Representation Learning for Tactile Manipulation

Repräsentationslernen für taktile Manipulation
Master-Thesis von Zhizhen Wang aus Jiangsu, China
Tag der Einreichung:

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dr. Riad Akrour
3. Gutachten: Filipe Veiga

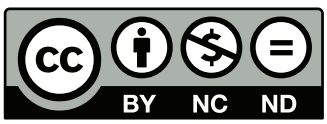
TECHNISCHE
UNIVERSITÄT
DARMSTADT

Representation Learning for Tactile Manipulation
Repräsentationslernen für taktile Manipulation

Vorgelegte Master-Thesis von Zhizhen Wang aus Jiangsu, China

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dr. Riad Akrour
3. Gutachten: Filipe Veiga

Tag der Einreichung:



Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 27. August 2018

(Zhizhen Wang)

Abstract

In robot manipulation tasks, tactile sensor is important in order to provide tactile feedback from grasped objects. The tactile information such as forces, mircovibrations and temperature is able to be detected, having similar sensory capability of human fingertips. While the complex, non-linear and high-dimensional tactile data is hard for control policy to learn, incorporating this feedback into a policy is challenging in more than one aspect.

First, tactile sensors rely on complex physical interactions that preclude the use of an accurate simulator, and hence model-based RL. Model-free RL has been successfully applied to robotics using low dimensional representation of movement primitives. However, these movement primitives are often executed in open-loop and cannot process tactile feedback. In recent years, there has been progress in learning complex policies from raw sensor data in a model-free way, but the sample complexity of such methods prevents them to run on a physical robot. For this purpose, we propose to learn the transferable representation of the tactile data, which can be utilized into the policy search in the simulator and further transfered on the physical robot.

Representation learning can reduce the dimensionality of the problem. However the learned latent representation by applying unsupervised learning is not able to transfer. As in the simulator we don't have the corresponding accurate tactile model to obtain the latent representation. We will propose supervised representation learning to infer variables of tactile representation distribution. As the tactile feedback often provides implied information about physical properties of grasped object, we learn the pose of manipulated object as representation from tactile data. The representation, namely the object pose, is available in the simulator and served as observation for learning the control policy. Afterwards we deploy the policy on the physical robot for evaluation. Meanwhile using the prediction of our supervised learning model by feeding in the tactile data, the representation can be successfully transfered on the real robot.

Acknowledgments

I would like to first thank my supervisor Dr. Riad Akrour, he provides me this interesting research topics. He is always patient and discusses with me at countless meeting. He encourages me to try new approaches, from which I learned a lot outside the lecture. I would also thank Filipe Veiga, who always provides help on configuring robot hand and gives comprehensive explanation on the robot. Meanwhile I'm grateful to Prof. Dr. Jan Peters, who provides the opportunity and recommends me to appropriate IAS group members to write the thesis according to my research interest.

I sincerely thank my parents, who gives me affectionate support as always, and motional inspire when I'm depressed. Also express my gratitude to Lan Yu for her love and care.

Finally I appreciate the discussion with Jonas Sperling and Yi Zhen, which give me helpful suggestion.

Contents

Figures and Tables

List of Figures

List of Tables

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q Network
GAE	Generalized Advantage Estimation
i.i.d.	independently and identically distributed
KL	Kullback-Leibler divergence
LSTM	Long Short Term Memory
MDP	Markov Decision Process
NAF	Normalized Advantage Function algorithm
PCA	Principal Components Analysis
PPO	Proximal Policy Optimization
RNN	Recurrent Neural Network
TRPO	Long Short Term Memory

List of Symbols

Notation	Description
$\mathcal{N}$	Normal distribution
$\mathbf{A}$	Advantage function
η	Expected accumulated reward

$\mathbf{H}$	Hessian matrix
π_θ	Parametric policy
$\mathbf{Q}$	State-action Q function
$\dot{\mathbf{q}}$	Joint velocity
τ	Trajectory
$\boldsymbol{\theta}$	Vector of parameters from a probability distribution
$\mathbf{V}$	Value function

List of Operators

Notation	Description	Operator
∇_θ	Derivative with respect to parameter θ	$\nabla_\theta(\bullet)$
E	Expected value of a random variable	$\mathrm{E}[\bullet]$
$\int$	Integral	$\int \bullet$
ln	The natural logarithm	$\ln(\bullet)$
$\prod$	Product	$\prod \bullet$

1 Introduction

Since the robot systems assist human to accomplish more and more challenging task and some also equipped with anthropomorphic hands, having the ability of in-hand manipulation is crucial to achieve the real-world tasks. In order to perform object or tool manipulation task in unstructured environments, it is important for robot to make use of tactile information. In our work, we will learn the representation of the tactile feedback, which is one of the observations in the reinforcement learning phase and with which the policy can be successfully transfered from the simulator to the real world.

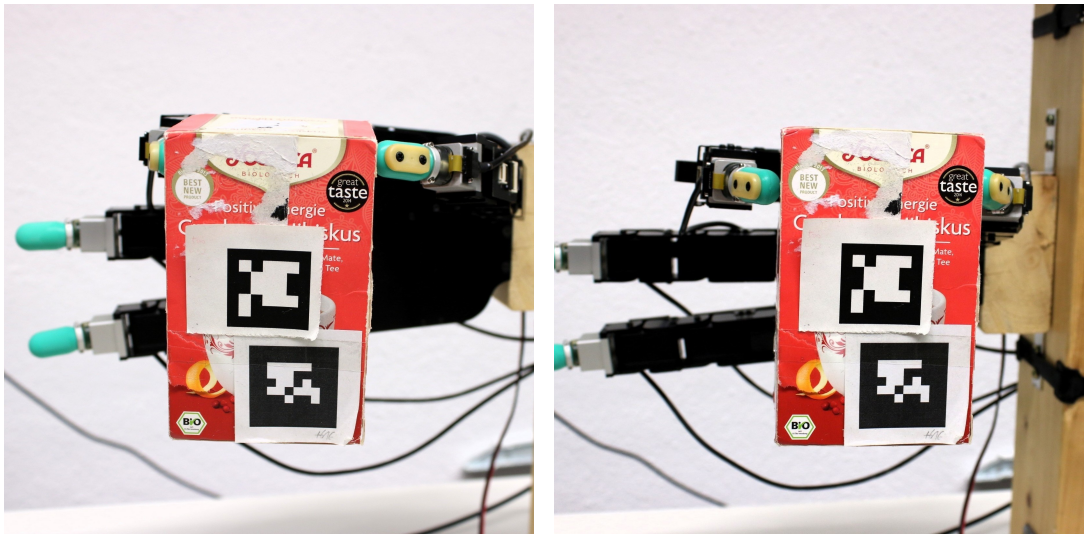


Figure 1.1: We learn the object pose, which tracked using Aruco marker, as representation mapping from BioTac tactile feedback. Then use this representation as the observation to learn object translation task by applying deep reinforcement learning in the simulator. At last, we deploy the policy on the real robot with this transferable representation.

We introduce the status quo of the in-hand manipulation with tactile feedback, includes the work others have done and the issue still need to be solved. In addition we proposed our method to solve it.

When the visual feedback is not available during manipulation, or due to partial occlusion by other hand, then the tactile feedback can provide implied information about physical properties of grasped object. In addition, tactile sensing can provide robustness and adaptation to variation in object properties [7], also it can detect slippage and instability[4]. With the help of tactile information, irreversible event is less likely to happen. Thus tactile feedback is necessary for us to provide more stable grasping performance.
On the other side, high-dimensional and non-linear tactile sensory pose a major challenge and is hard to train the learning algorithm on the real robot adopting raw data directly. Furthermore, it's not possible to build a accurate model of as delicate tactile sensory in the simulator as in the real world. Researches such as [8, 3] proposed using Variational Autoencoder [9] to learn latent variables of probabilistic generative models of the tactile feedback to solve the problem. Such latent representation can be used when the control policy trained on the real robot.
However we want the control policy that based on the representation be applied on the real robot, while training it in the simulator to save the trial and error time on the real physical environment. Thus we proposed the supervised learning to capture the object physical properties from tactile information. Unlike unsupervised learning, the proposed method will learn directly important features of object manipulation task, e.g. object pose, object material or internal physical state, and is able to transfer such representation from the simulator to the real world. The object properties we used here is object pose, which can be easily obtain via marker tracking algorithm. The supervised learning method we used here to learn the tactile representation is recurrent neural network with memory, namely LSTM [10], whose output depends on historical inputs, and it's able to avoid gradient vanish problem occurred on the traditional recurrent neural network [11].

We also need to solve the robot hand manipulation problem in order to evaluate the representation of tactile information we learned. Reinforcement learning has recently achieved significant progress. [12] shows the RL agent has the ability to master Atari Game with only video input. Bots trained using massively-scaled version of Proximal Policy Optimization also win against the world's top professionals at Dota 2, which has long time horizons, a partially-observed state and a high-dimensional continuous action space. Compared to such complex game decision making, in-hand manipulation task also needs to face difficulties such as dynamic partially-observed state and high-dimensional continuous action space. [13, 14, 15] deals with manipulation task with model-based RL method, which optimizing the policy by estimating the dynamics models. They are either 7 degrees of freedom robot (PR2), or using human demonstration or assuming simple linear model from data, which infers such method is not suitable for our case. In contrast, model-free RL methods do not require a dynamic model when optimizing the policy.

In this work, we focus on actor-critic model-free reinforcement learning, which includes policy search methods and function approximation methods. Both approaches have recently been combined with deep neural networks for learning in-hand manipulation task. We apply here concretely TRPO and PPO reinforcement method to train the policy based on the representation of tactile information, and learn to translate object to the target position. And we further deploy the policy to the real robot to evaluate the performance of object pose estimator.

1.1 Related work

In the next paragraph, we discuss related work on in-hand manipulation with tactile sensing, as well as reinforcement learning method used for different kinds of object manipulation task.

Tactile Sensing for robot object manipulation

Although using the image data as robot perception pay more attention in the literature [15, 16, 17, 18], human manipulation control and reaction rely heavily on the tactile sensory information [19], so does the robot manipulation. [20, 21] used the sensor feedback to learn a predictive model of task outcome. It allows much safer interaction with the environment, which share the same workspace with human, if the robot can predict the failure before it happens. In our case, we terminate the interaction if robot looses contact on the fingertips, and utilize tactile feedback mainly on robot trajectories generation.

[1] uses tactile feedback for in-hand object localization and object manipulation task. They learned a manipulation action by dynamic motor primitives incorporated with tactile feedback. And reducing the number of tactile feedback parameters by performing PCA on the tactile images. The result showed that the tactile feedback can significantly improves the movement execution in an altered task environment. However the tactile sensor they used is 8x8 dynamical matrix analog pressure sensors equipped on the robot gripper. It is coarse tactile sensor compared to Biotac sensor we used. [2] also solves the problem of localizing the pose and shape of an object by applying particle filtering. The object state is estimated during manipulation by integrating the likelihood of contact measurements over possible contact positions. But the sensor is on the Robonaut 2 hand, having a different setup from us. The same tactile sensor (BioTac [22, 23]) as we used here are [4, 8, 3]. [4] uses biomimetic tactile feedback for slip detection and contact force estimation. Since Biotac is a high dimensional tactile sensor, it may include strongly correlated or non-relevant dimensions. A better strategy is to use its compact low dimensional representation, like [8, 3]. They use Autoencoder to learn state representation for tactile and visual data. The result showed the reinforcement learning with learned state representation performs better in the presence of noise from raw data. The detailed setting shows in the Figure 1.2.

We consider also use the tactile information to assist our manipulation task, and learn its representation due to high dimensionality of tactile senor. However different from the above mentioned manipulation tasks which are trained either on the real-robot or in the simulator using visual information, we want to obtain the representation of the tactile feedback learned from the simulator, and which is further able to be transfered on the real robot. The reason is that first the space between the finger epidermis and rigid core is inflated with conductive liquid, such BioTac sensor is hard to model. Second, the reinforcement learning we used later is model-free policy search. It requires large number of trajectory samples. If they are generated on the real robot, it needs not only human supervision but also more time compared with working with simulated system. So if we can learn the policy based on the representation, which can be transfered between real world and simulated system, the exploration time can be largely shorten. Therefore we propose the supervised learning method to learn the representation of biomimetic tactile feedback. As tactile information provides additional cues about the object state, such as its position and orientation, we try to learn the explicit mapping from tactile information to object pose.

