to solve a one-to-many problem. This is because grasping is naturally multimodal; given a single observation there exist many grasp poses that will lead to a successful grasp. There are two predominant end-to-end approaches for tackling this: 1) *Stochastic actor networks* that produce a multimodal distribution between a sensory observation and possible grasps. 2) *Critic networks* that produce a map between sensory observation with grasp proposal and probability of success. In this section we choose to focus on a *critic network* approach, building on the recently demonstrated effectiveness of its use in grasping by [18].

We hypothesize that by representing grasps (actions) with a representation that is compatible with the intrinsic assumptions of the network predictions of grasp success, *it is possible to learn grasps for novel objects in the real world without the necessity of any real world training data*. We argue that a meaningful representation for grasping must be: 1) Applicable to common sensory modalities accessible to robots; namely color images, depth

images, and their combination, 2) Domain invariant, such that it can seamlessly work in either simulated or real environments; in other words, the representation of grasp quality should be independent of whether the sensory input is synthetic or real, 3) Generalizable to new objects, and 4) Invariant to translations in sensor-space; grasp experience should be efficiently transferred between locations of the task-space.

We test our hypothesis by proposing a novel representation, called *Action Image*, that contains the above characteristics through embedding a grasp proposal as rendered features and included as channels of an image. This representation allows us to build a deep convolutional grasp policy that innately is invariant to translation in image space, assuming both the object and grasp is transformed in the same manner; information about the grasp hypothesis naturally fits the implicit structural assumption imposed by convolutions.

Furthermore, this section provides experimental validation of our hypothesis that *Action Image* is a meaningful representation that improves grasp success prediction and facilitates domain transfer. We first demonstrate performance gains and compare efficiency of the method across increasing amounts of simulated data. Then, we provide strong evidence of domain invariance with transfer to real-world scenarios, achieving 90% grasp success. Finally, we show that *Action Image* networks trained with synthetic data generalize to real world objects with only slightly degraded performance

In summary, the contributions of this section are as follows:

- A new representation, called *Action Image*, that makes grasp prediction invariant to translation in image space.
- Demonstrating an *Action Image* representation is easily extensible to different input modalities, e.g. RGB, D, and RGB-D images.
- Experimentally verifying the data efficiency and the domain invariant performance of our method.

4.2 Related work

Grasping is a building block for manipulation and has long been explored by the robotics community [115], [116]. Early approaches to grasping used models of the environment coupled with grasp planning based on force closure metrics [117] or grasping-specific simulations [118] to compute optimal grasps. In order to execute these predicted grasps in the real world, the 3D object models have to be identified, located and their pose estimated [119]–[122].

Recently, data-driven methods have become popular [115], due to their flexibility in specifying sensory input and ease of incorporating data from experience through real robot grasp executions [123]. Some of the first work to apply deep learning to the problem of grasp detection used manually annotated datasets of images and corresponding grasps. These approaches applied two-stage learning pipelines similar to those used in object detection and classification in computer vision [16], [124]. Since manual annotation can be time-consuming,

simulation approaches have been exploited to generate real-world like sensory data [125]–[128]. For real world data collection researchers have since moved towards self-supervised data collection, wherein a robot labels its own grasps based on heuristics [17]. Pushing self-supervised learning to millions of trials has allowed learning of hand-eye coordination for grasping [18], as well as applying a variant of Q-learning [129] to grasp unknown objects in a cluttered environment using a single color camera. These aforementioned data-driven approaches are related to our work, since they both train a critic network that accepts images and actions as inputs and output a metric of grasp success. In our work we also use the cross-entropy method (CEM) [130] to find optimal actions from a critic network. Our work is differentiated from previous efforts in a few ways: (1) we represent actions directly in image space, which simplifies the network and allows learning from just simulated data, and (2) our network is easily extensible to use depth data.

With the rise of data-driven techniques considerable attention has been given to leverage simulation to reduce the need for real-world robot data, tedious and time-consuming to collect. Researchers have used domain randomization [131]–[133] or trained pixel-to-pixel networks, converting either from simulated images to real images [134], or vice versa [135]. Compared to [135], our network is trained supervised, significantly smaller and faster to train, as it does not rely on another network to transfer images from one domain to another.

Work in this area has also demonstrated that using depth data [126] (as opposed to RGB images alone) results in better performance when transferring policies trained in simulation directly to real robots. This observation informed our experimental design to test improvements in performance using depth data over RGB data alone. Compared to [126], our representation is applicable to both RGB and depth, and allows easier extension to higher action dimensions such as 6D grasping or the gripper angle.

Fully convolutional networks have also been leveraged to perform grasp detection in image space by producing several output heat maps indicating grasp success discretized over top-down gripper orientation [136]. While this approach shares some of the advantages of our grasp representation, such as translation-invariance, our approach naturally extends to higher degrees of freedom without the need to further discretize the output heat map space.

4.3 Method

The grasping problem was formulated as an iterative optimization procedure using a supervised critic network approach, similar to [18]. A learned critic network $\mathcal{G}$ was used to evaluate the likelihood of grasp success for the inputs of sensor observations and action representations for table-top grasps. The Cross Entropy Method (CEM) was used in conjunction to refine the best grasp candidate by iteratively sampling in an action space [130].

The grasp prediction process is laid out in Figure. 4.2. The initial sample distribution at $t = 0$ is a design parameter and without lack of generality this implementation used a uniform distribution over a predefined robot workspace for simplicity. All following samples were fed through the *Action Image* preprocessor (Section 4.3) that generated the network

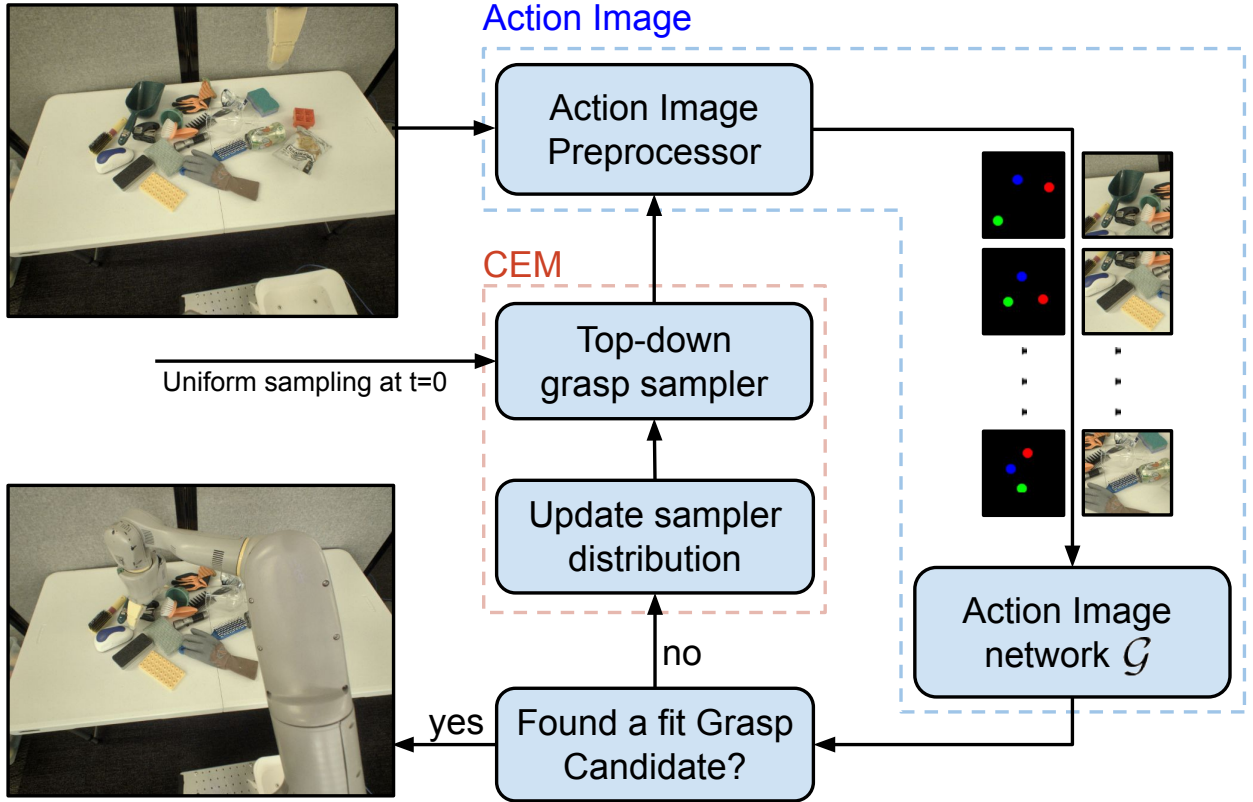


Figure 4.2: The procedure for performing end-to-end grasping using an *Action Image* representation. Our mobile manipulator first approaches the workspace and then grasp candidates are uniformly sampled from the workspace. Next, a preprocessor generates pairs of sensed-image and action-image data corresponding to crops centered around the grasp proposals. Then, an *Action Image* network evaluates the candidate pairs and if a grasp proposal exceeds the desired probability of success the grasp will be executed by the robot. Otherwise, the grasp candidate distribution is updated with the top 10% of elite proposals and the process is repeated.

input representation based on the grasp proposals. Next, the critic network $\mathcal{G}$ evaluated the probability of success $\mathcal{P}$ for all grasp candidates. When the grasp with the highest probability exceeded a threshold $\mathcal{P} \geq \mathcal{P}_{thr}$ that grasp was selected and executed by the robot. Otherwise, the sampling distribution was updated using the top $M\%$ of elite candidates and the process was repeated until a grasp was found or a maximum number of iterations was reached. During evaluation, if no grasp reached the success criteria the best available candidate was selected.

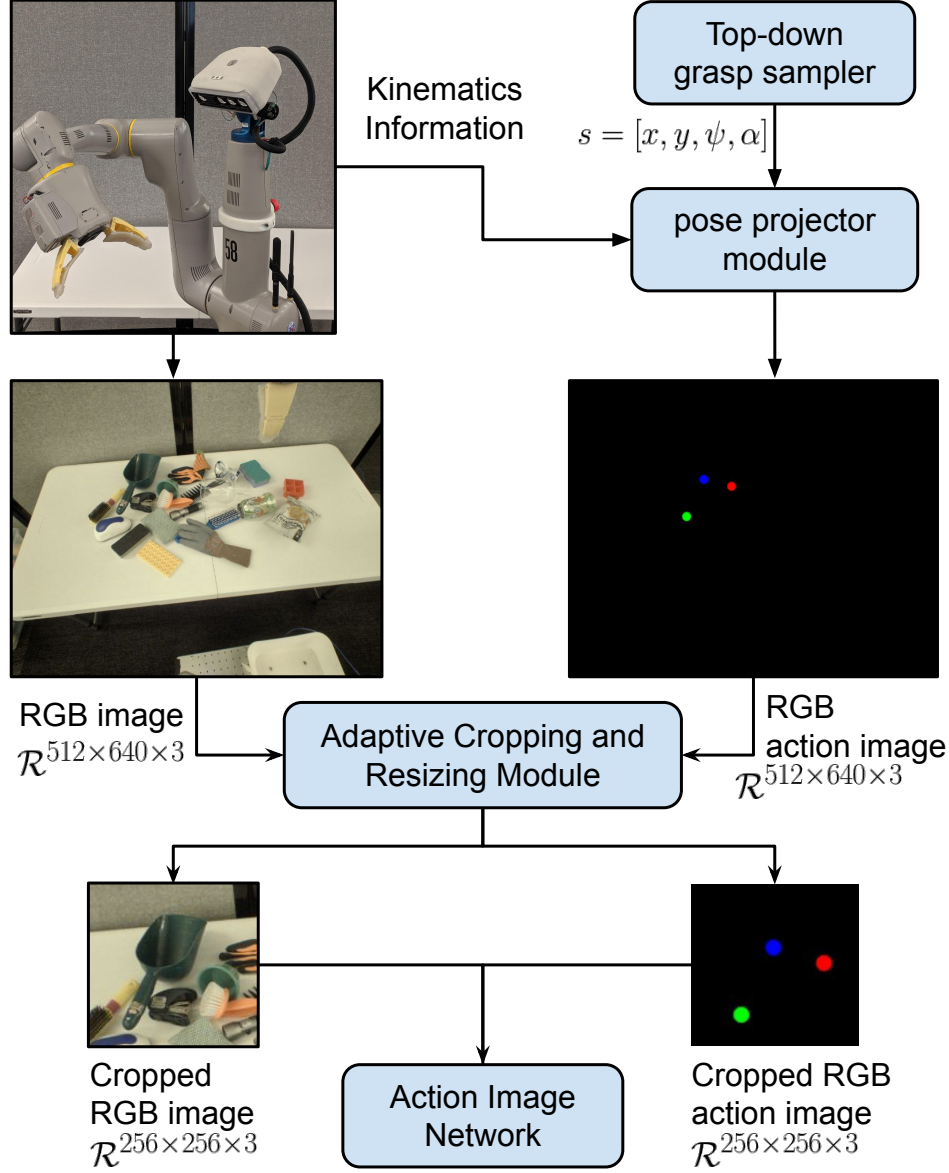


Figure 4.3: The description of the *Action Image* preprocessor. Given a sample pose, the 3D position of the gripper features is calculated and projected into camera-space producing the *Action Image* representation. The number of these features determines the expressiveness of the representation and at least 3 points are required to estimate the pose. The values of the projected points can carry additional information, such as depth in camera frame. Next, the resulting *Action Image* and sensor-image data are adaptively cropped to a $128 \times 128 \times 3$ square centered at the proposed grasp candidate and then are resized to $256 \times 256 \times 3$.