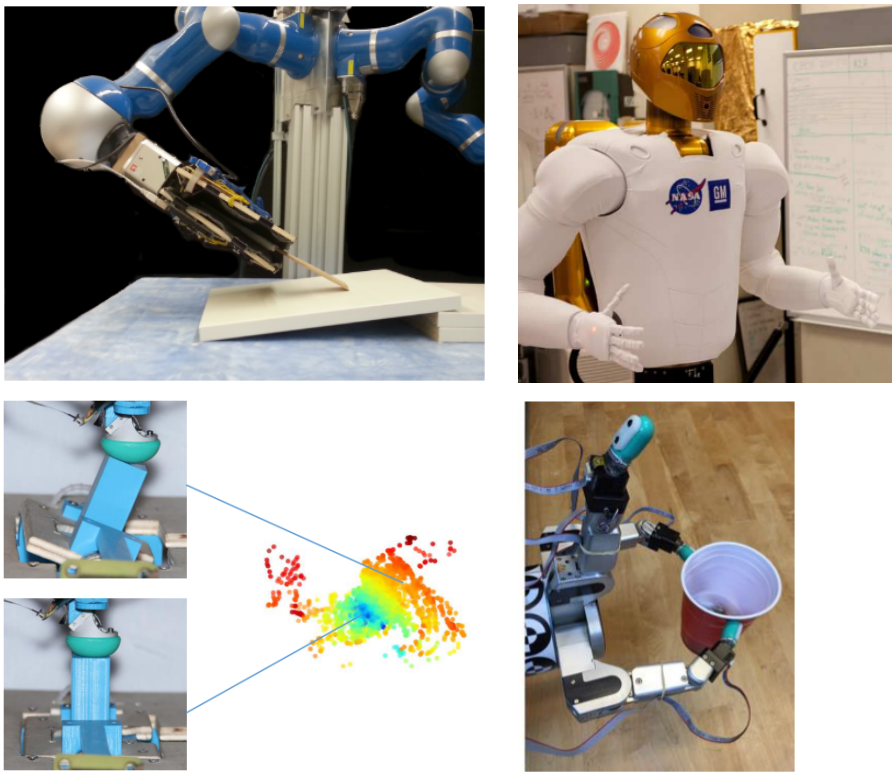


Figure 1.2: Different setup of the object manipulation using tactile sensing. Top left is [1], which uses two dynamical matrix analog pressure sensors binded on the gripper to perform a scraping task. Top right is [2], which applying particle filtering to estimate the pose of an object that is captured by the Robonaut 2 hand. Bottom left is [3], using a 5-DoF robot with 228-dimension sensor data to learn to manipulate using a learned three-dimensional feature space. Different contact states, shown on the left, yield different latent-space values. Bottom right is [4], showing robotic arm grasping a fragile object using the proposed force grip controller.

Reinforcement learning for robot object manipulation

In prior robot learning work, reinforcement learning (RL) has been used to explore both model-based and model-free learning algorithm. Model-based algorithm has the method estimating a variety of dynamics models such as local linear model [13], hidden Markov model [14], dynamic motor primitives [24] or guided policy search based on the trajectories [15] and so forth. While the model-based policy search has the potential to require fewer interactions with the robot and to efficiently generalize to unforeseen situations, inaccurate model can lead to control strategies that are not robust to model error since learned policy is based on internal simulation with learned models [25]. It's also hard to transfer to real-world manipulation, since learning complex models on real world systems with significant contact dynamics is difficult.

In contrast, model-free RL methods do not require a dynamics model when optimizing the policy. In addition with human demonstration imitation learning results in policies that exhibit natural and robust trajectories. Using actor-critic model-free RL method deep deterministic policy gradient (DDPG) incorporating human demonstrations, [5] shows success on various dexterous manipulation skills on the simulator such as object relocation, in-hand manipulation, tool use, and opening doors. Although we can do imitation learning for our manipulation task, the learned policy will highly dependent on the trajectory we demonstrated. What we want is to let policy dependent on the representation from the tactile feedback, therefore imitation learning is not appropriate in our task. [6] used the model-free method learning without any prior knowledge. They presented asynchronous variant of Normalized Advantage Function algorithm (NAF) for door-opening task. The experiment showed the model-free method can learn stable non-linear policy to accomplish complex robotic manipulation tasks. The detailed setting of both experiment shows in Figure 1.3.

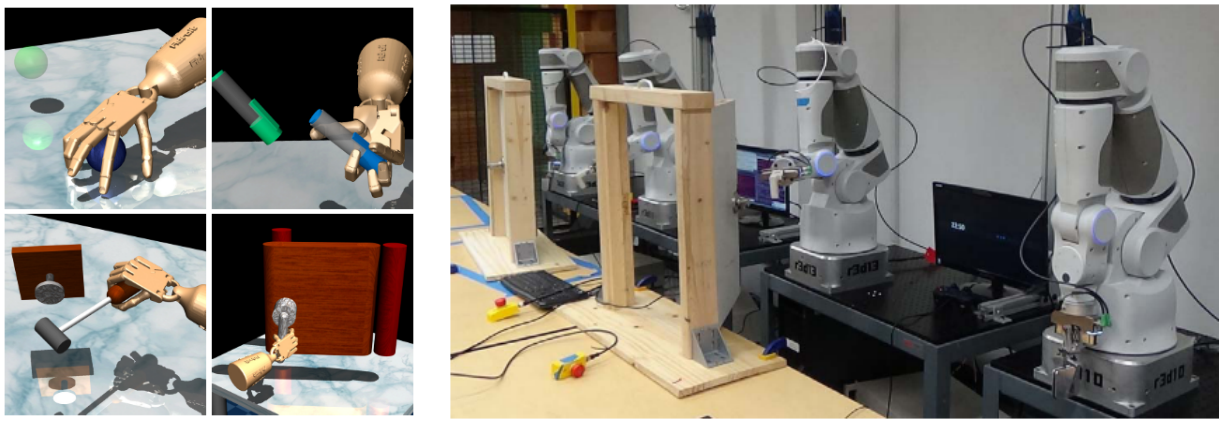


Figure 1.3: Here shows using RL to learn manipulation tasks. Left one is [5], which uses imitation learning to learn dexterous manipulation skills such as object relocation, in-hand manipulation, tool use, and opening doors. Right one is [6], which demonstrates two robots learning door opening task by asynchronous deep reinforcement learning algorithm.

We will then also use model-free method to learn in-hand manipulation skills from scratch by using Proximal Policy Optimization Algorithms (PPO) [26] and Trust Region Policy Optimization (TRPO) [27], since PPO and TRPO outperforms other methods like Vanilla PG, A2C and Cross-Entropy Method with comparison on almost all the continuous control environments (Roboschool) [26].

The thesis is structured as follows. The second section introduces background information on LSTM, PPO and TRPO we used. Third section brings our approach to solve tactile manipulation task based on representation. Fourth section includes experiment setup. Fifth section is the result we experienced. Last section we conclude with a discussion and future works.

2 Background

In this chapter we will introduce the technical background of method we used. In order to learn the representation of the tactile information, namely the object pose, we use recurrent neural network with memory, that is Long Short Term Memory (LSTM). It's one kind of the recurrent neural network, that can in addition capture long distance dependencies between input data. The mathematical form and its advantage over traditional recurrent neural network will be given in first section.

In the second section, we will introduce reinforcement learning (RL). The method is used to learn our control policy to manipulate the object. We will first briefly introduce the reinforcement learning. It includes the fundamental mathematical form of RL learning problem, which is Markov decision process. And the objective of reinforcement learning. Last types of RL algorithms, which mainly includes policy gradient, value-based, actor-critic RL method. The RL method we used is PPO, which is actor-critic type. The actor-critic method estimates the Q-function or values function, and uses it to calculate the gradient of the policy, which separately denotes policy evaluation and policy update. Policy evaluation estimates the accumulated expected rewards for a particular state. Policy update improves the policy by calculating the gradient of the objective of RL learning problem.

2.1 Representation Learning

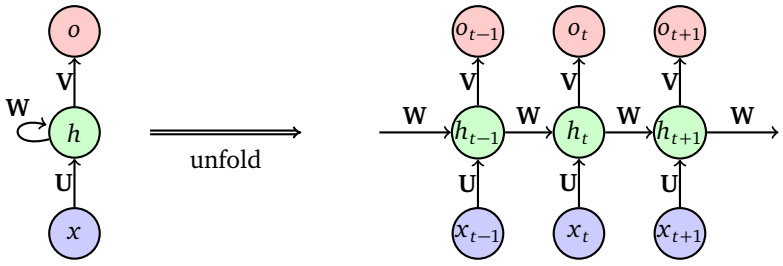


Figure 2.1: The recurrent network can be explained, if we unfold it in time sequence. It's a feed-forward neural network, with distinction that activation of the current hidden state is dependent on its last state concatenated with the input of current time step. Note here the weight matrix **U**,**V**,**W** is shared in the RNN.

To deal with time series data, recurrent neural network (RNN) could be considered. It's a class of neural network having recursive connection between nodes. This allows it to present dynamic behavior for a time sequence. The computational graph of RNN shows in Figure 2.1. We denote input vector x_t, hidden layer vector h_t, output vector o_t, parameter matrix $\mathbf{W}$, $\mathbf{U}$, $\mathbf{V}$ and activation function σ. RNN can be present in following mathematical form.

$$\begin{aligned} h_t &= \sigma_h(x_t\mathbf{U} + h_{t-1}\mathbf{W}) \\ o_t &= \sigma_o(h_t\mathbf{V}) \end{aligned} \tag{2.1}$$

However in the traditional recurrent neural network, the gradient signal can end up being multiplied a large number of times by the weight matrix during learning process. And it leads to the vanishing gradients or exploding gradients problem [11]. This issue could be solved by Long Short Term Memory [10](LSTM) (Figure 2.2).

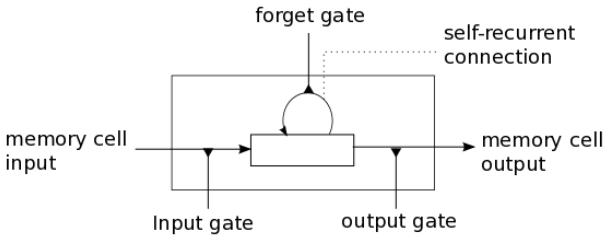
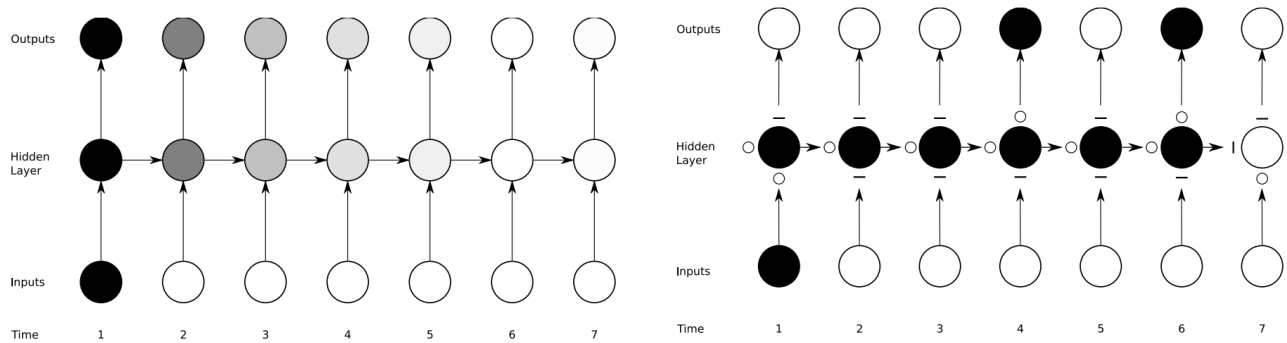


Figure 2.2: Illustration of an LSTM memory cell. The memory cell is composed of four elements: an input gate, a neuron with a self-recurrent connection, a forget gate and an output gate. The gates serve to modulate the interactions between the memory cell itself and its environments. The **input gate** can allow input signal to alter the state of memory cell or not. The **output gate**, on the other way, can allow the state of memory cell to influence the neurons on the next time step or not. Finally, the **forget gate** can allow the cell to remember or forget its previous state. [1]

The idea behind the LSTM model is to keep around memories to capture long distance dependencies using memory cell. We denote $F(x,h;\theta) = \sigma(x\mathbf{W} + h\mathbf{U})$ The memory cell is composed of four elements: an input gate $i_t = F(x_t, h_{t-1}, \theta_i)$, a neuron with a self-recurrent connection, a forget gate $f_t = F(x_t, h_{t-1}, \theta_f)$ and an output gate $o_t = F(x_t, h_{t-1}, \theta_o)$. In the beginning, we have new memory cell $\tilde{C}_t = \tanh(x_t\mathbf{W} + h_{t-1}\mathbf{U})$ depends on input and last hidden state as gates. The gate serve to modulate the interactions between the memory cell C_t itself and its environments. The input gate can allow input signal to alter the state of memory cell or not. The output gate, on the other way, can allow the state of memory cell to influence the neurons on the next time step or not. Finally, the forget gate can allow the cell to remember or forget its previous state. In the mathematical form it's

$$\begin{aligned} C_t &= f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \\ h_t &= o_t \cdot \tanh(C_t) \end{aligned} \tag{2.2}$$

Figure 2.3 from [28] illustrate the gradient information propagate in the RNN and LSTM. The shading of the nodes indicates their sensitivity to the input unit at time one. In LSTM the state of the input, forget, and output gate states are displayed below, to the left and above the hidden layer node, which corresponds to a single memory cell. For simplicity, the gates are either entirely open ('O') or closed ('—'). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed, and the sensitivity of the output layer can be switched on and off by the output gate without affecting the cell. Since LSTM could long distance dependencies along time series, we use here LSTM model to learn the representation. We assign the manipulated object pose as learning representation that could be transfered from the simulator to the real robot experiment.



(a) Vanishing gradient problem for RNNs

(b) Preservation of gradient information by LSTM

Figure 2.3: Comparison gradient information of LSTM with RNN. The shading of the nodes indicates their sensitivity to the input unit at time one. The sensitivity in RNN decays exponentially over time as new inputs overwrite the activation of hidden unit and the network 'forgets' the first input. However LSTM can preserve of gradient information by owning the gate.

[1] http://deeplearning.net/tutorial/lstm.html

2.2 Reinforcement Learning

We introduce first briefly the idea of reinforcement learning. It includes the definition of Markov decision process, it's fundamental mathematical form of Reinforcement Learning (RL), then provide the objective we need to optimize in RL, at last is the overview of RL algorithm type. Second we discuss policy evaluation, which estimates the Q function, value function or advantage given specific state and action and policy update method, which updates the policy to improve the objective.

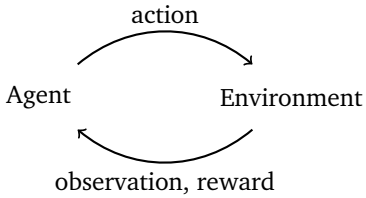


Figure 2.4: Agent-Environment Loop

The agent-environment interaction loop in Figure 2.4 shows how the agent to learn the policy by interacting with the environment. At each time step, the agent chooses an action according to its current observation or the state when it's fully observed, and the environment returns an observation or state and a reward base on the action the agent choose. The agent is aimed to optimize the policy, so that the accumulated expected rewards it gets is maximum.

Markov decision process $M = \{S, A, T, r\}$, which is mathematical form of reinforcement learning problem, see Figure 2.5. S denotes state space, the state s in the set can be discrete or continuous. A denotes action space. T is transition operator, it's the probability of entering the new state given current state and action, that is $p(s_{t+1}|s_t, a_t)$. r is the reward function, mapping from the set of state, action space to real number. At each time step t, the process is in state s_t. The decision maker can take any action a_t in A (action space), and receives the reward $r(s_t, a_t)$. MDP satisfies the Markov property, that means the prediction of future state s_{t+1} only depends on the current state action pair (s_t, a_t), independent of its future or past state action pair.

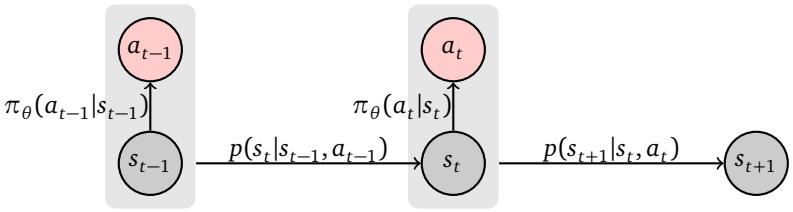


Figure 2.5: The Graph of Markov decision process, s denotes the state, a denotes the action. The state depends on the previous state and action given the probability $p(s_{t+1}|s_t, a_t)$ and the policy $\pi_\theta(a_t|s_t)$ relates the state to the action.

We form the policy as $\pi_\theta(a|s)$, and the environment has the transition operator $p(s'|s, a)$. Below is the probability over trajectories, when agent experiences in the environment:

$$\pi_\theta(\tau) = p_\theta(s_1, a_1, s_2, a_2, \ldots) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{2.3}$$

The objective of reinforcement learning is trying to optimize the policy, so that it maximizes the expected accumulated trajectory reward $J(\theta) = E_{\tau\sim\pi_\theta}[r(\tau)]$, where the reward is given by $r(\tau) : \sum_{t=1}^{T} r(s_t, a_t)$. When infinite horizon case is considered, the reward $r(\tau) = \sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t)$.

Next we introduce several types of RL algorithms, mainly includes policy gradient, value-based, actor-critic, model-based RL method.
Policy gradient method directly calculates the gradient of observed accumulated reward, that is $\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$. Methods include REINFORCE, Trust region policy optimization.

Model-based RL method will estimate the transition, and learn a model of system dynamics. Afterwards the model generates the trajectory for policy learning. In contrast, model-free algorithm learn the policy based on the trajectory sampled from the robot. They're often easier than learning a accurate forward model. However real robot interaction is time consuming and would cause wear and tear in robots. Methods include Dyna, Guided policy search.
Value-based method will estimate the Q-function $Q(s_t, a_t) = E_{\tau\sim\pi(\theta)}[r(\tau)|s_0 = s, a_0 = a]$ or value function $V(s_t) = E_{\tau\sim\pi(\theta)}[r(\tau)|s_0 = s]$ of the optimal policy. And acts by maximize these function. However value function approximation will a very difficult problem in high-dimensional state and action space. And often only used in discrete space. Methods include Q-learning, DQN, Temporal difference learning.
Actor-critic method estimates the Q-function or values function, and use it to calculate the gradient of the policy. Methods are Asynchronous advantage actor critic (A3C), Deep Deterministic Policy Gradient(DDPG), Proximal Policy Optimization(PPO).

2.2.1 Policy Evaluation

We talk here how to evaluate the policy using neural network. The value function is the expected reward given particular state $V^\pi(s_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t]$. We can approximate the value function by neural network. The true target value is $y_t \approx r(s_t, a_t) + \gamma \hat{V}_\phi^\pi(s_{t+1})$, ϕ is neural network parameter. The estimated value is $\hat{V}_\phi^\pi(s_t)$. Then we can use supervised learning method to minimize the loss between them.
Combined with policy gradient, which both approximated by neural network, is called ***actor-critic*** method. The algorithm looks like so: first sample $\{s_i, a_i\}$ from $\pi_\theta(a|s)$ by running the policy, then fit $\hat{V}_\phi^\pi(s)$ we discussed above using supervised learning, meanwhile calculate the policy gradient $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i)\hat{A}^\pi(s_i, a_i)$, finally use the gradient to update the policy $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$. The whole process can be iterated until the policy converged.
One of the estimators computing variance-reduced advantage-function making use a learned value function $V(s)$ is generalized advantage estimation (GAE). The generalized advantage estimation [29] is designed to reduce the variance of policy gradient estimate. The k-step advantage estimator $\hat{A}(k)$ formed below, is involves a k-step estimate of the returns, minus a baseline term $-V(s_t)$.

$$\begin{aligned} &\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V = -V(s_t) + r_t + \gamma r_{t+1} + \ldots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) \\ &\text{where } \delta_t^V = -V(s_t) + r_t + \gamma V(s_{t+1}) \end{aligned} \tag{2.4}$$

And the generalized advantage estimator GAE(γ, λ) is defined as the exponentially-weighted average of $\hat{A}(k)$:

$$\begin{aligned} \hat{A}_t^{\text{GAE}(\gamma,\lambda)} &= (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \ldots\right) \\ &= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \end{aligned} \tag{2.5}$$

By setting $\gamma = 0$ and $\gamma = 1$, there are two special case of the formula:

$$\begin{aligned} &\text{GAE}(\gamma, 0): && \hat{A}_t = \delta_t && = r_t + \gamma V(s_{t+1}) - V(s_t) \\ &\text{GAE}(\gamma, 1): && \hat{A}_t = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} && = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V(s_t) \end{aligned}$$

GAE$(\gamma, 1)$ use the sum of true reward, does not include bias but has high variance. GAE$(\gamma, 0)$ use an approximate value function, which induces bias but has much lower variance. The advantage estimator controlled by two parameter γ and λ makes a compromise between bias and variance.

2.2.2 Policy Update

We have talked about the objective of reinforcement learning, and the expected accumulated rewards $J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau) r(\tau) d\tau$. We assume having the random variable x and its function, knowing the identity

$$\nabla \log f(x) = \frac{\nabla f(x)}{f(x)} \implies \nabla f(x) = f(x) \nabla \log f(x) \tag{2.6}$$

Then using identity (2.6)

$$\begin{aligned}\nabla_\theta J(\theta) &= \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau \\ &= \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau \\ &= E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]\end{aligned} \tag{2.7}$$

Remember that the trajectory distribution is given by (2.3), and we take log of both sides, we got $\log \pi_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)$. Insert the form into Equation 3.3, we got

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t=1}^{T} r(s_t, a_t)\right)\right] \tag{2.8}$$

The way evaluate the expectation from (2.8) is by approximating it using Monte-Carlo method, which repeats random sampling on the dynamic system to obtain numerical results. Other method to evaluate the policy is using neural network to approximate the value function. We will discuss it in section 2.2.1. Monte-Carlo method are unbiased, however, they typically exhibit a high variance. We talk about how to reduce the variance of policy gradient later. So we could approximate the equation (2.8):

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)\left(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t})\right) \tag{2.9}$$

This part $\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)$ is actually the gradient of maximum likelihood method in supervised learning. The REINFORCE [30] algorithm introduced by Williams in 1991 is the first policy gradient method. It first samples the trajectories by running the policy, then uses (2.9) to compute the gradient, at last updates the parameter of the policy by gradient accent $\theta' = \theta + \alpha \nabla_\theta J(\theta)$. And repeats the whole process until the policy converged. The intuition here is to increase the probability of trajectory returns higher reward, and decrease the probability of trajectory returns lower reward.
In order to reduce the variance of the gradient, we introduce here two tricks. First, we know the policy at time t cannot affect the reward at time t', if $t' < t$, so

$$E_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^{T}\left(\nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=1}^{t} r(s_{t'}, a_{t'})\right)\right] = 0 \tag{2.10}$$

We can then reform the equation (2.9) like this:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})\right) \tag{2.11}$$

Because we sum up fewer numbers, the overall variance decreases. Some denotes $\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$ as $\hat{Q}_{i,t}$. If we use true expected accumulated rewards instead without sampling, this part should be $Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t, a_t]$. Rewrite the gradient from (2.8), we got

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^\pi(s_t, a_t)\right] \tag{2.12}$$

Another trick is called baselines. That is subtract the trajectory reward with some constant,

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\big[\nabla_\theta \log \pi_\theta(\tau)\big(r(\tau) - b\big)\big] \tag{2.13}$$

And the policy gradient estimate remains unbiased. The average value of $Q^\pi(s_t, a_t)$ is $V^\pi(s_t)$, as the definition $V^\pi(s_t) = E_{a_t\sim\pi_\theta(a_t|s_t)}[Q^\pi(s_t, a_t)]$. So value function is a suitable baseline. And $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is called advantage, denote as $A^\pi(s_t, a_t)$. So $\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\big[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^\pi(s_t, a_t)\big]$.