Action Sampler

In general, grasp candidates are generated by sampling from an action-space that describes the task. For this study we defined a specific task-space appropriate for top-down grasping from a table. A grasp candidate was represented by a four dimensional vector $s = [x, y, \psi, \alpha]$, where x and y corresponded to the planar position of the object on the table, ψ was the gripper yaw angle, and α was percentage of gripper closure where 0 was fully open and 1 fully closed. Bounds on the action-space were set by assuming information about the workspace, such as table height, width, and length, and denoted as $\mathcal{W}$.

Given a sampling distribution $\mathcal{D}$, we generated a batch of N top-down grasp candidates. Samples that were outside the workspace $\mathcal{W}$ or kinematically unfeasible were filtered out leaving a subset of allowed samples, $\hat{N}$. This process was repeated until a desired batch size N^* was obtained.

Action Image Representation

In this section we describe the core contribution of this section; *Action Image* representation combines the power of convolutional networks to capture the spatial relationship of a grasp hypothesis with the local appearance and shape of objects. This process is outlined in Fig. 4.3. In general, any salient feature set that expresses relevant grasping data and is expressible in image-space, such as rendering of gripper wireframes or CAD models or simple points could be used by this approach. For this work we chose to encode information about the gripper using virtual points.

For a given grasp candidate, we can use the end-effector kinematics to determine the relative transform from end-effector $\mathcal{E}$ to feature point ξ_k , $T_{\xi_k}^{\mathcal{E}}$. Further, since the grasp candidate is expressed in a workspace $\mathcal{W}$ relative to the robot, we know the transform from the robot frame R to the end-effector pose given in $\mathcal{W}$, $T_{\mathcal{E}}^R(s; \mathcal{W})$. Assuming the camera is calibrated to the robot frame, we can also use the robot kinematics to find the camera frame c relative to the robot frame as T_R^c . Combining these, we can find the location of each feature point in the camera frame as:

$$T_{\xi_k}^c(s; \mathcal{W}) = T_R^c T_{\mathcal{E}}^R(s; \mathcal{W}) T_{\xi_k}^{\mathcal{E}} \quad (4.1)$$

These feature points can in turn be projected onto the camera image plane through the camera intrinsic projection matrix. As a result, the *Action Image* feature points can be synthesized for any grasp candidate without the need to execute the grasp.

In this study, we described our specific pinch gripper using 3 feature points as can be seen in Fig. 4.4. Only 3 points were selected to provide the simplest graphical representation sufficient to infer the top-down grasp pose and opening of the gripper.

In addition to the spatial features of an *Action Image*, the value of the rendered feature pixels can embed additional information relevant to the sensory modality. In our work, the virtual point features were rendered as a simple binary mask channel for the models with RGB information alone. In this case depth information was implicit, encoded only through

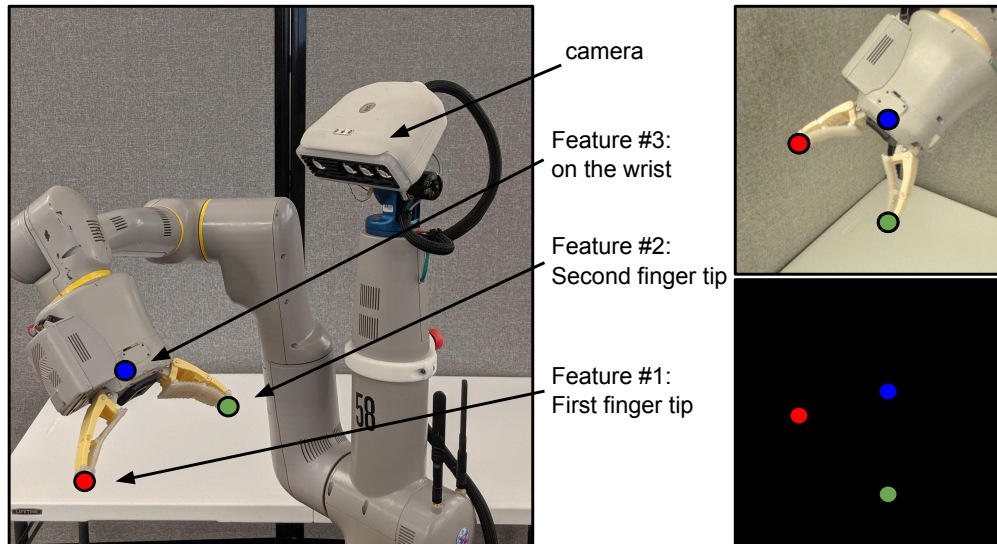


Figure 4.4: **Left:** We represent a gripper by 3 feature points: two features are placed on the gripper finger tips and the third one is placed on the point where the fingers are attached to the wrist. **Right:** These feature points can be easily projected into camera-space through known kinematic transformations available from the robot. The generated *Action Image* is shown in the bottom-right image.

the projection of the 3D points. In contrast, the D and RGB-D models directly set depth values for the feature patches and explicitly encoded this information. This showcases another advantage of the *Action Image* representation, since it allows the network to exploit the intrinsic structural assumptions of convolutional neural networks and supports encoding prior knowledge in the value of the pixels.

The final piece of the representation used in this study was to exploit the locality of the image-based sensory information (RGB, D, RGB-D) by cropping image-data and *Action Image* tightly around the *gripper features*. This preprocessing step resulted in partial network re-computation and could be further optimized by region-of-interest (ROI) feature pooling [15]. In practice, the application of CEM-based grasp optimization to our naive input cropping approach was sufficiently fast, resulting in inference times on the order of a few milliseconds.

Critic Network Architectures

The architecture employed in this study was a composable critic network that allows the assembly of feature towers to rapidly incorporate different sensory modality inputs. Fig. 4.5) illustrates the architecture for the three *Action Image* networks proposed in this work and the architecture for the baseline following the design proposed in [18].

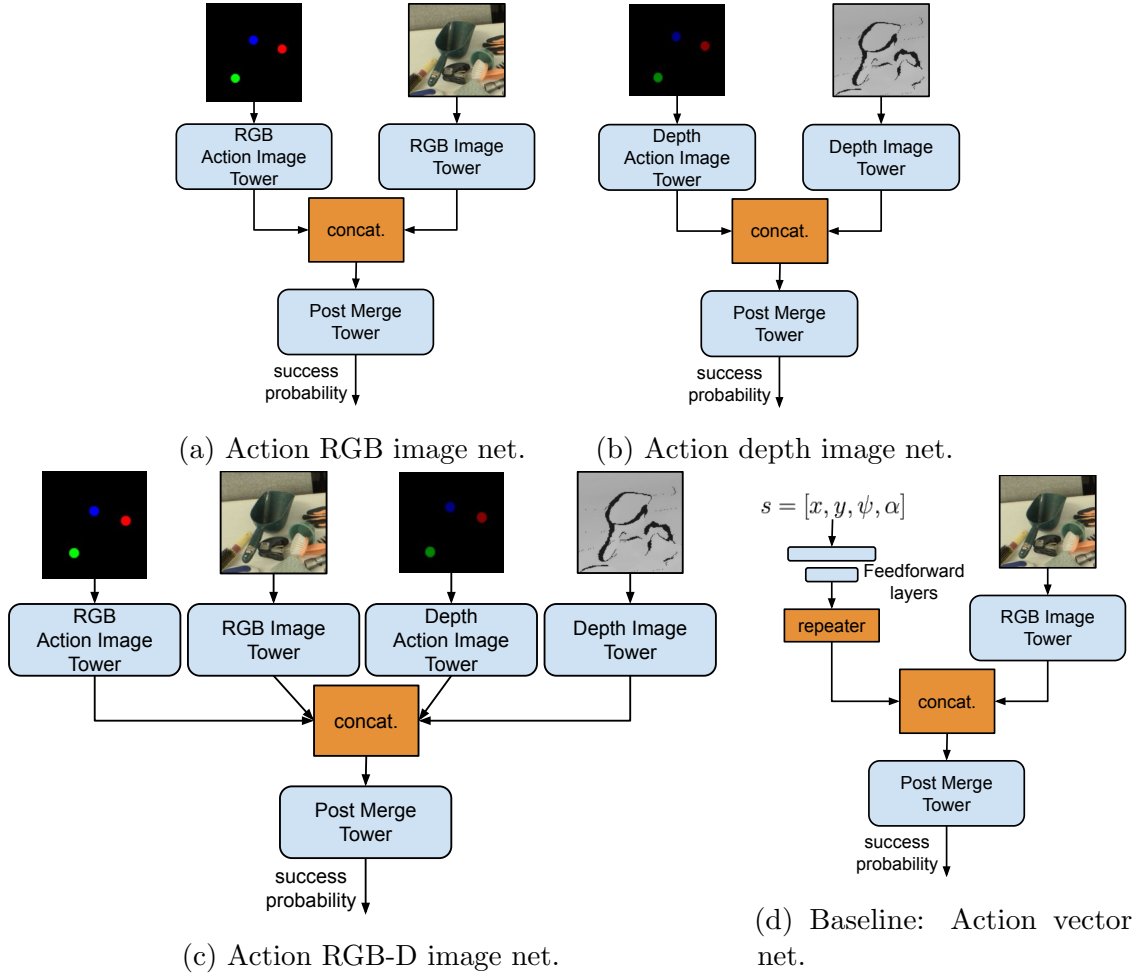


Figure 4.5: Representations for the *Action Image* networks used in this section were (a) action RGB (b) action D and (c) action RGB-D. All networks were composed solely of convolutional layers. These networks were compared to a baseline (d) action vector network where the grasp candidates were fed to the network as a 4D vector of floats $[x, y, \psi, \alpha]$.

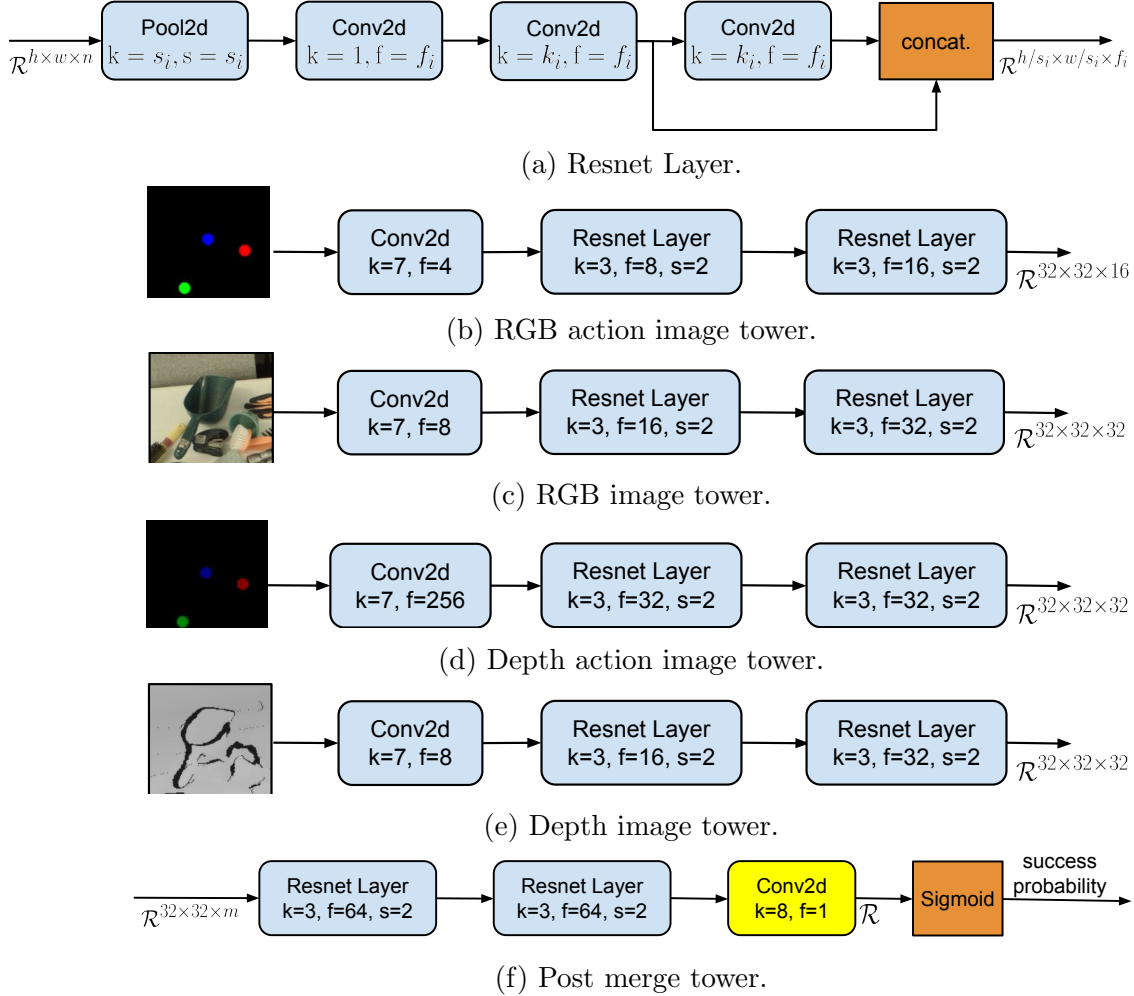


Figure 4.6: Details that illustrate the sub-components of the different action image networks.