Note that the policy gradient is on-policy method. It's inefficient, and we need to draw new trajectories when each time the policy changes. This could be changed to off-policy method using importance sampling. Importance sampling is the technique to estimate the property of a distribution, when having samples from another different distribution.

$$\begin{aligned} E_{x\sim p(x)}[f(x)] &= \int p(x)f(x)dx \\ &= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\ &= E_{x\sim q(x)}[\frac{p(x)}{q(x)}f(x)] \end{aligned} \tag{2.14}$$

So we only have samples from old policy $\pi_{\theta'}(\tau)$, the objective could be estimated as $J(\theta) = E_{\tau\sim\pi(\theta')}\left[\frac{\pi_\theta(\tau)}{\pi_{\theta'}(\tau)}r(\tau)\right]$. Combine with (2.3), the form can be expended as:

$$\begin{aligned} J(\theta) &= E_{\tau\sim\pi(\theta')}\left[\frac{p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)}{p(s_1)\prod_{t=1}^{T}\pi_{\theta'}(a_t|s_t)p(s_{t+1}|s_t,a_t)}r(\tau)\right] \\ &= E_{\tau\sim\pi(\theta')}\left[\prod_{t=1}^{T}\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)}r(\tau)\right] \end{aligned} \tag{2.15}$$

The importance sampling weights will explode or vanish, when t goes to infinity.

Trust Region Policy Optimization (TRPO)

Except the issue we talked above in section 2.2.2 when we use importance sampling, another issue of the policy gradient method is that, it updates the policy in the parameter space. However small changes in the policy parameter may unexpectedly lead to big changes in the policy. So updating the policy according to parameter will lead to excessively large policy update. We need to bound the policy performance to solve the issue.
Consider now an infinite-horizon discounted Markov decision process (MDP), $\eta(\pi) = E_{\tau\sim\pi(\theta)}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)\right]$, where $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi(a_t|s_t)$, $s_{t+1} \sim p(s_{t+1}|s_t,a_t)$. In [31] derived an useful identity to express the expected return of other policy $\tilde{\pi}$ in terms of advantage over π.

$$\eta(\tilde{\pi}) = \eta(\pi) + E_{\tau\sim\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t A^\pi(s_t,a_t)\right] \tag{2.16}$$

Let $\rho_\pi(s) = \sum_{t=0}^{\infty}\gamma^t p(s_t = s|\pi)$ be the unnormalized discounted visitation frequencies. The Equation 2.16 can be rewritten without timesteps:

$$\begin{aligned} \eta(\tilde{\pi}) &= \eta(\pi) + \sum_{t=0}^{\infty}\sum_{s}p(s_t = s|\tilde{\pi})\sum_{a}\tilde{\pi}(a|s)\gamma^t A^\pi(s,a) \\ &= \eta(\pi) + \sum_{s}\sum_{t=0}^{\infty}\gamma^t p(s_t = s|\tilde{\pi})\sum_{a}\tilde{\pi}(a|s)A^\pi(s,a) \\ &= \eta(\pi) + \sum_{s}\rho_{\tilde{\pi}}(s)\sum_{a}\tilde{\pi}(a|s)A^\pi(s,a) \end{aligned} \tag{2.17}$$

This equation implies that if at each state s, the expected advantage is nonnegative when update from π to $\tilde{\pi}$, then it's guaranteed to increase the policy performance η, or keep constant if expected advantage is zero everywhere. Another thing the equation brings is, it solves the importance sampling weights exploding or vanishing problem when uses importance sampling, due to independence of time right now. However using the $\rho_{\tilde{\pi}}(s)$ on $\tilde{\pi}$ is hard to optimize directly. A local approximation form is below to replace $\rho_{\tilde{\pi}}(s)$ with $\rho_\pi(s)$, denoted as $L_\pi(\tilde{\pi})$.

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_{s}\rho_\pi(s)\sum_{a}\tilde{\pi}(a|s)A^\pi(s,a) \tag{2.18}$$

And the good news is L_π matches η to first order [31], if the policy is parameterized.

TRPO [27] proposed the policy update scheme uses KL constraint, also called trust region constraint. So it can be converted to optimization problem:

$$\begin{aligned} &\max_{\theta} L_{\theta_{\text{old}}}(\theta) \\ &\text{subject to } D_{\text{KL}}^{\theta_{\text{old}}}(\pi_{\theta_{\text{old}}}, \pi_\theta) \le \delta \\ &\text{where } L_{\theta_{\text{old}}}(\theta) = E_{(s,a)\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A_{\theta_{\text{old}}}(s,a)\right] \end{aligned} \tag{2.19}$$

And this optimization problem is solved by conjugate gradient algorithm followed by line search. The search direction is computed by approximately solving the equation $Hx = g$, where H is the Fisher information matrix, here equals to Hessian matrix of KL divergence. The quadratic approximation to the KL divergence constraint: $\bar{D}_{\text{KL}}(\theta_{\text{old}}, \theta) \approx \frac{1}{2}(\theta - \theta_{\text{old}})^T H(\theta - \theta_{\text{old}})$, $H = \nabla_\theta^2 \bar{D}_{\text{KL}}(\theta_{\text{old}} \| \theta)$ and approximate the objective $L_{\theta_{\text{old}}}(\theta) \approx L_{\theta_{\text{old}}}(\theta_{\text{old}}) + g^T(\theta - \theta_{\text{old}})$, $g = \nabla_\theta L_{\theta_{\text{old}}}(\theta)$, resulting the following problem:

$$\begin{aligned} &\max_{\theta} g^T(\theta - \theta_{\text{old}}) \\ &\text{s.t. } \frac{1}{2}(\theta_{\text{old}} - \theta)^T H(\theta_{\text{old}} - \theta) \le \delta \end{aligned} \tag{2.20}$$

After using conjugate gradient algorithm to approximate the search direction $s \approx H^{-1}g$, the maximal step length β is computed such that $\theta + \beta s$ will satisfy the KL constraint. Therefore $\delta = \frac{1}{2}(\beta s)^T H(\beta s) \to \beta = \sqrt{2\delta/(s^T H s)}$. Last use the line search to ensure the improvement of surrogate objective and satisfy the KL constraint. So start with maximal value of β, if not meet the condition that $\bar{D}_{\text{KL}}(\theta_{\text{old}}, \theta) \le \delta$ and $L_{\theta_{\text{old}}}(\theta)$ improves, then shrink the β exponentially until the condition meets. Other tricks used in practice such as Monte-Carlo method and importance sampling are omitted here, if interested, could see [27]. TRPO showed that optimizes a local approximation to the expected return of the policy with a KL divergence constraint achieves good empirical results on a range of challenging policy learning tasks.

Proximal Policy Optimization (PPO)

Unlike TRPO uses KL constraint to keep new policy near old policy, PPO uses a simpler way.
Let $r_t(\theta)$ denote the probabilty ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. The main object they propose is:

$$L^{\text{CLIP}}(\theta) = E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]\right] \tag{2.21}$$

By clipping the probability ratio, makes the new policy not far from the old one. Finally, taking the minimum of the clipped and unclipped objective leads the final objective to a pessimistic bound. The PPO algorithm first collect the trajectories on the old policy. And estimatess the advantages using any advantage estimation algorithm. Then optimize the surrogate objective 2.21 by gradient ascent with K epochs. And loop the whole process. These methods have the stability and reliability of trust-region methods but are much simpler to implement [26], requiring only few lines of code modifying a vanilla policy gradient code, and have better overall performance.
Another approach is to use adaptive KL penalty instead of clipped surrogate objective. First define the target value of KL divergence δ we expect to achieve each policy update. It optimizes the KL-penalized objective

$$L^{\text{KLPEN}}(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t - \beta D_{\text{KL}}^{\theta_{\text{old}}}(\pi_{\theta_{\text{old}}}, \pi_\theta)\right] \tag{2.22}$$

Denote $d = \hat{E}_t[D_{\text{KL}}^{\theta_{\text{old}}}(\pi_{\theta_{\text{old}}}, \pi_\theta)]$, if $d < \delta/1.5$, $\beta \leftarrow \beta/2$, if $d > 1.5\delta$, $\beta \leftarrow 2\beta$. The updated β is used for the next policy update.
The PPO learning algorithm that uses fixed-length trajectory segments is shown below. Each iteration, each of N actors collect T timesteps of data. Then the algorithm constructs the surrogate loss on these NT timesteps of data, and optimize it with minibatch SGD for K epochs.

3 In-hand Manipulation with Tactile Representation

We will explain here the whole framework of in-hand manipulation with tactile representation. In robot hand manipulation task, tactile feedback is as important as visual feedback, especially when the visual information is occluded by the hand. The tactile feedback can provide robustness and adaptation to the environment and makes irreversible event like slipping or falling less likely to happen.

However on the one hand, learning such control policy based on the tactile feedback on the real robot is impossible. The model-free reinforcement learning needs millions of transitions, it is inefficient and needs human supervision when training directly on the real robot. It will also cause wear and tear in robots. On the other hand, when modeling the interaction in the simulator, the accurate high dimensional, liquid inflated tactile sensor is hard to model. However the explicit representation of the tactile information is able to be transfered. Therefore, we train the policy in the simulator based on the object pose and transfer the policy on the real robot based on the tactile representation. Those dependency is identical and can be transfered.

We use supervised learning to learn the mapping from the tactile information to its representation, which is the pose of manipulated object. The non-linear model, which is neural network we trained, can predict the representation by feed in the tactile data. We also train the control policy in the simulator based on the object pose, which is provided by physical engine. In the end, we deploy the policy on the real robot. The policy observes the current joint state and the tactile representation, which is the prediction of trained neural network, and can accomplish in-hand manipulation task based on the representation of tactile feedback.

3.1 Supervised Learning for tactile representation

In order to apply supervised learning, we need the input-output, or feature-label pair. The learning function optimized on the training data by minimizing the loss between the true label and predicted label mapping from the features. And the optimal trained model is able to predict correctly on the unseen instances. In our case, the input is tactile data, output is object pose. More precisely, we predict the object pose based not only on the current tactile feedback but the last k tactile feedback. It leads to more robust prediction and steady learning process.

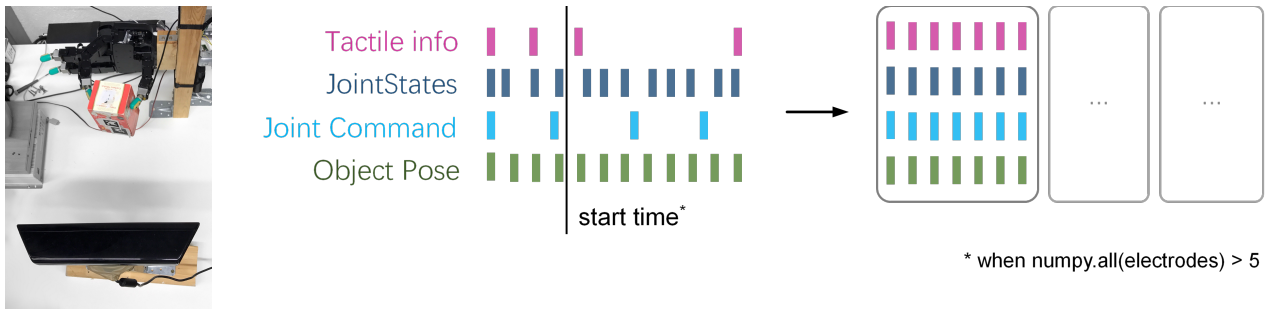
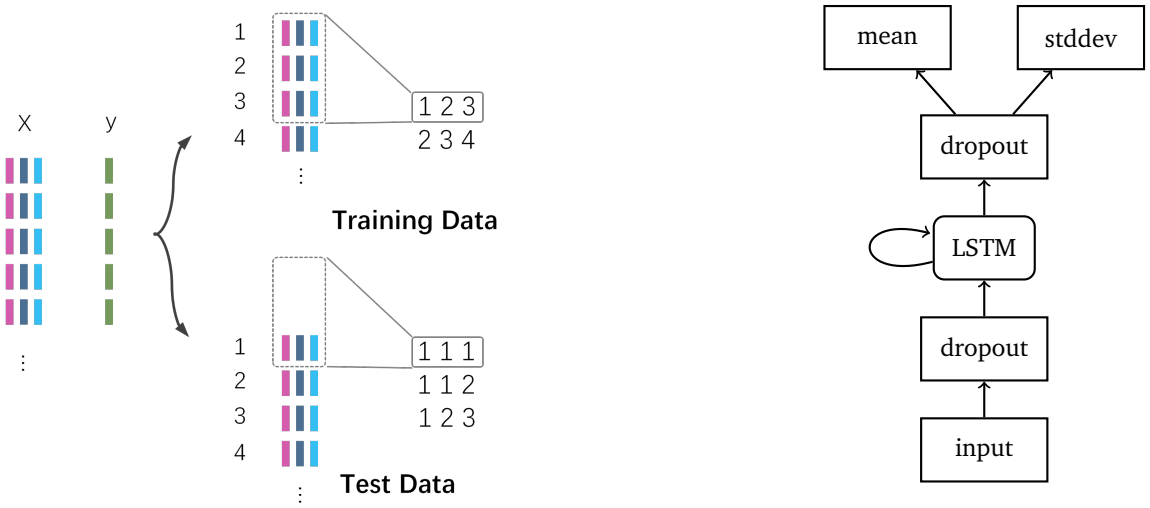


Figure 3.1: The left figure shows how we record the data. Using the glove to demonstrate the translation task and Aruco to track the object pose, we record BioTac electrodes, robot joint states, joint command from glove and tracked object pose. The right figure shows how we preprocess the data. The rectangle represents each trajectory file. We preprocessed the data such that they have aligned at the same time sequence and discard the data after the hand grasped the object.

We gather the data by demonstration the object translation task to learn the representation of tactile information. The robot hand is controlled by the glove, however due to the different construction of human hand and robot hand, we cannot easy rotate robot finger based on its root. What in-hand manipulation task we can demonstrate using the glove is the object translation task. The objective is obtain by augment reality marker tracking algorithm using additional camera mounted in front of object. Then we recorded all the data into rosbag, including tactile information, robot joint states,

joint command from glove and tracked object pose. Afterwards, preprocessing was made to ensure all the data is aligned and has frequency of 300 Hz. Also we chop the data starting from the point when all fingers were grasped on the object by checking the tactile values. The Figure 3.1 shows this process, and the rectangle is each trajectory rosbag file. We have here 76 dimensional tactile electrodes, 16 dimensional joint states of robot hand and 16 dimensional joint states command as features, also 7 dimensional object pose as objective.



(a) Generate data sequence

(b) Neural network structure

Figure 3.2: Representation Learning technical details. The left figure shows the process of generating the date sequence feed in the RNN. The right figure is the network structure for learning the representation.

The recurrent neural network is based on the time sequence data, in another word, the prediction is based on the historical observation. So we need to reform the data in such data sequence. We using sliding window method to obtain data based on time sequence, it can not only meet the input shape for RNN but also reuse the old observation data. So given the window size W, we create the data sequence shows in Figure 3.2a by shifting the window one by one. In this example, $W = 3$. In order to obtain true data sequence, the window only wrap within the data, it leads to the training data has lag of $W - 1$ time steps. For testing data, it requires to predict at the beginning, so we use padding to fill the repeated observation in the missing time steps. The initial window sequence is filled with data at first time step. After new data comes, the oldest data will pop up from the window, like the structure of queue.
After preparing the data sequence, we feed them into the recurrent neural network. Its structure shows in Figure 3.2b. It's simple LSTM with two dropout layers surrounding in order to prevent overfitting. And the network outputs the normal distribution of object pose. It is optimized by maximizing the log probability

$$J(\theta) = \sum_t \log \mathcal{N}(y_t; \mu_t, \sigma_t^2) \tag{3.1}$$

here y_t is the true object pose $(x, y, z, \alpha, \beta, \gamma)$. μ_t is the predicted mean value of the object pose distribution, σ_t is the standard deviation of the distribution. Note we use here euler angle to predict the orientation, because the quaternion (qx, qy, qz, qw) defined by axis-angle representation is

$$\begin{aligned} qx &= ax * sin(\text{angle}/2) \\ qy &= ay * sin(\text{angle}/2) \\ qz &= az * sin(\text{angle}/2) \\ qw &= cos(\text{angle}/2) \end{aligned} \tag{3.2}$$

the quaternion also satisfied $qw^2 + qx^2 + qy^2 + qz^2 = 1$, it's hard for the prediction whereas euler angle doesn't have such constraint, thus easier to be predicted.

3.2 Reinforcement Learning from learned representation for in-hand manipulation

Meanwhile, the representation, that is object pose, can be present in the simulator for the reinforcement learning. We train the object translation task using TRPO and PPO algorithm. The actor-critic method contains value network and policy network, they have the same observation. The policy network is two hidden layer with tanh activation and outputs the Gaussian distribution of continuous action command. The policy network is two hidden layer with relu activation and outputs the estimated values. And they don't share parameters with each other. At the beginning, the network has high entropy of actions by including an entropy bonus in the optimization objective to explore the action space as possible, and the entropy decreases with time.
But before applying the learning algorithm, we developed the manipulation environment based on the openai gym library. It's better for the further benchmarks and easy to apply RL baseline algorithm. We have already shown the agent environment interaction cycle in Figure 2.4. Here we defined the detailed observation in the simulator. The observation includes the position of 16 hand joint (rad), the position and orientation of current object, and the position and orientation of the desired target. Whereas in the real robot the observations are the position of 16 hand joint (rad), the predicted object pose mapped from the 76 Biotac electrodes and its target pose.

The exploration process is started by calling reset() function, which reset everything and return the initial observation. Reset process includes resetting the target pose, initial object pose and choosing number of fingers to manipulate the object. At the initial state the object is already held in the hand. The algorithm simply grip the object by starting all fingers horizontal towards the object, and inwards themselves until they touch the object. The step(action) function is called when the agent chooses an action, and will return an observation and reward. We will terminate the episode when the hand looses contact with the object or the manipulation duration is longer than 2 seconds.

We give here mathematical form, how the reward is calculated. The reward at each step t is calculated by

$$r_t = r_t^O + r_t^P + r_t^F + b \tag{3.3}$$

The object pose reward r_t^O penalizes the sum of exponential difference of distance and orientation. The norm in distance is 2-norm. It obtains the euclidean distance in 3D space. $\ominus$ is the operation to calculate the quaternion difference $q_1 \ominus q_2 = q_1^{-1} q_2$, the norm here is the angle from its axis-angle representation using Form 3.1.

$$r_t^O = \exp\big(-\|\hat{p}_t - p_t\|\big) + \exp\big(-\|\hat{o}_t \ominus o_t\|\big) \tag{3.4}$$

The perturbation reward r_t^P encourages smoother joint movement and lower joint velocity. w_p is the weight for perturbation reward. We use here $w_p = 0.001$.

$$r_t^P = -w_p \sum(\dot{q}^2) \tag{3.5}$$

The force reward penalize the difference between normal force applied on the object and desired force. The desired moderate force in the simulator is 140.

$$r_t^F = \sum \frac{|f_t - 140|}{50} \tag{3.6}$$

The bonus encourages the agent to move the object towards the target. Each time the object is within the thresholds range from the target, the agent gets extra reward. Also penalize when the agent loose contact with the object.

For randomization, the object initial position is able to be randomized. However changing the object position also effects the hand configuration, so we pseudo randomize the object position by any point in 7x7 grids. And store all the corresponding hand configuration in advance. The desired reaching target position is also randomized around the initial object position.

3.3 Representation Transfer from simulator to real robot

After the policy is learned, we have policy network and value network shown in the right part in Figure 3.3. The value network is used to estimate the expected rewards by given the observation. While applying the policy we don't need to update the policy, thus don't need the value network. However the policy network is needed to output the joint command based on the current observation. In the policy learning part, the observation of the policy is object pose, target pose and joint state. When we transfer the policy on the real robot, the object pose is replaced with the output of the trained representation learning model. It predicts the object pose by feed in the tactile information.

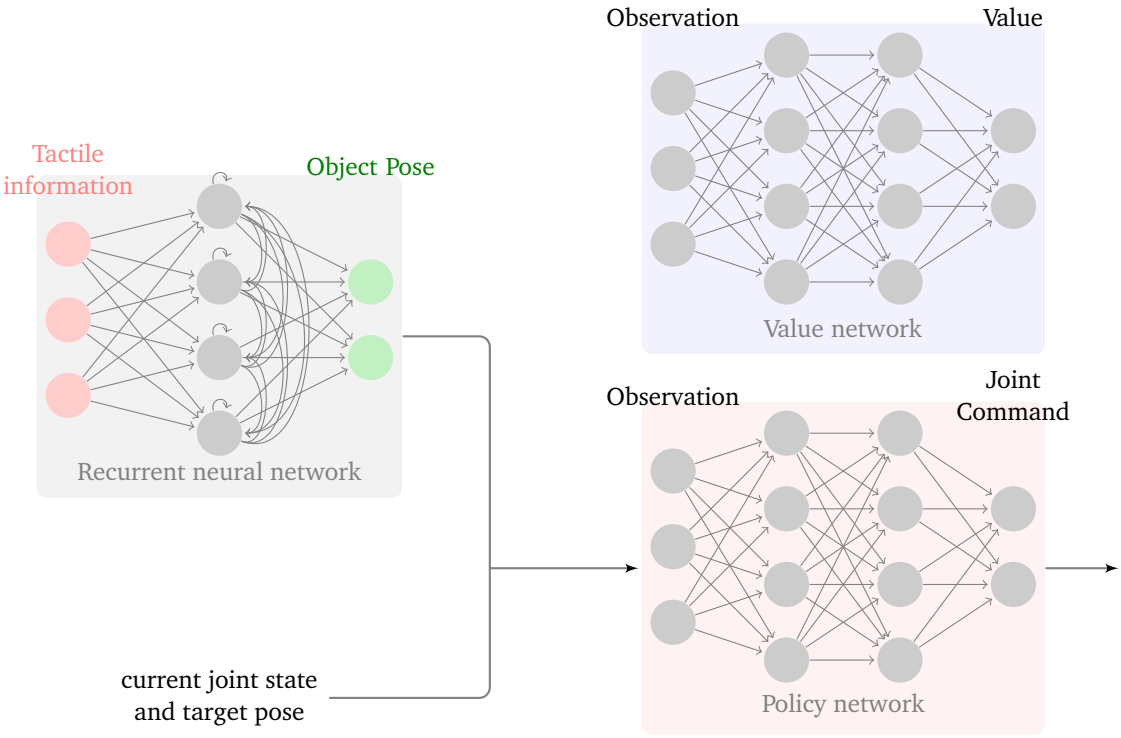


Figure 3.3: Transfer the representation. After learned the representation of the tactile feedback and trained the policy for object translation task, we just replace the object pose in the observation space with the representation. It's the prediction of the recurrent neural network by feed in the current tactile information.

The transfer process in Figure 3.3, shows how to transfer the representation of the tactile feedback to the real world object manipulation task. Instead of directly utilizing the high dimensional tactile information as observation, we just transfer its representation to the real robot, which is the pose of object in the hand. The representation is obtained using the trained recurrent neural network that can predict the object pose by feed in the past BioTac electrodes sequence.