Each *Action Image* network was composed of a collection of feature towers from the set of: 1) image tower, 2) depth tower, 3) RGB action image tower, 4) depth action image tower, and 5) post-merge tower, Figure 4.3 shows the architectural details. These towers used convolution or ResNet layers as their building blocks (see Fig. 4.6a). The output from the first 4 feature towers was a $32 \times 32 \times 32$ tensor. The post-merge tower was the common tower across all networks. This tower took a $32 \times 32 \times 32$ tensor input and produced a grasp success probability.

The baseline model, called action-vector, differed from the *Action Image* networks, since it represented a grasp as a vector of floats (see Fig. 4.5d). In this network, action-vector feature tower consisted of the input, a grasp vector, passing through a series of fully connected layers (size 256 and 32) and then concatenating with the output from the image tower by repeating

Algorithm 3 Scripted data collection policy in sim.

Require: $\epsilon, \sigma_x, \sigma_y, \sigma_\psi, \sigma_\alpha$ and workspace $\mathcal{W}$

- 1: Create a new scene with $n \sim \text{Uniform}(8, 12)$ objects.
- 2: Store a snapshot of the scene $\mathcal{I}$.
- 3: Draw $p \sim [0, 1]$
- 4: **if** $p > \epsilon$ **then:**
- 5: Get objects position: $[x_i, y_i, \psi_i], i \in [0, n]$.
- 6: For each object generate grasp pose by:
 $\beta_i = [\mathcal{N}(x_i, \sigma_x), \mathcal{N}(y_i, \sigma_y), \mathcal{N}(\psi_i, \sigma_\psi)]$.
- 7: Randomly select a kinematically feasible grasp pose
 $\beta \in \{\beta_0, \dots, \beta_n\}$.
- 8: **else**
- 9: Draw a feasible top-down pose from the workspace.
- 10: Generate gripper opening angle $\alpha = \mathcal{N}(0.5, \sigma_\alpha)$,
- 11: Set the grasp pose $s \leftarrow [\beta, \alpha]$
- 12: Execute the grasp pose s .
- 13: Determine the grasp success l_s by lifting the object.
- 14: **Return** $\mathcal{I}, s$ and $l_s = 0$

the same feature vector across the 32x32 spatial representation.

4.4 Experiments

Model Training

Supervised training of the critic networks was accomplished using 3 datasets generated *solely in simulation*. No real world data was used during training for any of the models. We used Bullet to simulate the task [137]. Fig. 4.9 shows the simulation setup for the data collection in sim. Fig. 4.7 illustrates our training object dataset consisted of 48 objects distributed between four object classes: bottles (8), cans (6), cups (19), and mugs (15). For each run, the robot moved to a random base location within its base workspace illustrated in Fig. 4.9. From the set of objects, a random number was drawn from the range of [8, 12] and distributed randomly within the workspace. Objects were initialized slightly above the table and allowed to fall, introducing additional randomness for their position and orientation. This resulted in some objects being placed outside the workspace or on the floor. Domain randomization was applied to the table, floor and objects by shuffling the texture and color during each run.

Curriculum training was employed for each model to generate a rich collection of 7.2 million samples containing both grasp success and failures (Table 4.1). The initial bootstrap dataset (off-policy) contained 2.7 million training examples collected in simulation using a scripted grasping policy using Algorithm 3. We trained all of the networks by using an ADAM optimizer on the cross-entropy loss between the estimated outcome from $\mathcal{G}$ and the true label $\mathcal{O}$. We used batch size 32 and balanced each batch with equal number of successful and failed grasp data.

Next, each trained network was used to generate on-policy datasets according to Algorithm 4. Please refer to Fig. 4.2 and Section 2.3 for further details on how a grasp candidate was

selected. By pooling the 850 thousand grasps from each of the 4 models the on-policy #1 dataset of 3.4 million grasps is created. Finally, each network was trained on both the bootstrap and on-policy #1 datasets and in a similar fashion the on-policy #2 dataset of 1.1 million grasps was generated.



Figure 4.7: Set of 48 simulated objects used in the training of all models.



Figure 4.8: In total 172 objects are used in the real robot evaluation, including 80 familiar (left) and 92 novel (right) objects.

Experiments and Results

We developed a set of experiments to explore whether an *Action Image* representation can improve performance over a direct-abstract representation and tested its performance in domain transfer from simulated to real-world scenes. Simulation evaluation was performed for the

Algorithm 4 On-policy data collection and eval in sim.

Require: Policy $\mathcal{G}$, initial sampling distribution $\mathcal{D}$, and workspace $\mathcal{W}$

- 1: Create a new scene with $n \sim \text{Uniform}(8, 12)$ objects.
 - 2: Store a snapshot of the scene $\mathcal{I}$.
 - 3: Get the grasp pose $s \leftarrow \mathcal{G}(\mathcal{I}; \mathcal{D}, \mathcal{W})$.
 - 4: Execute the grasp pose s .
 - 5: Determine the grasp success l_s by lifting the object.
 - 6: **Return** $\mathcal{I}$, s and l_s
-

sole purpose of comparing the best possible performance across the different models. Testing on-robot was used to evaluate both domain transfer to real-world data and generalization to new objects. Furthermore, we compared 3 *Action Image* representations to evaluate the efficacy of the method to fuse different image information. We also present these results against purely random and action-vector baselines to demonstrate the relative performance of the method.

Simulation Evaluation

The simulation evaluation was performed according to Algorithm 4. No hold-out objects and no real-world data were used during simulation evaluation. The same set of objects used during training were used, but due to randomization of the color, texture, and pose, each scene was unique and different from those at training time.

Table 4.2 provides a summary of the evaluation in simulation. All the three *Action Image* models reached +75% success rate when trained only from scripted policy data. Using on-policy data improved all models to +85% success rate and action-depth network had the best performance at 89%. As hypothesized, the baseline model was less data-efficient and only achieved 20% success rate using the scripted policy data. The baseline model performance improved significantly (69%) when trained on all data, i.e. Dataset C, but did not reach the success rates of Action Image models that had been trained on much less data.

	scripted policy	on-policy #1	on-policy #2	total # grasps
Dataset A	2.7m	-	-	2.7m
Dataset B	2.7m	3.4m	-	6.1m
Dataset C	2.7m	3.4m	1.1m	7.2m

Table 4.1: A list of simulation datasets and their size. On-policy data #1 and #2 were collected from policies trained on Dataset A and B, respectively.

On-robot Real-world Evaluation

The real-world experiments were performed using a cluster of 8 mobile manipulators, see Fig. 4.1. On-robot evaluation was similar to the simulated data collection setup. First, the robot

	Dataset A	Dataset B	Dataset C
Action RGB image	79.09%	85.08%	85.47%
Action depth image	75.04%	87.42%	89.27%
Action RGB-D image	76.38%	85.32%	87.85%
Action Vector (baseline)	20.45%	60.53%	69.06%

Table 4.2: Evaluation results in simulation. Success rate for each model is based on +5K grasp evaluations.

moved to a random pose in front of the table and then executed the grasping policy (see Fig. 4.9). To evaluate the generalization capabilities of the different policies, we considered two object datasets (see Fig. 4.8): 1) A *familiar* set that included 80 objects from the classes of bottles, cans, cups, and mugs. These objects were similar, but not identical to the simulation dataset. 2) A *novel* set that included 92 random house-hold objects ranging from staplers, gloves, and sponges to combs, flashlights, and crushed cans. Both familiar and novel datasets had different visual properties from the simulated data, since the simulation did not render transparency or high reflectance. The novel dataset differed significantly from the simulation dataset in all aspects. It contained deformable objects, gloves; thin objects, combs; and complex concave objects, scoops.

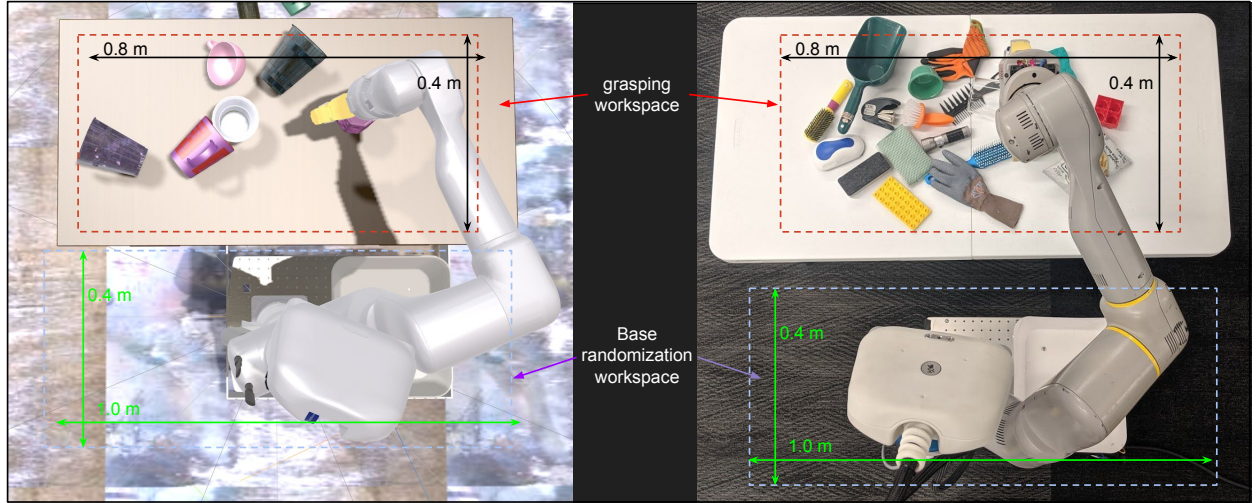


Figure 4.9: Workspace and random base footprints for real-world (right) and simulated (left) scenarios.

We considered three different scenario sizes each with an increasing number of objects, small (5), medium (10) and large (20), see Figure 4.10. For each evaluation run, we initialized a scene with one of a small, medium or large number of objects and drawn from either

	familiar			novel			Overall
	small	medium	large	small	medium	large	
Action RGB	85.3%	81.1%	88.8%	74.2%	83.5%	71.3%	80.6%
	85.1%			76.2%			
Action depth	83.9%	91.7%	82.7%	81.5%	83.1%	64.4%	81.3%
	85.8%			76.9%			
Action RGB-D	91.6%	94.8%	83.3%	76.1%	86.3%	74.4%	84.4%
	90.0%			78.8%			
Baseline:	62.1%	51.9%	58.2%	40.2%	53.8%	54.5%	53.4%
Action Vector	57.8%			49.0%			
Baseline: Random	4.2%	15%	7.5%	0%	0%	5.0%	5.4%
	8.3%			2.4%			

Table 4.3: Quantitative evaluation of on-robot testing. These results are based on 2160 grasps in total. The success rate for each scenario was calculated from, on average, 83 grasps, except for the random policy where only, on average, 28 grasps were performed.

a familiar (objects similar to the training set) or novel (objects unlike those seen during training) set. This produced 6 different experimental setups: 3 levels of object count $\times$ 2 object sets.

After each scene setup, the robot made 3 (small), 5 (medium), or 10 (large) grasp attempts based on the size of the scene. Successfully grasped objects were placed in a basket behind the robot. Only objects placed in the basket were counted as success. Objects that fell out of the gripper during placing in the basket were counted as failure. Objects that were disturbed and fell off the table during grasp attempts were not returned to the table. This implicitly penalized grasp mistakes by reducing the available objects.

In total, we performed 18 *Action Image* evaluations, 3 scene sizes $\times$ 2 object sets $\times$ 3 networks, and an additional 12 baseline evaluations, 3 scene sizes $\times$ 2 object sets $\times$ 2 benchmark policies (action-vector and pure-random). The random policy was simply to query a single sample from the action sampler and execute the proposal. Table 4.3 reports the quantitative results of the real-world experimental evaluation based on 2160 grasps in real world. The success rate 5.4% from the random policy provides the lower-bound and illustrates the difficulty of the task and the experiment setup. Overall, every *Action Image* model achieved a success rate over 80%, while the action-vector baseline model only achieved a 53.4% success rate. Note, *Action Image* performs well from sim-to-real, with between 4 – 8 percent points decrease whereas Action Vector dropped by 15 percent points. This indicates that *Action Image* indeed allows to better generalize to the real world.