Thanks transferable representation of the tactile data, we avoid to large training time on the physical robot, also avoid to model the physical interaction of tactile sensor. Meanwhile the policy is no need to deal with high dimensional noisy raw tactile data. Deploy the policy based on the representation of tactile feedback providing more stable observation.

4 Experiments

In this section, we describe the experiment setup, including hardware and software. We then state the experiment details. At last report the experiment results. The experiment includes two parts: pose prediction and reinforcement learning for simulated in-hand manipulation. We predict the tactile representation, that is object pose by using recurrent neural network with memory. The object pose is obtained by marker tracking algorithm. And we use PPO and TRPO to learn the robot in-hand manipulation task in the simulator. The policy we trained is based on the tactile representation and joint state, so that is able to transfer on the real robot.

4.1 Hardware Description

The hardware we use is mainly the allegro hand equipped with BioTac tactile sensor. The tactile sensor is used in the representation learning and representation transfer part, not involved in the reinforcement learning part. What else hardwares are the glove to control the allegro hand for representation learning and camera for tracking the AR marker attached on the object.

Allegro Hand

We use the Allegro Hand from WONIK ROBOTICS, it has lightweight and portable anthropomorphic design, can perform low-cost dexterous manipulation. It has 16 independent current-controlled joints, totally four finger, and each has four degrees of freedom. Each joint is actuated by DC motor. And we control the hand by position control. Also the fingertips can adapt different kinds of tactile sensors. It communicates via CAN at 333 Hz frequency.

BioTac

We equipped the allegro-hand with biometric tactile sensor BioTac [32]. Each BioTac consists of a rigid core housing with 19 electrodes surrounded by an elastic skin filled with conductive liquid 4.1. The curved, deformable nature of both the BioTac and biological fingertips provides mechanical features that are desirable for the manipulation of objects. The BioTac mimics the senory function of the human fingertip, and is able to detect the tactile information such as: forces, mircovibrations and temperature.

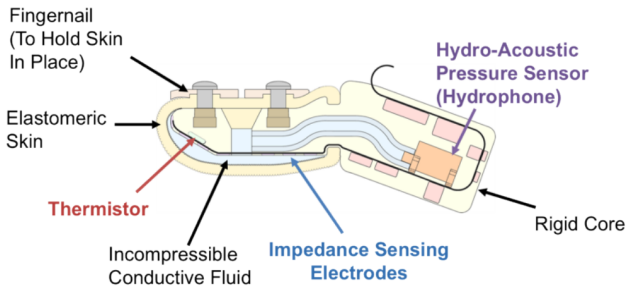


Figure 4.1: BioTac Structure. The BioTac is a unique tactile sensor capable of acquiring sensory modalities that mimic the full range of capabilities in the human fingertip. It consists of a rigid core that houses all of the sensory electronics and an elastomeric skin made of low-cost silicone. The space between the skin and core is inflated with a liquid giving the sensor a compliance very similar to the human fingertip.

The sensor provides different channels of information, includes 19 electrodes voltages (when pressing down over an electrode the measured voltage will decrease), DC pressure (increases linearly when the fluid pressure increases), AC pressure (allow for high resolution of vibrations), AC Temperature (decreases as the device is cooled), DC Temperature (decreases as the device warms up). We will purely use 19 electrode voltages for each finger, and learn the representation based on them.

Glove

Figure 4.2: The glove we used

The glove is self developed glove with additional photosensitive strip on each finger, which used to detect the extent of finger bending. The glove is used to control the allegro hand for demonstration of object manipulation. However the glove can only detect movement for each finger, but not each joint. And different from allegro hand, human finger can not rotate based on its root. So combine with algorithm, other joint values command has linear correlation with glove data. Thus such restriction allows us only to demonstrate translation task.

Cameras

We use Kinect for Xbox 360 to track the object during manipulation. However we don't use its depth sensor but only RGB camera for tracking marker. THe RGB video stream has 640 $\times$ 480 resolution at 30 Hz frame rate.

4.2 Software Description

Here we describe the software setup. They're mainly the simulator includes the V-REP and Pybullet providing the physical environment for reinforcement learning. Other software includes the Aruco algorithm to track the object pose and the ROS system to communicate between software and hardware.

Marker Tracking Algorithm

For object pose estimation we use Aruco [33] Augment Reality marker detector library, a popular library for detection of square fiducial markers. The main benefit of these markers is that a single marker provides enough correspondences (its four corners) to obtain the camera pose. Also, the inner binary codification makes them specially robust, allowing the possibility of applying error detection and correction techniques. We also did camera intrinsic and extrinsic calibration. The extrinsic calibration, or eye-on-base calibration, is based on hand-eye calibration [34]. It's able to compute the static transform from a robot base to the optical frame of a camera. We can then get directly the Cartesian coordinates of the object when publish all the transformation we need via ROS.

ROS

ROS is an open-source, meta-operating system for the robot. It provides the service including hardware abstraction, low-level device control, message-passing between processes and so on. We publish the robot joint command to control the robot. And use rosbag to record all the published topics we need during hand manipulation for supervised learning.

V-REP

The robot simulator V-REP is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, ROS nodes, BlueZero nodes, remote API clients, or a custom solution. This makes V-REP very versatile and ideal for multi-robot applications. It can simulate real-world physics and object interactions, also has full interaction with the object during simulation. We use this simulator to evaluate the precision of the representation prediction, since it has nice visualization comparing true object pose and predicted object pose.

Pybullet
PyBullet is an easy to use Python module for physics simulation for robotics, games, visual effects and machine learning. With PyBullet you can load articulated bodies from URDF, SDF, MJCF and other file formats. PyBullet provides forward dynamics simulation, inverse dynamics computation, forward and inverse kinematics, collision detection and ray intersection queries. After our comparison we found RL training time is more shorter on the Pybullet than V-REP. Thanks to the lightweight design of Pybullet, for 8 million transition Pybullet needs 30 minutes whereas V-REP needs 8 hours in our experiment.

4.3 Experiments

The experiments include the supervised learning for object pose prediction task and reinforcement learning in the simulator. Their details are discussed in the following section.

4.3.1 Pose Prediction

Before we collect the manipulation data, we did hand-eye calibration on the system. By attached the Aruco maker on the finger, we can get the transform of the fingertip by forward kinematics. Also we have the position of maker based on camera. By several sampling the hand-eye algorithm can compute the static transform from a robot's base to the optical frame of the camera. So we can get the object pose based on the robot base.
Then we gather the data by rosbag command recording demonstration the object translation task on the real robot. The experiment setup is shown in Figure 4.3. We totally has 10 trajectories, and after chopped into windows we have 55362 transitions (each includes 60 time steps). We reserve one trajectory for testing and one for evaluation. The LSTM network is feed in 76 dimensional BioTac electrodes and 16 dimensional joint position, outputs object pose including position and orientation. The network is optimized on the log probability of object pose by Adam optimizer. We also add position error and orientation error between predicted and true object pose. The network will stop training if the position error on the development data set is not improved for 5 epochs. The hyperparameter is shown in Table 4.1.

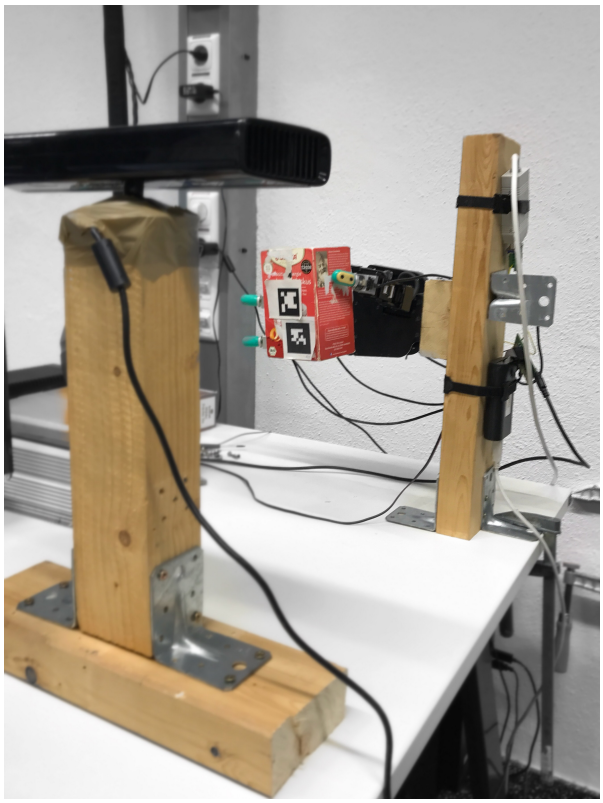


Figure 4.3: The setup for representation learning.

Hyperparameter of LSTM	Value
training sample numbers	48783
testing sample numbers	3509
hidden units	300
dropout rate	0.2
batch size	32
epochs	20
early stopping patience	5
optimizer	Adam
learning rate	1e-3
beta_1	0.9
beta_2	0.999
window size	60

Table 4.1: The table of hyperparameter of recurrent neural network

Afterwards we evaluate the learning performance by visualizing in the V-REP simulator, see in Figure 4.4. The red box represents the objective we recorded, and the white one is the box we predicted. The error between prediction and objective can be clearly seen through the simulator.

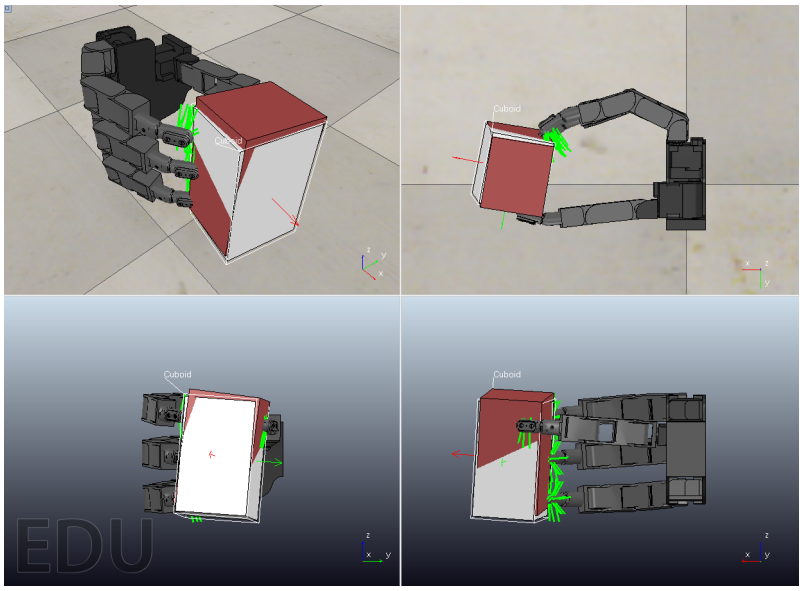


Figure 4.4: We evaluate the learning result in the V-REP, the prediction error can be straightforwardly seen in the simulator. The red box represents the objective we recorded, and the white one is the box we predicted.

We also quantify the position error and orientation error both on the training data and development data. Separately shown in Figure 4.5 and 4.6. On the training data the network can predict object position on the precision 0.005 meter, and object orientation above the precision 0.004 radiant. The learning curve is converged very fast. We tried also with and without joint position feature to check the learning performance. The plot shows that both data with and without joint position can converge fast, also the difference between them is tiny.

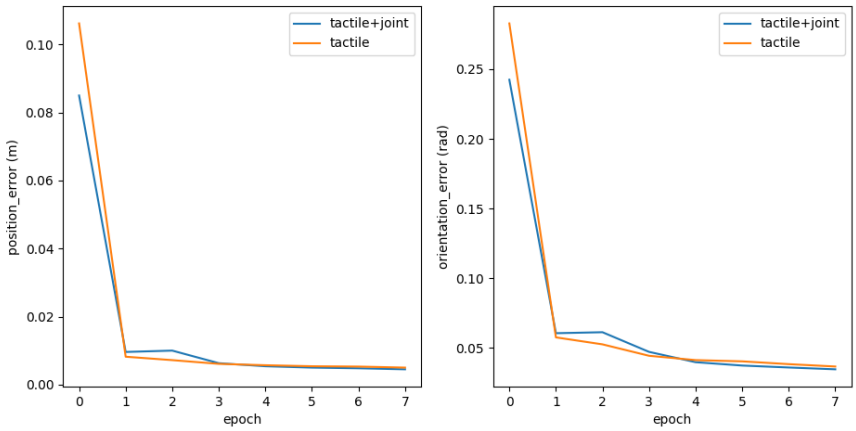


Figure 4.5: Comparing the learning curve of predicting the position and orientation on the training data. The orange line is the input only with tactile feedback. The blue one is the input with joint position and tactile feedback. The left plot shows the position error with unit meter. The right one shows the orientation error with unit radiant.