The results suggest that the medium-sized scene with familiar objects. This is likely because familiar objects were generally easier to grasp and the medium-sized scenario provided a goldilocks case where there was a minimum of clutter and plenty of objects to grasp. The small scene actually turned out to be more difficult than the medium scene due to the scarcity

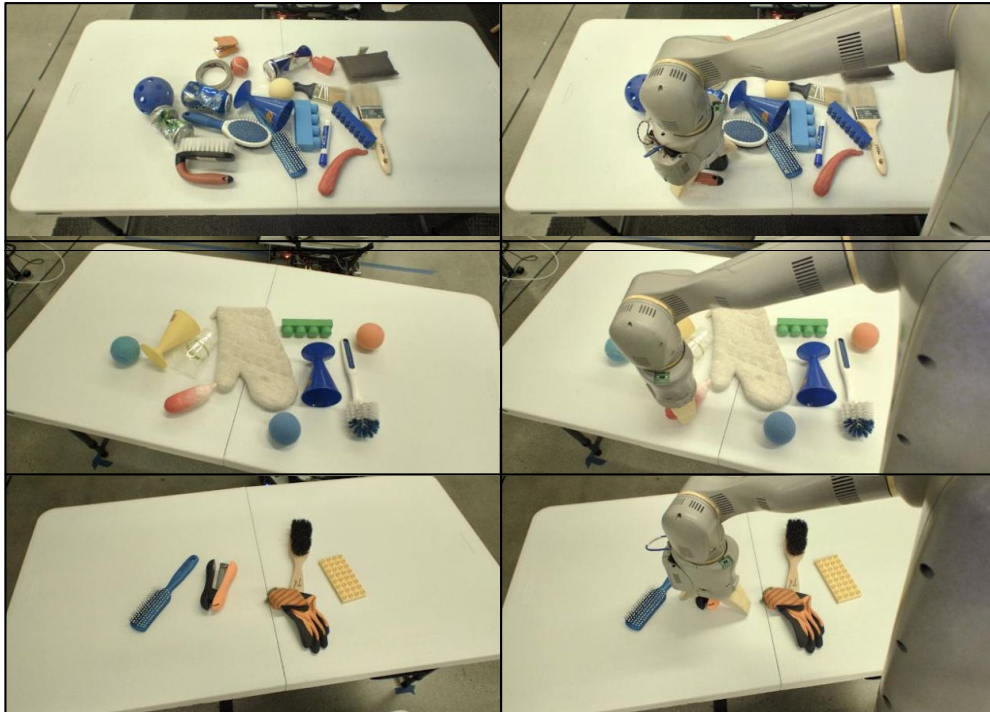


Figure 4.10: Snapshots of the grasping attempts on novel objects in the scenes large, medium, and small (from top to bottom). Images correspond to the initial scene (left) and the time of grasp (right).

of the objects. The base randomization before at the beginning of each grasp could potentially make some of the objects kinematically unreachable, hence giving the network less options for grasping. As expected the large scene with novel objects correspond to the hardest scenario due to the difficulty finding a good grasp for an object and the significant visual and physical differences between the novel and familiar objects.

Another interesting result was that all models, including action-vector, actually generalized decently to the novel objects. However, *Action Image* models with depth data were relatively more over-fit. In the medium-sized scene action-RGB-D decreased in performance for the novel objects (familiar at 95% vs. novel at 86%) while action-RGB actually improved slightly (familiar at 81% vs. novel at 84%).

4.5 Discussion

In this section we hypothesized that the action representation for visual-based grasping is key to learning critic networks efficiently and allowing them to generalize. We introduced *Action*

Image, a visual representation of grasps to better exploit the structural assumptions implicitly encoded in convolutional neural networks. By simply rendering gripper feature into image-space and providing a cropped, focused image-based information to the network we achieved over 80% real-world grasping performance without training on any real-world examples. In our experiments, we showed that *Action Image* representations can easily incorporate additional image-based information, such as depth, and outperforms vector-based action representations. We confirm an established observation that grasping is an object-centric, local problem by demonstrating that *Action Image* with image-space cropping and diverse simulation training data can achieve high rates of grasp success. Furthermore, we provide more evidence that domain transfer and object generalization is achievable, strengthening the theory that grasping behaviors can be learned in simulation and transferred to real-world robots.

Chapter 5

Learning active pose estimation with domain randomization

5.1 Introduction

In the past decades, robots have become dominant in industrial automation. A recent trend in manufacturing is the move toward small production volumes and high product variability [138], where reducing the manual engineering for automation becomes important. For automating many industrial tasks, such as picking, binning, or assembly, accurate pose estimation is essential. In this work, we focus on model based pose estimation from RGB cameras. This setting is relevant to many industrial applications, where a 3D model of the objects can easily be obtained, while it does not require expensive hardware such as high-precision depth cameras [139], [140], nor making modifications to the object such as adding markers [141]. Methods using markers often require significant human effort and have limited accuracy when the marker is far away or perpendicular to the image plane.

While a number of methods have been proposed for model-based pose estimation using expensive depth cameras or extensive labelled datasets in the real world [8], [142], the cost and manual effort required for these methods prevent them from being widely and easily applicable. Recently proposed methods proposing leveraging simulation as a tool for model-based pose estimation given accurate models of objects in an environment [134], [143], [144]. These methods are typically trained by leveraging known poses in simulation and training pose estimators which transfer effectively to the real world, bridging the simulation to reality gap.

In [143], it was shown that domain randomization was able to reach a 1.5 cm error on 3D pose estimation. Many robotic tasks, such as assembly or bin placing, require a much higher precision. In this work, we investigate how to improve the accuracy of pose estimation based on domain randomization such that it is suitable for high precision robotic assembly tasks.

In this work, we aim to improve the accuracy of domain randomization based pose estimation by making the observation that robots don't have to be passive observers of a scene and can in fact interact with objects in the scene. We can perform a *known geometrical*

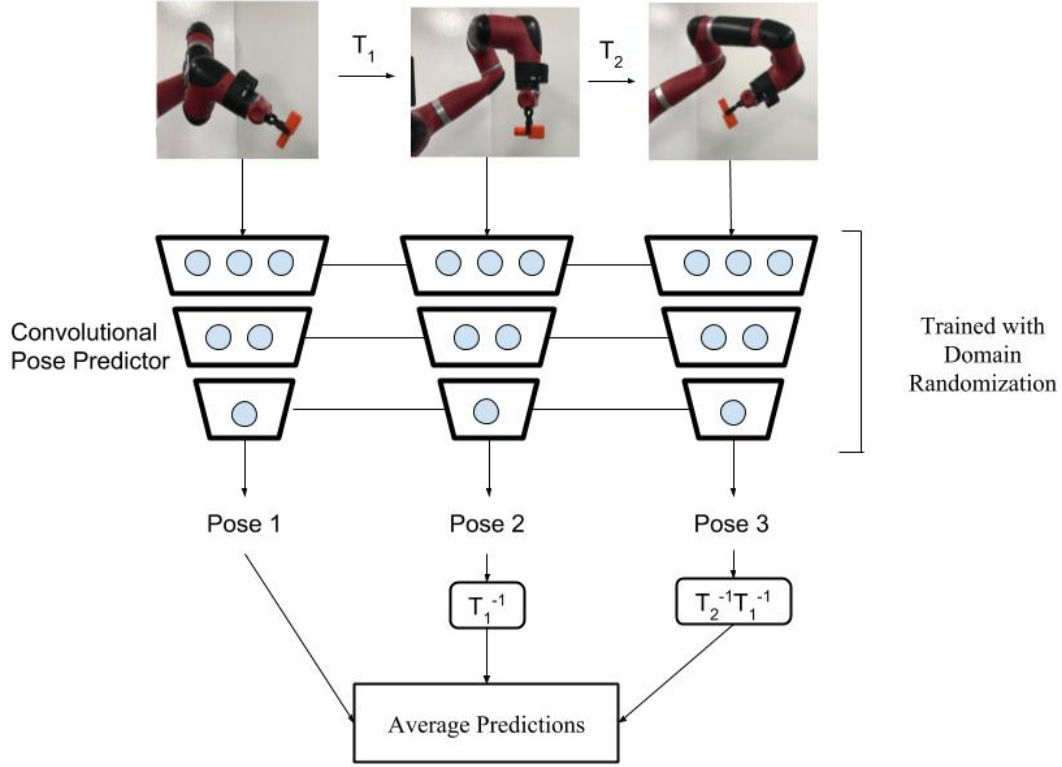


Figure 5.1: Inverse Transform Domain Randomization: We show that we can improve the accuracy of real world pose prediction with multiple images of a scene with known geometric transformations between object poses in the scenes.

transformation to the scene, such as moving objects in the scene, moving the arm or distractors, or changing the camera angle. Since this transformation between scenes is known and applied by the robot, all of these data-points can be used in order to improve the accuracy of pose predictions. We use this idea to propose a method for *active* pose estimation, which exploits the fact that being able to see an object from different angles and in different positions leads to a more accurate and robust predictions. In this way, we find that actually leveraging consistency among multiple different images of a scene ensures a much more accurate pose estimation compared to standard ensemble methods such as domain randomization.

Using models for active pose estimation transferred from simulation, we are able to decrease the average error predicted on real camera image from 2 cm to under 0.5cm, which is sufficient to enable a variety of high precision robotic manipulation tasks which were

otherwise very challenging with current methods.

5.2 Related work

Our work has several connections to a number of prior works from various fields

2D model based pose estimation Model based pose estimation of rigid objects from a 2D image has been studied extensively, largely building on a predefined feature points [145]–[148], edge detectors [149], [150], or image templates [151]. We refer to [152] for an extensive survey. Most of these algorithms rely on careful selection of the features to track, or on textured surfaces for points matching, and a careful calibration of the RGB camera. Our approach is agnostic to these factors.

3D model based pose estimation Using a depth camera, high precision pose estimation can be obtained [139], [140]. However, accurate depth cameras (e.g. a Photoneo) can be very expensive, limiting their use in many applications. Our approach only requires a 2D RGB image.

Fiducial markers The use of fiducial markers has become popular in augmented reality and robotics applications [141], [153], [154]. However, in realistic industrial applications, adding fiducials to objects may be undesirable, and the accuracy of fiducial based pose detection is limited for certain poses (for example, when the fiducial is perpendicular to the image plane). Our approach does not require any external modification of the object for pose detection.

Pose estimation based on supervised learning Several recent studies learned to map an image directly to pose using deep convolutional neural networks (CNNs) [8], [142]. While the CNN structure in these works is similar to ours, these works require a labeled training set for learning, which can be difficult to obtain. The domain randomization approach, in contrast, generates its own training data by rendering in simulation.

Active perception The study of active perception [155], [156] concerns how a robot should take actions to better estimate parameters of its environment. To our knowledge, our work is the first to study active perception in a simulation-to-real setting.

Domain randomization The gap between simulation and reality has been challenging robotics for decades. Recent work on trying to bridge this gap learns a decision making policy in simulation that works well under a wide variation in the simulation parameters, with the hope of learning a robust policy that transfers well to the real world. This idea has been explored for navigation [144] and pose estimation [143], by varying visual properties in the

scene, and also for locomotion [35] and grasping [143], by varying dynamics in simulation. In this work, we consider variation in the visual domain, and combine domain randomization with active perception, to improve its accuracy in pose detection.

5.3 Method

Problem Formulation and Preliminaries

We consider a model-based rigid body pose estimation problem. In our setting, we assume that we have geometrical 3D models of an object x and some reference object y . Let O_y denote a coordinate frame relative to y , and let P_x denote the 6D pose of x in the coordinate frame O_y . We are given an image of the scene I , that contains x and y , and our goal is to estimate P_x from the image.

Tobin et al. [143] proposed a domain randomization method for solving the pose estimation problem described above. In this method, a 3D rendering software is used to render scene images with different poses of x and y , and random textures, lighting conditions, camera orientations, and camera parameters. Let $D = \{I^1, P_x^1, \dots, I_N, P_x^N\}$ denote the data set of the rendered images and matching object poses (which are known, by construction). Supervised learning is then used to train a deep neural network mapping I to P_x . Since the network is trained to work on various texture, camera, and lighting conditions, it is expected that it also works on real world images since their statistics would roughly fall under the extremely wide distribution that was trained on. By making the training distribution extremely broad in terms of components such as texture, camera, and lighting, this method is able to ensure generalization to real world test environments by reducing the covariate shift. Indeed, the method in [143] reportedly obtained an average 1.5 cm error in predicting 3D pose on real world test images.

In this work, we propose an active perception approach based on domain randomization. To motivate our approach, we start by discussing the working hypothesis underlying domain randomization:

Working Hypothesis (Domain Randomization): *There exists a set of features that can be extracted from all images in the data and are sufficient for predicting the image label (pose). These features can also be extracted from real images and are sufficient for predicting the real label.*

This working hypothesis means that if the training data is sufficiently randomized, and the neural network is expressive enough, then with enough data, the model has to discover the features which are common to all images, and base its prediction only on these features (otherwise it would suffer a higher training loss on spurious correlations that it picks up on). In that case, the network predictions are likely to transfer well to the real world.

One may question whether such features should even exist. However, for pose prediction, we know that the relative pose P_x is a purely geometrical property of the objects, and since we assume an accurate 3D model of x , then geometrical properties (e.g., relative sizes and

shapes) should be maintained in all the rendered images and also in the real images. Thus, the network has the potential to learn predictions based solely on geometrical properties of objects, abstracting away any other visual cues such as textures and lighting, and such features should transfer well to real images.

As discussed thus far, this pose estimation process is done completely passively. The robot does not interact with objects in the scene, but simply observes a single image of the scene and needs to predict the pose. In this work, we provide a key insight that we *can* in-fact interact with the scene, and apply known geometric transformations to objects in the scene. These transformations allow us to obtain a number of different images of the scene to estimate the pose of the object as the transformations are all *applied by us*. In this sense, we propose an active procedure to improve pose estimation by interacting with the scene and using multiple images to make a better prediction.

Active Perception based on Domain Randomization with Geometric Transformations

Recall that in the standard domain randomization problem, training data is in the form of image-pose pairs, $\{I, P_x\}$. Following the active perception paradigm [155], we can apply to the scene some *known* geometric transformation, with the hope that it improves our perception capabilities. For example, consider a robotic arm grasping an object, and the problem of estimating the position of the object within the robot’s gripper. In this case, we can move the gripper closer to the camera to obtain a better pose estimate. Since we know the transformation applied when moving the gripper, we can potentially combine several images to obtain a better prediction. As another example, consider moving the camera to obtain a better view of the object.

Concretely, we define the **Domain Randomization with Geometric Transformations** problem (DR-GT). let $T_1, \dots, T_k$ denote a set of k transformations that can actively be applied to the geometry of the scene *both in the real world and in simulation*. In particular, we consider rigid body transformations applied to objects in the scene and to the camera [157]. We propose to generate training data in the form of tuples $\{I, T_1(I), \dots, T_k(I), P_x, T_1(P_x), T_k(P_x)\}$, where, slightly abusing notation, we denote by $T_i(I)$ and $T_i(P_x)$ the rendered image and pose when applying transformation T_i to the scene. The supervised learning problem we consider now is learning a mapping from $I, T_1(I), \dots, T_k(I), T_1, \dots, T_k$ to P_x .

Inverse Transform based Domain Randomization

To solve the DR-GT problem, we propose the following method, based on inverse transforms. Let T_i^{-1} denote the inverse transform of T_i ¹. Let f be the standard domain randomization

¹We restrict our approach to transformations with a well-defined inverse, such as rotations and translations.

mapping from I to P_x . Then, we propose to calculate

$$\begin{aligned} P_{x;0} &= f(I), \\ P_{x;1} &= T_1^{-1}(f(T_1(I))), \\ &\dots, \\ P_{x;k} &= T_k^{-1}(f(T_k(I))). \end{aligned} \tag{5.1}$$

Note that for each $i \in 0, \dots, k$, the inverse transformation in (5.1) means that the prediction $P_{x;i}$ is an estimate of P_x . Therefore, we can predict P_x as the sample average:

$$\hat{P}_x = \frac{1}{k+1} \sum_{i=0}^k P_{x;i}.$$

We term this method Inverse Transform based Domain Randomization (ITDR). We expect that as we enlarge the number of transformations k , the precision of ITDR improves.

While it is seemingly naive to use the sample average as a prediction, we found that it is surprisingly effective compared to more complicated methods with models which consider several images at once as input and produce a single pose estimate directly.

The key intuition behind using ITDR for improved estimation is that using known transformations in an environment allows us to use a wider data distribution to make several predictions of the same pose. Since several of these transformations provide easier to model prediction problems than the original problem, it makes the accuracy of the model significantly higher in the real world.

Model Architecture

In order to perform accurate pose estimation directly from images, we used a convolutional neural network architecture [1]. The neural network takes a single image as input, and generates a pose as output. In our experiments we investigated predicting 3DoF pose composed of 2DoF translation and 1DoF rotation. The model architecture takes in an RGB image through 16 convolutional layers, each two convolutional layers is followed by a max-pooling operation and a ReLU nonlinearity. These convolutional layers are followed by 3 fully connected layers with decreasing hidden units and ReLU nonlinearity. This architecture is similar to the one used in [143], based on the VGG architecture [13] using convolution layers pretrained on ImageNet. The loss function for training this model is a combination of L1 regression loss for the 2DoF translation, and a cosine loss for the orientation, given by

$$L(x, \theta) = \|x - \hat{x}\| + \|\cos(\theta - \hat{\theta}) - 1\|,$$

where x and $\hat{x}$ are the true and predicted pose, and θ and $\hat{\theta}$ are true and predicted orientation.

For active pose estimation, we pass a number of different images through the same network and then average the predictions *after* applying a known rigid transform between them, as described in the ITDR algorithm.

In Simulation



In Real Life

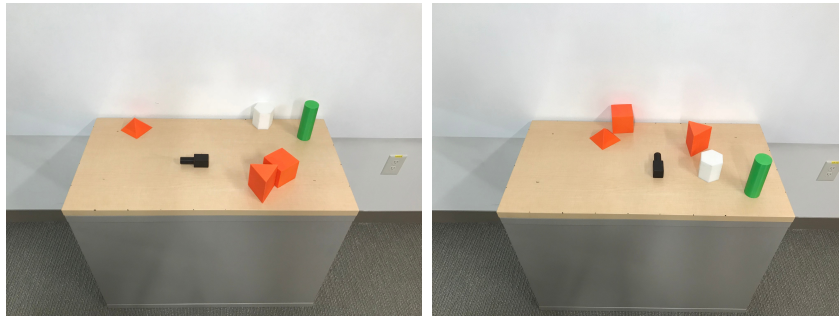


Figure 5.2: Simulation to reality transfer with active reference object tabletop movement

5.4 Experiments

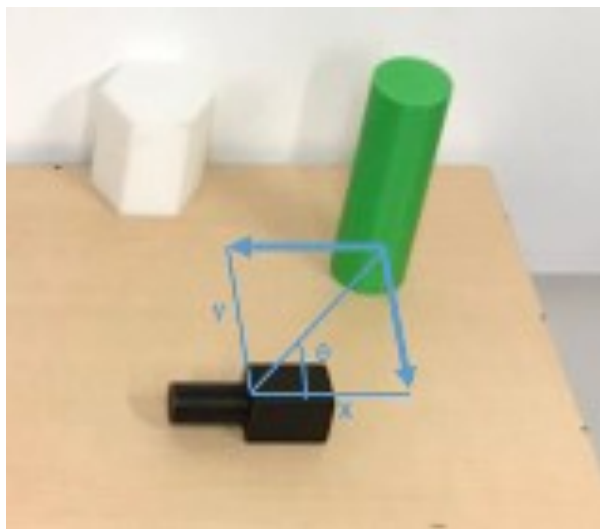
In this section we report our experiments on active perception using domain randomization. In our investigation, we aim to determine whether geometric transformations can give significantly better performance for model-based pose estimation in the real world. We designed our experiments to investigate whether using the robot to actively move elements of the scene yields gains in estimating the pose of objects in the scene. In particular, we investigate the performance of ITDR in the following situations:

1. Moving reference objects in the environment,
2. Moving a robot manipulator holding an object, and
3. Moving a camera held by the robot.

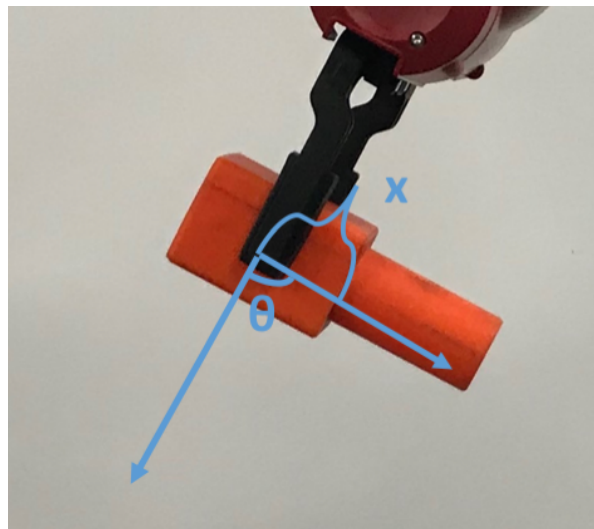
We believe that these experiments showcase the potential of active perception used within domain randomization.

Average error	x [cm]	y [cm]	θ [radians]
Using only one image	1.57	1.10	0.065
Reference Object on Diagonal	0.64	0.456	0.025
Reference Object in Parallel	1.17	0.47	0.038
Reference Object on Four Corners	0.62	0.46	0.037

Table 5.1: Table showing the mean prediction error for pose prediction for the moving reference object scenario described in Section 5.4. This table shows that performance of pose prediction transferred from simulation to reality can be significantly improved by using the robot actively to modify the scene being considered. Prediction error is low when the reference object is placed at all four corners or on diagonal corners rather than along an edge of the table. This indicates that multiple images with known transformations do help reduce pose prediction error, and the choice of which transforms to use has a significant effect on the prediction error.



(a) Relative pose measurements with respect to green reference object (Section 5.4)



(b) Relative pose measurements when the object is grasped in the gripper (Section 5.4)

Figure 5.3: Experiment setups: (a) estimate pose relative to movable reference object. (b) estimate pose relative to robot gripper.

Moving Reference Objects

We consider estimating the pose of a peg-shaped object x resting on a table, similar to the experimental setting of [143] (see Figure 5.3a for more details). This is an important task for

estimating where to grasp an object for further manipulation. Since the table has a fixed height, and the object is at rest, the relevant degrees of freedom are 3-dimensional – the 2D position of the object center and its orientation.² We predict the pose of x with respect to the green cylinder object y . For active perception, we consider moving the reference object y between a fixed set of 4 points in the corners of the table, and using ITDR to predict the pose from all images. We expect that actively moving objects in the scene will give us much more accurate pose estimation than if we had just a single image since the diversity of data to predict from is larger.

In Table 5.1 we show the results of pose estimation from a single image, and compare to using multiple images using ITDR. The improvement is on the order of 3x bringing down the estimation error to sub-centimeter ranges which enables a number of high precision applications.

We also investigate how to choose the best set of transformations for the reference object such that the pose estimation error is minimized. It is preferable for us to choose as few points as possible while ensuring accurate pose estimation. We choose pairs amongst the four points shown in Figure 5.5 to evaluate whether particular transformations are more effective than others. In Table 5.1 we report the error of each pair of possible reference object positions on real data. We see that the real world error when moving object y to the diagonal corners is significantly lower than if we moved the object to corners which share an edge. This is likely caused because it gives a more significant difference in the relative poses.

This experiment helps us understand the effect of moving objects in the scene actively in enabling better pose estimation. We see that simply moving reference objects in the scene and using the known geometric transformations allows us to estimate pose more accurately for real-world peg insertion tasks.

Moving Robot Manipulator Holding an Object

From the results above, we see that moving reference objects in the scene to get a wider variety of relative poses significantly helps with pose estimation. Alternatively, we can consider a scenario where an object has already been grasped, but its position within the gripper is not known accurately. This would be a typical case when the pose estimation before grasping is not perfect. It is also important in robotic reinforcement learning experiments [97], where during learning, interaction with other objects in the environment can move an object that is grasped within the gripper.

As in the previous section, the object x is peg-shaped, while the reference object y in this case is the robot gripper. We estimate a 2-dimensional pose: the distance of the center of the object from the gripper, and its orientation within the gripper. The measured quantities are depicted in Fig 5.3b. These are challenging to estimate with extreme precision but are extremely important for the tasks we consider.

²Note that Tobin et al. [143] only predicted the 2D position, while here we also predict orientation.

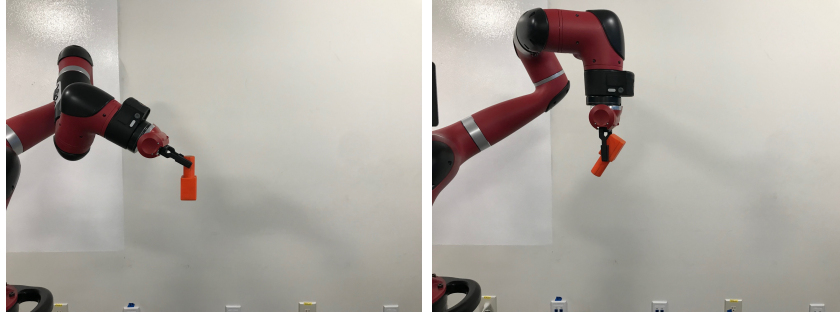
In Simulation**In Real Life**

Figure 5.4: Simulation to reality transfer with active gripper motion. Left: simulated images with domain randomization. Right: real images. Active perception here is based on moving the robot gripper.

For active perception, in this case we move the robot gripper between a set of 5 fixed positions, and use ITDR to estimate the pose from all images. The different movements of the gripper show the camera different elements of the object itself, which is likely to help with better pose estimation since the model can latch on to different parts of the object. We find that this strategy indeed helps with pose estimation in the real world. We are able to identify the offset and the angle of the object grasped within the gripper significantly more accurately. ITDR performs significantly better than the baseline of simply using a single image and a model trained with domain randomization. As we can see from Table 5.2, the x position error is improved by around 20% and the angle accuracy is improved by around $3\times$, from 0.129 to 0.047. Additionally, we find that using fewer gripper locations leads to worse performance. This suggests that using the multiple images does indeed improve performance and scales with using more images for estimation. Note that in this setting, the object does not change pose with respect to the gripper, therefore the inverse transformations in ITDR are just the identity.