Figure 4.6 shows the performance on perdition of unseen data. The model with joint state feature performs better than without joint state overall, however the gap on the position error is not huge. At the 2 epoch, the position error reached at the minimum point nearly 0.005 meter, and stopped training after 5 no improving epochs, also the orientation error reached at the bottom at fourth epoch with 0.1. Interesting to see the gap of orientation error with and without joint state is not so big, maximum 0.02 radiant. It means if we predict the object orientation without joint state, the performance difference will not be obvious.

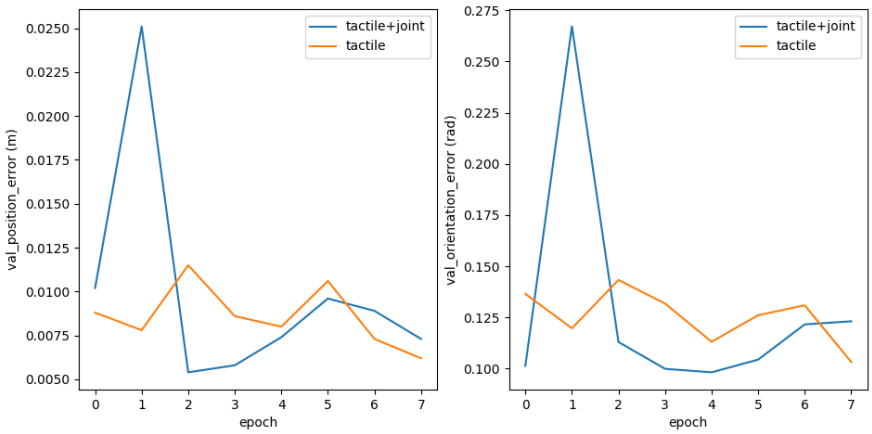


Figure 4.6: Comparing the learning curve of predicting the position and orientation on the development data. The orange line is the input only with tactile feedback. The blue one is the input with joint position and tactile feedback. The left plot shows the evaluation position error with unit meter. The right one shows the evaluation orientation error with unit radiant.

Due the our prediction model is the normal distribution, we not only obtain the mean of the prediction value but also the variance of the distribution. With a run on the different training data, the figure 4.7 shows the variance of the prediction distribution. By the way, the curve shows the learning phase also converge fast both on the position and orientation prediction, and keeps stable from fifth epoch. The standard deviation of position distribution is around 0.003. The standard deviation of orientation keeps around 0.02.

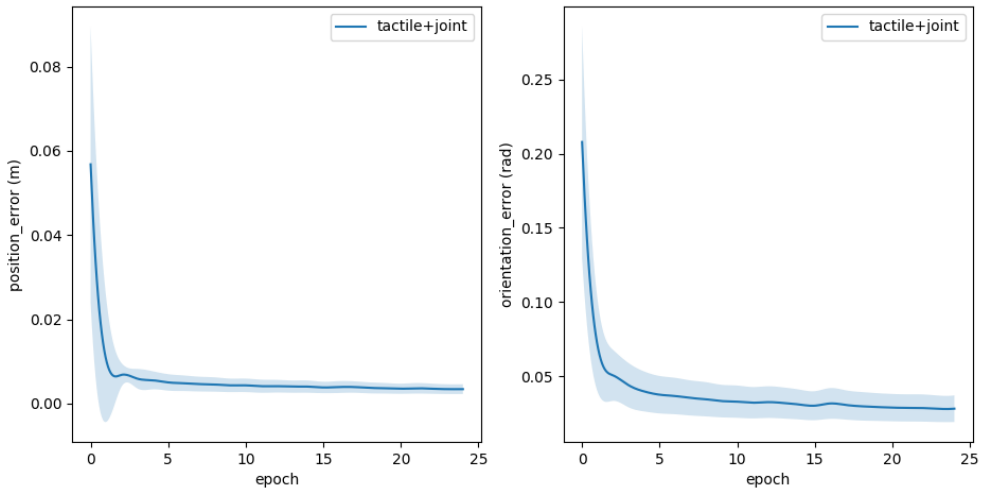


Figure 4.7: Comparing the learning curve of predicting the position and orientation with error on the training data. The left plot shows the evaluation position error with unit meter. The right one shows the evaluation orientation error with unit radiant. The shade denotes one standart deviation of the prediction distribution.

In the Figure 4.8 shows the variance of the prediction distribution on the development data. The learning curve on the development data is more twisted than on the training data, because they're unseen in the training process. The prediction on the position has 0.007 meter lowest error. And on the orientation prediction the error is nearly 0.12 radiant, equals 7 degrees. Both position and orientation error of development data are higher than on the training data in the anticipation. The standard deviation of position distribution is around 0.003. The standard deviation of orientation keeps around 0.02. Interesting to find out that the standard deviation of the prediction distribution on the development data doesn't go worse, which means the performance of our object pose prediction will be as steady as on the training phase.

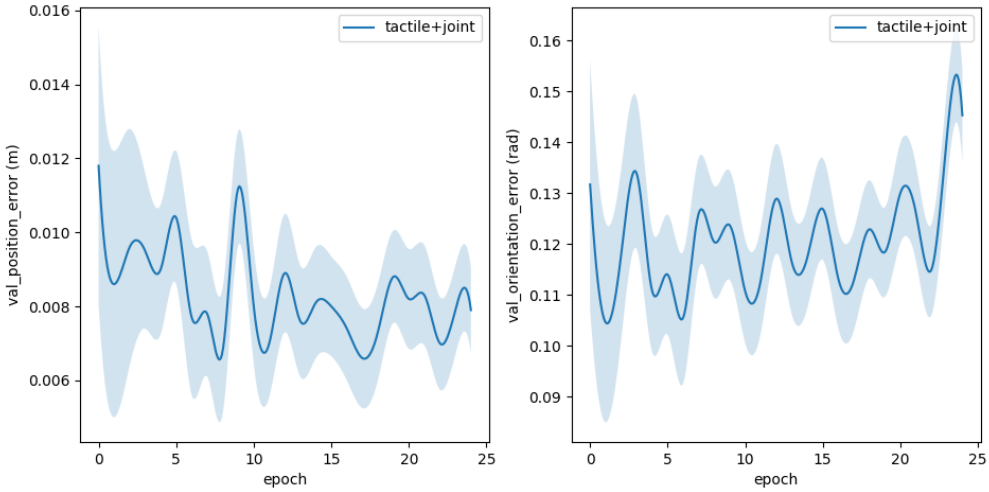


Figure 4.8: Comparing the learning curve of predicting the position and orientation with error on the development data. The left plot shows the evaluation position error with unit meter. The right one shows the evaluation orientation error with unit radiant. The shade denotes one standart deviation of the prediction distribution.

From the result we can conclude the representation of tactile data can be well predicted with lower bias and variance using LSTM model. Both position and orientation of predicted object accuracy can meet our requirements for transferring the representation.

4.3.2 RL for Simulated In-hand Manipulation Policy

We simulated the object manipulation task in the Pybullet shown in Figure 4.9. The white line draws the bounding box of target position. In order to make irreversible event less likely to happen at the learning start, we freeze the object and control the fingers inwards from the expanded state to the grasped state until all manipulating fingers touching the object. The simulation then starts with this grasped gesture preventing object from falling. Also every joint command is the action adding on the initial grasped joint position (absolute joint command). We also tried the joint command adding on the current joint position (relative joint command), but by comparison the absolute joint command performs better on the real robot.

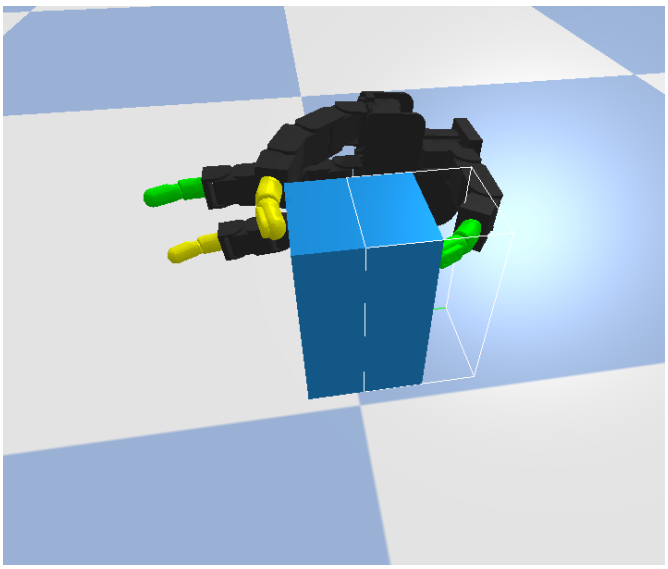


Figure 4.9: Here shows manipulating the object in the Pybullet with two fingers by performing reinforcement learning. The tea box is simpfied as the cube to accelerate the rendering speed. The white line draws the bounding box of the target position.

We also randomize the finger configuration at the start of each epoch and randomize the target position and initial object position to generate different manipulation task.

The table 4.2 and 4.3 shows the hyperparameter we used separately on the PPO and TRPO algorithm. The network structure is keeps same on both of them, that is MLP with two hidden layer with 64 neurons.

hyperparameter	**value**
manipulate object	cube
max manipulation time steps	600
iteration	500
minimum transition	1600
PPO clipping parameter	0.2
gradient norm clipping coefficient	0.5
discounting factor	0.99
advantage estimation discounting factor	0.95
optimizer	Adam
learning rate	5e-4

Table 4.2: The table of hyperparameter of PPO

hyperparameter	**value**
manipulate object	cube
max manipulation time steps	600
iteration	500
minimum transition	1600
max KL divergence	1e-3
conjugate gradient damping	1e-3
discounting factor	0.98
advantage estimation discounting factor	0.95

Table 4.3: The table of hyperparameter of TRPO

On the object manipulation task, we tried the task with fixed initial object pose and fixed target pose, task with random initial object pose and the task with random target pose. The learning curve of each manipulation task shows separately in Figure 4.10, 4.11 and 4.12. The result shows the for the simple manipulation task without randomization, the PPO algorithm performs better than TRPO algorithm. However when brings with the randomization, the both algorithm performs also worse. When randomize the initial object pose, PPO goes to flat early as same as TRPO on the first manipulation task. On the third task, when we randomize the target pose, TRPO performs better than PPO within the 1e6 timesteps.

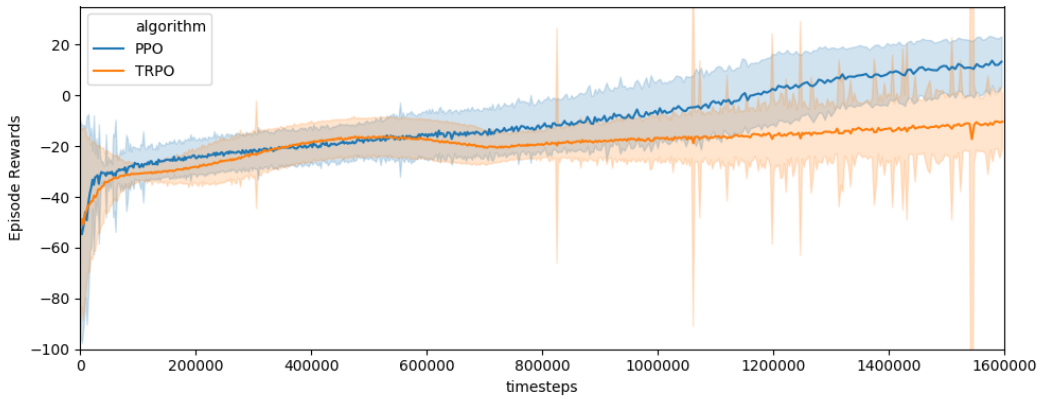


Figure 4.10: Here shows the comparison of the learning curve of PPO and TRPO on object manipulation task with fixed initial position and fixed target position. The line value is the average reward at one episode. The shade draws the standard deviation with the sliding window on the single seed. We see that PPO overperforms TRPO on this task within 1.6e6 timesteps.