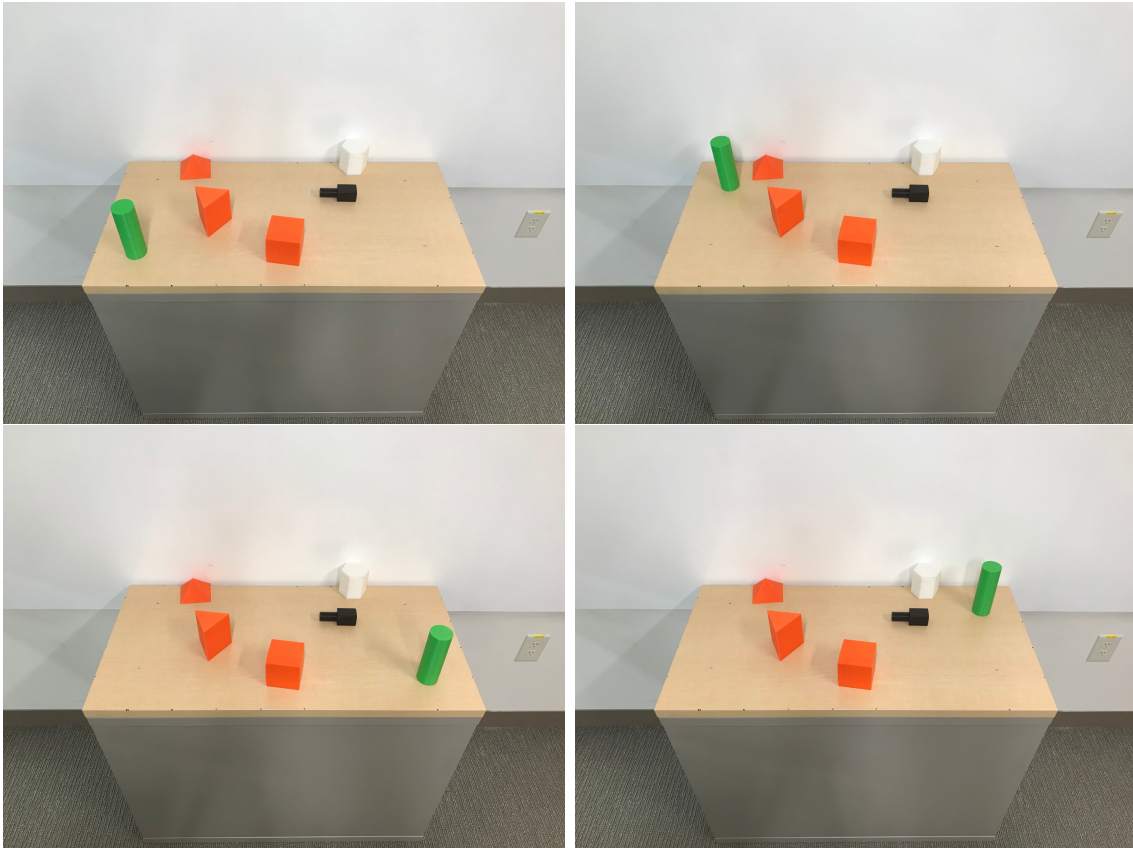


Figure 5.5: Different transform applied to the reference object as described in Section 5.4. The green cylinder is the reference object and we are estimating the pose of the black peg. As seen from these figures we can transform the position of the green cylinder to four different positions and use multiple images to improve pose estimation

Average error	x [cm]	θ [radians]
one image	0.30	0.129
five images	0.26	0.047
two images	0.27	0.086

Table 5.2: Table showing the mean prediction error for pose prediction for moving gripper scenario described in Section 5.4. We see that using multiple images with known geometric transformations is able to significantly reduce the angle error and provide some improvements in the estimation of the offset of the gripped object as well.

In this experiment, we demonstrate that actively moving the camera can improve the pose prediction performance. Figure 5.6 depicts our experimental setup.

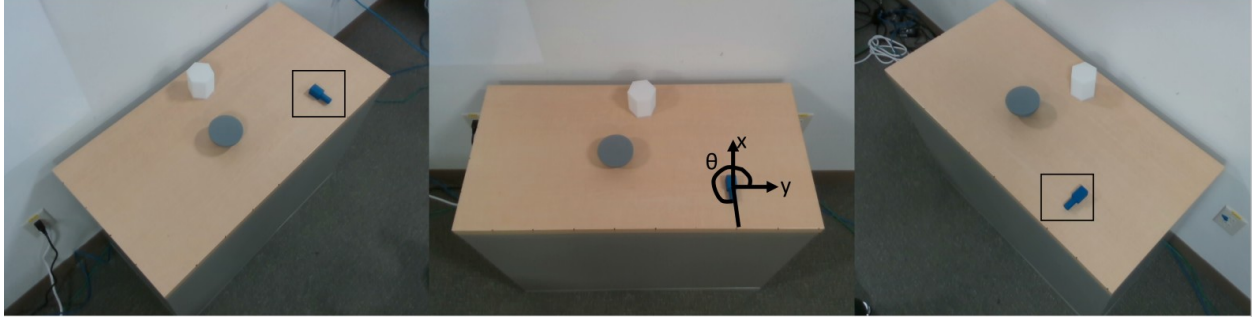


Figure 5.6: Different camera angles of the same setting with target object boxed up. We want to estimate the position and orientation of the target object relative table corner. We can use multiple observing angles to explore the geometric properties of the target object and achieve a better pose estimation.

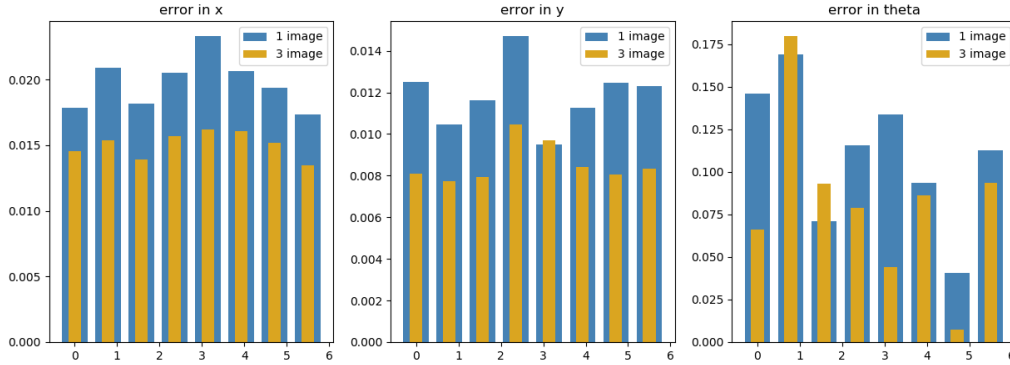


Figure 5.7: Distribution of the error in horizontal direction, vertical direction, and θ with respect to the orientation of the object(θ). Using one image, we see a high prediction error when the peg is orientated away from the camera (θ around 3.14) due to the asymmetric shape of the peg. When using three images, the prediction for this previous difficult object orientation is significantly improved.

Moving a Robot-Held Camera

As in Section 5.4, we estimate the pose of a peg-shaped object x resting on a table among various distractor objects, and a fixed reference object y . To obtain more contrasting result,

Average error	x [cm]	y [cm]	θ [radians]
Using only one image	1.97	0.12	0.11
Using three images	1.50	0.08	0.08

Table 5.3: Table showing the mean prediction error for pose prediction for the moving camera scenario described in Section 5.4. We see that using multiple images taken from different point of view is able to reduce both the coordinates and angle error.

the peg is 0.7^3 times smaller than the one use in 5.4. Here, we mounted our camera to the robot’s end effector, and we actively change the viewpoint by moving the robot arm to various positions. In particular, we chose a set of three fixed positions in which we can view the object from. Note that similarly, to Section 5.4, the object does not change pose with respect to the reference, therefore the inverse transformations in ITDR are just the identity.

In Table 5.3, we show the results for our method. We observe a significant improvement in pose prediction for the x, y coordinates the object. To better understand these results, in Figure 5.7 we plot the prediction error as a function of the orientation of the object. We observe that for some orientations, estimating the pose from a single viewpoint is very difficult, which is attributed to the asymmetric shape of the peg – when placed such that it is perpendicular to the camera plane, most of the object is occluded, making it difficult to estimate a correct orientation. In test cases, adding additional viewpoints significantly improves the results.

5.5 Discussion

In this work, we evaluated the use of active perception within the domain randomization paradigm. We have shown that active perception strategies which are able to interact with objects in the scene with known geometric transformations can significantly improve the performance of pose estimation compared to passive perception approaches. In particular, we have reduced the 1.5cm pose estimation error in domain randomization state-of-the-art to less than 0.6cm, which can lead to new robotic capabilities in downstream tasks such as tight fitting assembly problems.

In future work, we intend to explore additional methods for improving the performance of domain randomization based pose estimation, for example, in a semi-supervised setting where unlabeled images from the real domain are available to the learning algorithm.

Chapter 6

Conclusions

In this thesis, we discuss how to learn robotic skills efficiently from several perspectives: injecting suitable policy priors, constructing generalizable representation, using simulation data, etc. Our main motivation is to explore most suitable ways to leverage prior information in robot learning; making the problem more structured in that the learning part could mostly focus on generalizing instead of reinventing every nut from scratch. Despite our various attempts, we are not concluding that our methods are only ways to solve robot learning problems neither do we state that end-to-end learning methods are not suitable for robot learning. No one has a concrete answer on how to improve or manipulate a complex dynamic system by best leveraging massive data available, we should best exploit structures and properties for different problems.

As we show in chapter 2, injecting suitable policy prior indeed helps us speed up the learning process a lot; but we are getting this advantage at the cost of potentially being sub-optimal: the amount of residual correction that the RL agent can do has an upper bound that is proportional to the divergence between prior policy and the “optimal policy”. In other words, if the prior policy is far away from the “optimal policy”, the RL agent would never reach it. However, this provable property does not mean that residual RL is fundamentally limited. Instead, this inspires us to analyze the problem in hand more carefully rather than putting a brute-force learning agent everywhere. That’s the message we try to convey throughout this thesis, there won’t be a “panacea” to all problems we are solving in robotics, different problem has different properties that we should consider.

In chapter 3, we introduce an action space for RL algorithms, which shows superior performance in terms of learning efficiency and success rate in several high-precision assembly tasks. This method essentially moves all action calculations from joint space to task space with an additional stabilizing torque term; this adjustment makes a lot sense because task space is the space we care about; sampling actions in this space thus becomes meaningful. In this regard, we are also injecting a prior into the learning system since the operational space controller is a robot specific formulation that’s widely studied in classical robotics. As we can see in the results, injecting such a prior dramatically increase the performance of learned policy, this layer of abstraction together with RL can serve as a powerful tool for solving

contact-rich high-precision robotic manipulation problems.

In chapter 4 and 5, we show how a good geometric representation can dramatically increase performance on two specific domains: grasping and pose estimation; we only use simulation data for training in these two scenarios. For the grasping problem, the gripper representation “action image” serves as a crucial ingredient in the successful sim-to-real transfer because of its geometric invariance. For the pose estimation problem, rigid transformations between robot frames also contribute to the success of sim-to-real transfer again because of its geometric invariance. These two vivid examples inspire us to think more about such beautiful properties amongst robotic problems: once integrated properly, the learning component can play a powerful role in generalizing to new scenes owing to the shared discovered properties.

We argue that in certain hard robotic tasks such as high-precision manipulation, large-scale grasping, etc., injecting such prior information is the key to make learning algorithm work and generalizable. An end-to-end learning algorithm has its advantage for being universal applicable, but it is less suitable when the complexity of the problem scales up or tremendous additional engineering needs to be done to make such algorithm work. While injecting priors into learning process could make the final solution biased towards a certain class of problems, we should actually analyze such bias more deeply, quantify them in a way that we can systematically design such priors for different kinds of problems.

We are just entering an exciting new area where AI and robotics merges, AI is certainly unlocking great potentials for robotics, we are firm believers the right mixture between classical robotics and AI would bring an exciting time for researchers in robotics.

Bibliography

- [1] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.
- [2] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning”, *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. DOI: [10.1038/nature14539](https://doi.org/10.1038/nature14539). [Online]. Available: <https://doi.org/10.1038/nature14539>.
- [3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition”, *Neural Computation*, vol. 1, pp. 541–551, 1989.
- [4] K. Park, A. Mousavian, Y. Xiang, and D. Fox, *Latentfusion: End-to-end differentiable reconstruction and rendering for unseen object pose estimation*, 2019. arXiv: [1912.00416](https://arxiv.org/abs/1912.00416) [cs.CV].
- [5] X. Deng, Y. Xiang, A. Mousavian, C. Eppner, T. Bretl, and D. Fox, *Self-supervised 6d object pose estimation for robot manipulation*, 2019. arXiv: [1909.10159](https://arxiv.org/abs/1909.10159) [cs.R0].
- [6] X. Deng, A. Mousavian, Y. Xiang, F. Xia, T. Bretl, and D. Fox, “Poserbpf: A rao-blackwellized particle filter for 6d object pose tracking”, in *Robotics: Science and Systems (RSS)*, 2019.
- [7] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, “Deepim: Deep iterative matching for 6d pose estimation”, in *European Conference Computer Vision (ECCV)*, 2018.
- [8] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, “Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes”, *arXiv preprint arXiv:1711.00199*, 2017.
- [9] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection”, *CoRR*, vol. abs/1506.02640, 2015. arXiv: [1506.02640](https://arxiv.org/abs/1506.02640). [Online]. Available: <http://arxiv.org/abs/1506.02640>.
- [10] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement”, 2018. arXiv: [1804.02767](https://arxiv.org/abs/1804.02767). [Online]. Available: <http://arxiv.org/abs/1804.02767>.
- [11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, “SSD: single shot multibox detector”, *CoRR*, vol. abs/1512.02325, 2015. arXiv: [1512.02325](https://arxiv.org/abs/1512.02325). [Online]. Available: <http://arxiv.org/abs/1512.02325>.

- [12] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection”, *CoRR*, vol. abs/1708.02002, 2017. arXiv: [1708.02002](https://arxiv.org/abs/1708.02002). [Online]. Available: <http://arxiv.org/abs/1708.02002>.
- [13] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition”, *arXiv preprint arXiv:1409.1556*, 2014.
- [14] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision”, in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016. DOI: [10.1109/CVPR.2016.308](https://doi.org/10.1109/CVPR.2016.308).
- [15] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks”, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., pp. 91–99, 2015. [Online]. Available: <http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf>.
- [16] I. Lenz, H. Lee, and A. Saxena, “Deep learning for detecting robotic grasps”, *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.
- [17] L. Pinto and A. Gupta, “Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours”, in *2016 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2016, pp. 3406–3413.
- [18] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection”, *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.
- [19] C. Finn and S. Levine, “Deep visual foresight for planning robot motion”, in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 2786–2793.
- [20] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies”, *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.
- [21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: <http://incompleteideas.net/book/the-book-2nd.html>.
- [22] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing Function Approximation Error in Actor-Critic Methods”, *arXiv preprint arXiv:1802.09477*, 2018.
- [23] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, “Trust region policy optimization”, in *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2015.
- [24] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning”, *arXiv preprint arXiv:1312.5602*, 2013.

- [25] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning”, *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 00280836. [Online]. Available: <http://dx.doi.org/10.1038/nature14236>.
- [26] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search”, *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. DOI: [10.1038/nature16961](https://doi.org/10.1038/nature16961).
- [27] J. Luo, R. Edmunds, F. Rice, and A. M. Agogino, “Tensegrity robot locomotion under limited sensory inputs via deep reinforcement learning”, in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 6260–6267.
- [28] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, “Deepmimic: Example-guided deep reinforcement learning of physics-based character skills”, *ACM Trans. Graph.*, vol. 37, no. 4, Jul. 2018, ISSN: 0730-0301. DOI: [10.1145/3197517.3201311](https://doi.org/10.1145/3197517.3201311). [Online]. Available: <https://doi.org/10.1145/3197517.3201311>.
- [29] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine. (2020). Learning agile robotic locomotion skills by imitating animals. arXiv: [2004.00784](https://arxiv.org/abs/2004.00784).
- [30] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver. (2017). Emergence of locomotion behaviours in rich environments. arXiv: [1707.02286](https://arxiv.org/abs/1707.02286).
- [31] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, and A. M. Agogino, “Deep reinforcement learning for robotic assembly of mixed deformable and rigid objects”, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 2062–2069.
- [32] J. Luo, E. Solowjow, C. Wen, J. Aparicio Ojea, A. Agogino, A. Tamar, and P. Abbeel, “Reinforcement learning on variable impedance controller for high-precision robotic assembly”, in *ICRA*, 2019.
- [33] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates”, *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, ISSN: 0028-0836. DOI: [10.1038/nature20101](https://doi.org/10.1038/nature20101). arXiv: [1610.00633](https://arxiv.org/abs/1610.00633). [Online]. Available: <https://arxiv.org/pdf/1610.00633.pdf%20http://arxiv.org/abs/1610.00633>.

- [34] J. Peters and S. Schaal, “Reinforcement learning of motor skills with policy gradients”, *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008, ISSN: 08936080. DOI: [10.1016/j.neunet.2008.02.003](https://doi.org/10.1016/j.neunet.2008.02.003), arXiv: [arXiv:1411.3159v1](https://arxiv.org/abs/1411.3159v1). [Online]. Available: <https://pdfs.semanticscholar.org/eb5b/459c8a3e56064158fb3514eeab763486e437.pdf>.
- [35] I. Mordatch, K. Lowrey, and E. Todorov, “Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids”, in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, IEEE, 2015, pp. 5307–5314.
- [36] U. Kannengiesser, R. Heininger, L. Billy, P. Terpak, M. Neubauer, C. Stary, D. Majoe, A. Totter, and D. Bonaldi, *Lot-Size One Production*. Jan. 2017, pp. 69–111, ISBN: 978-3-319-48465-5. DOI: [10.1007/978-3-319-48466-2_4](https://doi.org/10.1007/978-3-319-48466-2_4).
- [37] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”, in *International Conference on Machine Learning*, 2018. arXiv: [arXiv:1801.01290v2](https://arxiv.org/abs/1801.01290v2). [Online]. Available: <https://arxiv.org/pdf/1801.01290.pdf>.
- [38] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning”, in *AAAI*, 2018.
- [39] A. Y. Ng, D. Harada, and S. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping”, in *International Conference on Machine Learning (ICML)*, 1999. [Online]. Available: <https://www-cs.stanford.edu/people/ang/papers/shaping-icml99.pdf>.
- [40] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-maroon, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, M. Riedmiller, and M. R. Deepmind, “Data-efficient Deep Reinforcement Learning for Dexterous Manipulation”, *CoRR*, vol. abs/1704.0, 2017. arXiv: [1704.03073](https://arxiv.org/abs/1704.03073). [Online]. Available: <https://arxiv.org/pdf/1704.03073.pdf>.
- [41] C. Daniel, M. Viering, J. Metz, O. Kroemer, and J. Peters, “Active Reward Learning”, in *Robotics: Science and Systems (RSS)*, 2014. [Online]. Available: <http://www.roboticsproceedings.org/rss10/p31.pdf>.
- [42] J. Peters, K. Mülling, and Y. Altün, “Relative Entropy Policy Search”, in *AAAI Conference on Artificial Intelligence*, 2010, pp. 1607–1612.
- [43] M. P. Deisenroth and C. E. Rasmussen, “PILCO: A model-based and data-efficient approach to policy search”, in *International Conference on Machine Learning (ICML)*, 2011, pp. 465–472. [Online]. Available: <http://mlg.eng.cam.ac.uk/pub/pdf/DeiRas11.pdf>.
- [44] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, “Robust Adversarial Reinforcement Learning”, in *International Conference on Machine Learning (ICML)*, 2017. [Online]. Available: <https://arxiv.org/pdf/1703.02702.pdf>.

- [45] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, “Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection”, *International Journal of Robotics Research*, 2017.
- [46] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning”, in *NIPS Workshop on Deep Learning*, 2013, pp. 1–9, ISBN: 1476-4687 (Electronic) 0028-0836 (Linking). DOI: [10.1038/nature14236](https://doi.org/10.1038/nature14236). arXiv: [1312.5602](https://arxiv.org/abs/1312.5602). [Online]. Available: <https://arxiv.org/pdf/1312.5602.pdf%20http://arxiv.org/abs/1312.5602%20https://www.cs.toronto.edu/%7B~%7Dvmnih/docs/dqn.pdf>.
- [47] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, “Continuous Deep Q-Learning with Model-based Acceleration”, in *ICML*, 2016, ISBN: 3405062780. DOI: [10.3390/robotics2030122](https://doi.org/10.3390/robotics2030122). arXiv: [1603.00748](https://arxiv.org/abs/1603.00748). [Online]. Available: <https://arxiv.org/pdf/1603.00748.pdf%20http://arxiv.org/abs/1603.00748>.
- [48] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine, “Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning”, in *ICML*, 2017. [Online]. Available: <https://arxiv.org/pdf/1703.03078.pdf>.
- [49] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. J. Johnson, and S. Levine, “SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning”, in *ICML*, Aug. 2019. arXiv: [1808.09105](https://arxiv.org/abs/1808.09105). [Online]. Available: <http://arxiv.org/abs/1808.09105>.
- [50] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar, “Dexterous Manipulation with Deep Reinforcement Learning: Efficient, General, and Low-Cost”, in *ICRA*, Oct. 2018. arXiv: [1810.06045](https://arxiv.org/abs/1810.06045). [Online]. Available: <http://arxiv.org/abs/1810.06045>.
- [51] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation”,
- [52] G. Thomas, M. Chien, A. Tamar, J. A. Ojea, and P. Abbeel, “Learning Robotic Assembly from CAD”, in *ICRA*, 2018. [Online]. Available: <https://arxiv.org/pdf/1803.07635.pdf>.
- [53] R. Loftin, J. Macglashan, B. Peng, M. E. Taylor, M. L. Littman, J. Huang, and D. L. Roberts, “A Strategy-Aware Technique for Learning Behaviors from Discrete Human Feedback”, in *AAAI Conference on Artificial Intelligence*, 2014, pp. 937–943.
- [54] W. Saunders, G. Sastry, A. Stuhlmüller, and O. Evans, “Trial without Error: Towards Safe Reinforcement Learning via Human Intervention”, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2018. arXiv: [arXiv: 1707.05173v1](https://arxiv.org/abs/1707.05173v1). [Online]. Available: <https://arxiv.org/pdf/1707.05173.pdf>.
- [55] L. Torrey and M. Taylor, “Teaching on a Budget: Agents Advising Agents in Reinforcement Learning”, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013.

- [56] J. Frank, S. Mannor, and D. Precup, “Reinforcement Learning in the Presence of Rare Events”, in *International Conference on Machine Learning (ICML)*, 2008. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.4737%7B%5C%7Drep=rep1%7B%5C%7Dtype=pdf>.
- [57] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, “Deep reinforcement learning from human preferences”, in *Advances in Neural Information Processing Systems (NIPS)*, 2017. arXiv: [1706.03741](https://arxiv.org/abs/1706.03741). [Online]. Available: <https://arxiv.org/pdf/1706.03741.pdf%20http://arxiv.org/abs/1706.03741>.
- [58] J. Kober and J. Peter, “Policy search for motor primitives in robotics”, in *Advances in Neural Information Processing Systems (NIPS)*, vol. 97, 2008, pp. 83–117, ISBN: 1099401052236. DOI: [10.1007/978-3-319-03194-1_4](https://doi.org/10.1007/978-3-319-03194-1_4). [Online]. Available: <http://papers.nips.cc/paper/3545-policy-search-for-motor-primitives-in-robotics.pdf>.
- [59] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine, “Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations”, in *Robotics: Science and Systems*, 2018. [Online]. Available: <https://arxiv.org/pdf/1709.10087.pdf>.
- [60] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys, “Learning from Demonstrations for Real World Reinforcement Learning”, in *AAAI Conference on Artificial Intelligence*, 2018. arXiv: [1704.03732](https://arxiv.org/abs/1704.03732). [Online]. Available: <https://arxiv.org/pdf/1704.03732.pdf%20http://arxiv.org/abs/1704.03732>.
- [61] M. Vecerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. M. O. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller, “Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards”, *ArXiv*, vol. abs/1707.08817, 2017.
- [62] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Overcoming Exploration in Reinforcement Learning with Demonstrations”, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018. [Online]. Available: <https://arxiv.org/pdf/1709.10089.pdf>.
- [63] C. Finn, S. Levine, and P. Abbeel, “Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization”, in *ICML*, 2016. [Online]. Available: <https://arxiv.org/pdf/1603.00448.pdf>.
- [64] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, “Maximum Entropy Inverse Reinforcement Learning.”, in *AAAI*, 2008, pp. 1433–1438. arXiv: [arXiv:1507.04888v2](https://arxiv.org/abs/1507.04888v2).
- [65] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine, “End-to-End Robotic Reinforcement Learning without Reward Engineering”, in *RSS*, 2019. arXiv: [1904.07854](https://arxiv.org/abs/1904.07854). [Online]. Available: <http://arxiv.org/abs/1904.07854>.

- [66] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. Aparicio Ojea, E. Solowjow, and S. Levine, “Residual Reinforcement Learning for Robot Control”, in *ICRA*, 2019. arXiv: [1812.03201v2](https://arxiv.org/pdf/1812.03201v2). [Online]. Available: <https://arxiv.org/pdf/1812.03201.pdf>.
- [67] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, “Residual Policy Learning”, Dec. 2018. arXiv: [1812.06298](http://arxiv.org/abs/1812.06298). [Online]. Available: <http://arxiv.org/abs/1812.06298>.
- [68] K. Subramanian, C. L. Isbell, and A. L. Thomaz, “Exploration from Demonstration for Interactive Reinforcement Learning”, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2016. [Online]. Available: www.ifaamas.org.
- [69] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition”, in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. arXiv: [1512.03385v1](http://image-net.org/challenges/LSVRC/2015/). [Online]. Available: <http://image-net.org/challenges/LSVRC/2015/>.
- [70] M. Srouji, J. Zhang, and R. Salakhutdinov, “Structured Control Nets for Deep Reinforcement Learning”, in *International Conference on Machine Learning (ICML)*, 2018. arXiv: [1802.08311](http://arxiv.org/abs/1802.08311). [Online]. Available: <http://arxiv.org/abs/1802.08311>.
- [71] E. Johnson and A. Calise, “Pseudo-Control Hedging: A New Method For Adaptive Control”, in *Advances in Navigation Guidance and Control Technology Workshop*, 2000.
- [72] R. Li, R. Platt, W. Yuan, A. Ten Pas, N. Roscup, M. A. Srinivasan, and E. Adelson, “Localization and Manipulation of Small Parts Using GelSight Tactile Sensing”, in *IROS*, 2014. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.448.7123%7B%5C%7Drep=rep1%7B%5C%7Dtype=pdf>.
- [73] A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel, “Learning from the Hindsight Plan – Episodic MPC Improvement”, *ArXiv e-prints*, Sep. 2016. arXiv: [1609.09001](https://arxiv.org/abs/1609.09001) [cs.R0].
- [74] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, “Deep reinforcement learning for high precision assembly tasks”, in *IROS*, 2017, pp. 819–825.
- [75] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning”, in *International Conference on Learning Representations (ICLR)*, 2016, ISBN: 0-7803-3213-X. DOI: [10.1613/jair.301](https://arxiv.org/pdf/1509.02971.pdf). arXiv: [9605103](https://arxiv.org/pdf/1509.02971.pdf) [cs]. [Online]. Available: <https://arxiv.org/pdf/1509.02971.pdf>.
- [76] H. Van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-learning”, in *AAAI*, 2016, ISBN: 1509.06461v3. arXiv: [1509.06461v3](http://www.aaai.org). [Online]. Available: www.aaai.org.