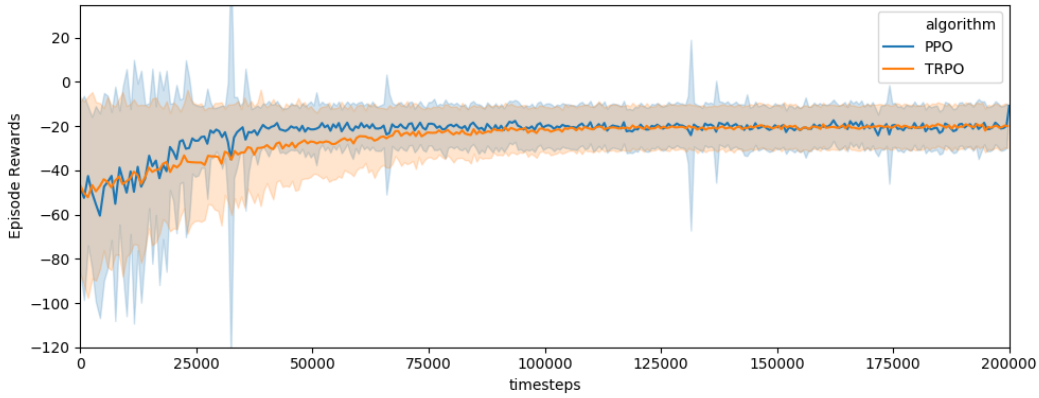


Figure 4.11: Here shows the comparison of the learning curve of PPO and TRPO on object manipulation task with random initial position and fixed target position. The line value is the average reward at one episode. The shade draws the standard deviation with the sliding window on the single seed. We see that the PPO also goes back to the average reward around -20 same as TRPO.

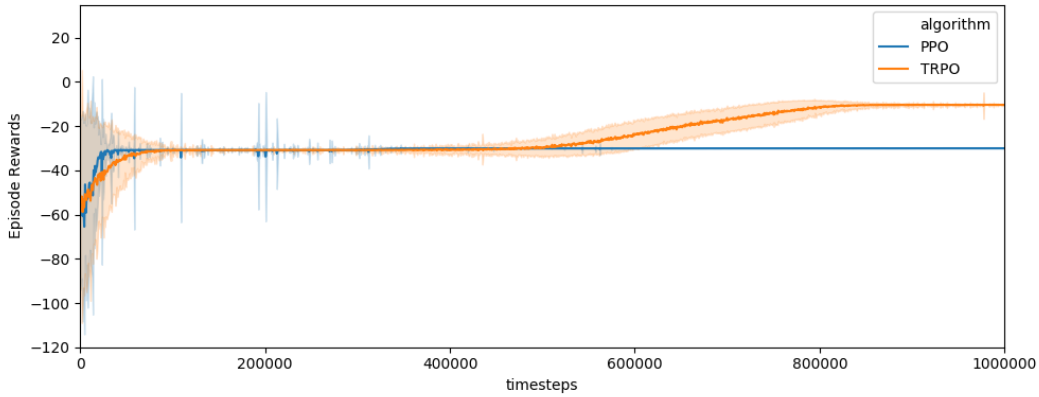


Figure 4.12: Here shows the comparison of the learning curve of PPO and TRPO on object manipulation task with fixed initial position and randomized target position. The line value is the average reward at one episode. The shade draws the standard deviation with the sliding window on the single seed. In this task the PPO is worse than TRPO with about 20 reward value.

With the learned policy on the simulator for in-hand object manipulation task, we combine its observation which is the prediction model of object pose. Then the policy can be successfully applied on the real robot based on the tactile representation instead of the high dimensional tactile data, without any additional learning phase on the real robot.

5 Discussion

In our work, we first study the literature on the related work about tactile sensing and reinforcement learning for object manipulation task. We then give the introduction about the technical background, that is supervised learning and reinforcement learning. Afterwards we demonstrated an approach to learning the robot in-hand manipulation task based on the tactile representation, which is also transferable on the real robot.

While the high-dimensional tactile data with complex physical properties is hard to be model, we need model-free reinforcement learning way to solve the manipulation task based on the tactile feedback. However the huge amount of transitions needs for RL training prevent us from interaction directly on the real robot. As the tactile feedback often provides implied information about physical properties of grasped object, the physical properties of object as representation from tactile data can be reasonable. They provide the possibility to transfer them from the simulator to the real robot, also decrease the dimensionality of the agent observation in the environment.

With the structure that the supervised learning model predicting the tactile representation and reinforcement learning policy that can deal with hand manipulation task based on the tactile representation, the policy can finally be smoothly applied on the real robot based on the trained supervised learning model providing the transferable tactile representation.

Future work will be using Leap motion controller to demonstrate the representation learning data so that the object manipulation task can be diverse. Also we can try to learn the control policy with different object shape and size to validate the robustness and adaptation of the tactile representation.

Bibliography

[1] Y. Chebotar, O. Kroemer, and J. Peters, "Learning robot tactile sensing for object manipulation," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, no. Iros, pp. 3368–3375, 2014.

[2] C. Corcoran and R. Platt, "A measurement model for tracking hand-object state during dexterous manipulation," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4302–4308, 2010.

[3] N. Chen and J. Peters, "Stable Reinforcement Learning with Autoencoders for Tactile and Visual Data," *IEEE International Conference on Intelligent Robots and Systems (IROS)*, no. August, 2016.

[4] K. Hausman, G. E. Loeb, Z. Su, K. Hausman, G. E. Loeb, G. S. Sukhatme, and S. Schaal, "Force Estimation and Slip Detection for Grip Control using a Biomimetic Tactile Sensor," no. October, pp. 297–303, 2015.

[5] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine, "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations," 2017.

[6] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3389–3396, IEEE, may 2017.

[7] T. Takahashi, T. Tsuboi, T. Kishida, Y. Kawanami, S. Shimizu, M. Iribe, T. Fukushima, and M. Fujita, "Adaptive grasping by multi fingered hand with tactile sensor based on robust force and position control," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 264–271, IEEE, 2008.

[8] M. Karl, J. Bayer, and P. van der Smagt, "Unsupervised preprocessing for Tactile Data," pp. 1–7, 2016.

[9] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov 1997.

[11] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, pp. 1310–1318, 2013.

[12] 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.

[13] V. Kumar, E. Todorov, and S. Levine, "Optimal control with learned local models: Application to dexterous manipulation," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 378–383, 2016.

[14] O. Kroemer, C. Daniel, G. Neumann, H. V. Hoof, and J. Peters, "Towards Learning Hierarchical Skills for Multi-Phase Manipulation Tasks," pp. 1503–1510, 2015.

[15] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," vol. 17, pp. 1–40, 2015.

[16] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-Real Robot Learning from Pixels with Progressive Nets," no. CoRL, pp. 1–9, 2016.

[17] D. Kragic and H. I. Christensen, "Survey on Visual Servoing for Manipulation," tech. rep.

[18] OpenAI, "Learning Dexterous In-Hand Manipulation," pp. 1–27, 2018.

[19] J. R. Flanagan, M. C. Bowman, and R. S. Johansson, "Control strategies in object manipulation tasks," *Current Opinion in Neurobiology*, vol. 16, pp. 650–659, dec 2006.

[20] P. Pastor, "Skill Learning and Task Outcome Prediction for Manipulation," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3828–3834, 2011.

[21] S. Uematsu, Y. Kobayashi, A. Shimizu, and T. Kaneko, "Prediction of object manipulation using tactile sensor information by a humanoid robot," pp. 0–5, 2012.

[22] N. Wettels, V. J. Santos, R. S. Johansson, and G. E. Loeb, "Biomimetic tactile sensor array," *Advanced Robotics*, vol. 22, no. 8, pp. 829–849, 2008.

[23] N. Wettels, J. A. Fishel, and G. E. Loeb, "Multimodal tactile sensor," *Springer Tracts in Advanced Robotics*, vol. 95, pp. 405–405, 2014.

[24] H. B. Amor, O. Kroemer, U. Hillenbrand, G. Neumann, and J. Peters, "Generalization of Human Grasping for Multi-Fingered Robot Hands,"

[25] M. P. Deisenroth, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2011.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," pp. 1–12, 2017.

[27] J. Schulman, J. Eecs, B. Edu, P. Abbeel, P. Cs, and B. Edu, "Trust Region Policy Optimization," 2015.

[28] K. Kawakami, *Supervised Sequence Labelling with Recurrent Neural Networks*. PhD thesis, PhD thesis. Ph. D. thesis, Technical University of Munich, 2008.

[29] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "HIGH-DIMENSIONAL CONTINUOUS CONTROL USING GENERALIZED ADVANTAGE ESTIMATION," tech. rep.

[30] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[31] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *ICML*, vol. 2, pp. 267–274, 2002.

[32] N. Wettels, V. J. Santos, R. S. Johansson, and G. E. Loeb, "Biomimetic tactile sensor array," *Advanced Robotics*, vol. 22, no. 8, pp. 829–849, 2008.

[33] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.

[34] R. Y. Tsai and R. K. Lenz, "A new technique for fully autonomous and efficient 3d robotics hand/eye calibration," *IEEE Transactions on robotics and automation*, vol. 5, no. 3, pp. 345–358, 1989.

[35] Y. Bai and C. K. Liu, "Dexterous manipulation using both palm and fingers," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 1560–1565, IEEE, 2014.

[36] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[37] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[38] Y. Bengio, "Continuous control with deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[39] B. Zoph, Q. V. Le, and G. Brain, "NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING," pp. 1–16, 2017.

[40] P. Montague, "Reinforcement Learning: An Introduction," *Trends in Cognitive Sciences*, vol. 3, no. 9, p. 360, 1999.

[41] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained Policy Optimization," 2017.

[42] C. Doersch, "Tutorial on Variational Autoencoders," pp. 1–23, 2016.

[43] N. Wettels and G. E. Loeb, "Haptic feature extraction from a biomimetic tactile sensor: Force, contact location and curvature," in *2011 IEEE International Conference on Robotics and Biomimetics, ROBIO 2011*, pp. 2471–2478, 2011.

[44] J. Peters and S. Schaal, "Policy Gradient Methods for Robotics," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2219–2225, 2006.

[45] A. Petrovskaya, O. Khatib, S. Thrun, and A. Y. Ng, "Touch Based Perception for Object Manipulation," *Robotics Science and Systems, Robot Manipulation Workshop*, no. February, pp. 2–7, 2007.

[46] R. S. Johansson and J. R. Flanagan, "Coding and use of tactile signals from the fingertips in object manipulation tasks," 2009.

[47] K. Hsiao, L. Kaelbling, and T. Lozano-Perez, "Task-Driven Tactile Exploration," in *Robotics: Science and Systems VI*, 2010.

[48] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," mar 2017.

[49] H. V. Hoof, T. Hermans, G. Neumann, and J. Peters, "Learning Robot In-Hand Manipulation with Tactile Features," pp. 121–127.

[50] R. R. Ma and A. M. Dollar, "On dexterity and dexterous manipulation," in *2011 15th International Conference on Advanced Robotics (ICAR)*, pp. 1–7, IEEE, jun 2011.

[51] L. Cramphorn, B. Ward-Cherrier, and N. F. Lepora, "Tactile manipulation with biomimetic active touch," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, pp. 123–129, IEEE, may 2016.

[52] P. Falco, A. Attawia, M. Saveriano, and D. Lee, "On Policy Learning Robust to Irreversible Events: an Application to Robotic In-Hand Manipulation," *IEEE Robotics and Automation Letters*, pp. 1–1, 2018.