- [77] V. Pong, S. Gu, M. Dalal, and S. Levine, “Temporal Difference Models: Model-Free Deep RL For Model-Based Control”, in *International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: <https://arxiv.org/pdf/1802.09081.pdf>.
- [78] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for model-based control”, in *International Conference on Intelligent Robots and Systems (IROS)*, 2012.
- [79] J. Falco, Y. Sun, and M. Roa, “Robotic grasping and manipulation competition: Competitor feedback and lessons learned”, in *Robotic Grasping and Manipulation*, Y. Sun and J. Falco, Eds., Cham: Springer International Publishing, 2018, pp. 180–189.
- [80] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, “Hindsight Experience Replay”, in *NIPS*, 2017. arXiv: [1707.01495](https://arxiv.org/abs/1707.01495). [Online]. Available: <https://arxiv.org/pdf/1707.01495.pdf%20http://arxiv.org/abs/1707.01495>.
- [81] R. Cheng, A. Verma, G. Orosz, S. Chaudhuri, Y. Yue, and J. Burdick, “Control regularization for reduced variance reinforcement learning”, in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, Long Beach, California, USA: PMLR, Sep. 2019, pp. 1141–1150. [Online]. Available: <http://proceedings.mlr.press/v97/cheng19a.html>.
- [82] F. Wang and J. Decker, “Demystifying Differentiable Programming: Shift/Reset the Penultimate Backpropagator”, *arXiv preprint arXiv:1803.10228*, 2018. arXiv: [arXiv:1803.10228v1](https://arxiv.org/abs/1803.10228v1). [Online]. Available: <http://colah.github.io/posts/2015-09-NN-Types-FP/>.
- [83] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke, “The Limits and Potentials of Deep Learning for Robotics”, *International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018, ISSN: 1756-8277. DOI: [10.1177/ToBeAssigned](https://doi.org/10.1177/ToBeAssigned). eprint: [1804.06557](https://arxiv.org/abs/1804.06557). [Online]. Available: <http://arxiv.org/abs/1804.06557>.
- [84] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, “A survey on policy search for robotics”, *Foundations and Trends in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.
- [85] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey”, *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.
- [86] O. Khatib, “A unified approach for motion and force control of robot manipulators: The operational space formulation”, *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, Feb. 1987, ISSN: 0882-4967.
- [87] J. Peters and S. Schaal, “Learning to control in operational space”, *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 197–212, 2008. DOI: [10.1177/0278364907087548](https://doi.org/10.1177/0278364907087548).

- [88] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, “Operational space control: A theoretical and empirical comparison”, *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 737–757, 2008. DOI: [10.1177/0278364908091463](https://doi.org/10.1177/0278364908091463).
- [89] J. Buchli, F. Stulp, E. Theodorou, and S. Schaal, “Learning variable impedance control”, *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 820–833, 2011. DOI: [10.1177/0278364911402527](https://doi.org/10.1177/0278364911402527).
- [90] Kuka, *Kuka lbr iiwa*. [Online]. Available: <https://www.kuka.com/en-us/products/robotics-systems/industrial-robots/lbr-iiwa>.
- [91] Rethink Robotics, *Sawyer user guide*. [Online]. Available: http://mfg.rethinkrobotics.com/mfg-mediawiki-1.22.2/images/1/1a/Sawyer_User_Guide_3.3.pdf.
- [92] Willow Garage, *Pr2 robot manual*. [Online]. Available: https://www.clearpathrobotics.com/wp-content/uploads/2014/08/pr2_manual_r321.pdf.
- [93] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning”, in *International Conference on Machine Learning*, 2016, pp. 1928–1937.
- [94] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, “An application of reinforcement learning to aerobatic helicopter flight”, in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, ser. NIPS’06, Canada: MIT Press, 2006, pp. 1–8. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2976456.2976457>.
- [95] S. Levine and P. Abbeel, “Learning neural network policies with guided policy search under unknown dynamics”, in *Advances in Neural Information Processing Systems (NIPS)*, 2014.
- [96] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, “Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search”, in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, IEEE, 2016, pp. 528–535.
- [97] S. Levine, N. Wagener, and P. Abbeel, “Learning contact-rich manipulation skills with guided policy search”, in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, IEEE, 2015.
- [98] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine, “Path integral guided policy search”, in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 3381–3388.
- [99] J. Fu, S. Levine, and P. Abbeel, “One-shot learning of manipulation skills with online dynamics adaptation and neural network priors”, in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, IEEE, 2016, pp. 4019–4026.
- [100] V. Kumar, E. Todorov, and S. Levine, “Optimal control with learned local models: Application to dexterous manipulation”, in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, IEEE, 2016, pp. 378–383.

- [101] P. Englert and M. Toussaint, “Learning manipulation skills from a single demonstration”, *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 137–154, 2018.
- [102] I. Lenz, R. A. Knepper, and A. Saxena, “Deepmpc: Learning deep latent features for model predictive control”, in *Robotics: Science and Systems*, 2015.
- [103] S. Levine and V. Koltun, “Guided policy search”, in *Proceedings of The 30th International Conference on Machine Learning*, 2013.
- [104] W. H. Montgomery and S. Levine, “Guided policy search via approximate mirror descent”, in *Advances in Neural Information Processing Systems*, 2016, pp. 4008–4016.
- [105] S. S. M. Salehian and A. Billard, “A dynamical-system-based approach for controlling robotic manipulators during noncontact/contact transitions”, *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2738–2745, Oct. 2018, ISSN: 2377-3766.
- [106] S. M. Khansari-Zadeh and A. Billard, “Learning stable nonlinear dynamical systems with gaussian mixture models”, *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, Oct. 2011, ISSN: 1552-3098.
- [107] I. Mordatch, E. Todorov, and Z. Popović, “Discovery of complex behaviors through contact-invariant optimization”, *ACM Trans. Graph.*, vol. 31, no. 4, 43:1–43:8, Jul. 2012, ISSN: 0730-0301.
- [108] I. Mordatch, K. Lowrey, and E. Todorov, “Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids”, in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 5307–5314.
- [109] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. V. Todorov, “Interactive control of diverse complex characters with neural networks”, in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., Curran Associates, Inc., 2015, pp. 3132–3140.
- [110] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, “Deep reinforcement learning for high precision assembly tasks”, in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, IEEE, 2017, pp. 819–825.
- [111] B. Siciliano and L. Villani, *Robot force control*. Springer Science & Business Media, 2012, vol. 540.
- [112] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*, 1st. New York, NY, USA: Cambridge University Press, 2017, ISBN: 1107156300, 9781107156302.
- [113] B. D. Ziebart, J. A. Bagnell, and A. K. Dey, “Modeling interaction via the principle of maximum causal entropy”, in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML’10, Haifa, Israel: Omnipress, 2010, pp. 1255–1262, ISBN: 978-1-60558-907-7.

- [114] E. Todorov and W. Li, “A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems”, in *Proceedings of the 2005, American Control Conference, 2005.*, Jun. 2005, 300–306 vol. 1.
- [115] J. Bohg, A. Morales, T. Asfour, and D. Kragic, “Data-driven grasp synthesis—a survey”, *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, 2013.
- [116] A. Bicchi and V. Kumar, “Robotic grasping and contact: A review”, in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, IEEE, vol. 1, 2000, pp. 348–353.
- [117] A. Bicchi, “On the closure properties of robotic grasping”, *The International Journal of Robotics Research*, vol. 14, no. 4, pp. 319–334, 1995.
- [118] A. T. Miller and P. K. Allen, “Grasplit! a versatile simulator for robotic grasping”, *IEEE Robotics Automation Magazine*, vol. 11, no. 4, pp. 110–122, 2004.
- [119] C. Goldfeder and P. K. Allen, “Data-driven grasping”, *Autonomous Robots*, vol. 31, no. 1, pp. 1–20, 2011.
- [120] C. Hernandez, M. Bharatheesha, W. Ko, H. Gaiser, J. Tan, K. van Deurzen, M. de Vries, B. Van Mil, J. van Egmond, R. Burger, *et al.*, “Team delft’s robot winner of the amazon picking challenge 2016”, in *Robot World Cup*, Springer, 2016, pp. 613–624.
- [121] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, “Cloud-based robot grasping with the google object recognition engine”, in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 4263–4270.
- [122] M. Ciocarlie, K. Hsiao, E. G. Jones, S. Chitta, R. B. Rusu, and I. A. Şucan, “Towards reliable grasping and manipulation in household environments”, in *Experimental Robotics*, Springer, 2014, pp. 241–252.
- [123] A. Herzog, P. Pastor, M. Kalakrishnan, L. Righetti, J. Bohg, T. Asfour, and S. Schaal, “Learning of grasp selection based on shape-templates”, *Autonomous Robots*, vol. 36, no. 1-2, pp. 51–65, 2014.
- [124] A. Nguyen, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, “Detecting object affordances with convolutional neural networks”, in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 2765–2770.
- [125] D. Kappler, J. Bohg, and S. Schaal, “Leveraging big data for grasp planning”, in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 4304–4311.
- [126] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, “Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics”, 2017.

- [127] J. Varley, J. Weisz, J. Weiss, and P. Allen, “Generating multi-fingered robotic grasps via deep learning”, in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 4415–4420.
- [128] M. Veres, M. Moussa, and G. W. Taylor, “Modeling grasp motor imagery through deep conditional generative models”, *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 757–764, 2017.
- [129] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, “Scalable deep reinforcement learning for vision-based robotic manipulation”, in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 87, 2018, pp. 651–673.
- [130] R. Rubinstein, “The cross-entropy method for combinatorial and continuous optimization”, *Methodology and computing in applied probability*, vol. 1, no. 2, pp. 127–190, 1999.
- [131] J. Tobin, W. Zaremba, and P. Abbeel, “Domain randomization and generative models for robotic grasping”, *arXiv preprint arXiv:1710.06425*, 2017.
- [132] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, “Sim2real viewpoint invariant visual servoing by recurrent control”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4691–4699.
- [133] S. James, A. J. Davison, and E. Johns, “Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task”, *arXiv preprint arXiv:1707.02267*, 2017.
- [134] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, *et al.*, “Using simulation and domain adaptation to improve efficiency of deep robotic grasping”, in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 4243–4250.
- [135] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, “Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 627–12 637.
- [136] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, N. Fazeli, F. Alet, N. C. Daffe, R. Holladay, I. Morona, P. Q. Nair, D. Green, I. Taylor, W. Liu, T. Funkhouser, and A. Rodriguez, “Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching”, in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2018.
- [137] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning”, *GitHub repository*, 2016.

- [138] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, “Industry 4.0”, *Business & Information Systems Engineering*, vol. 6, no. 4, pp. 239–242, 2014.
- [139] M. Ye, X. Wang, R. Yang, L. Ren, and M. Pollefeys, “Accurate 3d pose estimation from a single depth image”, in *Computer Vision (ICCV), 2011 IEEE International Conference on*, IEEE, 2011, pp. 731–738.
- [140] C. Choi, Y. Taguchi, O. Tuzel, M.-Y. Liu, and S. Ramalingam, “Voting-based pose estimation for robotic assembly using a 3d sensor.”, in *ICRA, Citeseer*, 2012, pp. 1724–1731.
- [141] E. Olson, “Apriltag: A robust and flexible visual fiducial system”, in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 3400–3407.
- [142] B. Tekin, S. N. Sinha, and P. Fua, “Real-time seamless single shot 6d object pose prediction”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 292–301.
- [143] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world”, in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, IEEE, 2017, pp. 23–30.
- [144] F. Sadeghi and S. Levine, “Cad2rl: Real single-image flight without a single real image”, *arXiv preprint arXiv:1611.04201*, 2016.
- [145] D. F. Dementhon and L. S. Davis, “Model-based object pose in 25 lines of code”, *International journal of computer vision*, vol. 15, no. 1-2, pp. 123–141, 1995.
- [146] I. Skrypnyk and D. G. Lowe, “Scene modelling, recognition and tracking with invariant image features”, in *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, IEEE Computer Society, 2004, pp. 110–119.
- [147] A. Collet, M. Martinez, and S. S. Srinivasa, “The moped framework: Object recognition and pose estimation for manipulation”, *The International Journal of Robotics Research*, vol. 30, no. 10, pp. 1284–1306, 2011. DOI: [10.1177/0278364911401765](https://doi.org/10.1177/0278364911401765).
- [148] A. I. Comport, E. Marchand, M. Pressigout, and F. Chaumette, “Real-time markerless tracking for augmented reality: The virtual visual servoing framework”, *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 615–628, Jul. 2006.
- [149] C. Harris, *Tracking with Rigid Objects*. MIT Press, 1992.
- [150] T. Drummond and R. Cipolla, “Real-time visual tracking of complex structures”, *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 932–946, 2002.

- [151] F. Jurie and M. Dhome, “Hyperplane approximation for template matching”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 996–1000, 2002.
- [152] V. Lepetit, P. Fua, *et al.*, “Monocular model-based 3d tracking of rigid objects: A survey”, *Foundations and Trends® in Computer Graphics and Vision*, vol. 1, no. 1, pp. 1–89, 2005.
- [153] H. Kato and M. Billinghurst, “Marker tracking and hmd calibration for a video-based augmented reality conferencing system”, in *Augmented Reality, 1999.(IWAR’99) Proceedings. 2nd IEEE and ACM International Workshop on*, IEEE, 1999, pp. 85–94.
- [154] F. Bergamasco, A. Albarelli, E. Rodola, and A. Torsello, “Rune-tag: A high accuracy fiducial marker with strong occlusion resilience”, in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, IEEE, 2011, pp. 113–120.
- [155] R. Bajcsy, “Active perception”, 1988.
- [156] R. Bajcsy, Y. Aloimonos, and J. K. Tsotsos, “Revisiting active perception”, *Autonomous Robots*, vol. 42, no. 2, pp. 177–196, 2018.
- [157] O. Faugeras, *Three-dimensional computer vision: a geometric viewpoint*. MIT press, 1993.