University of Central Florida

STARS

Electronic Theses and Dissertations

Doctoral Dissertation (Open Access)

Learning robotic manipulation from user demonstrations

2017

Rouhollah Rahmatizadeh
University of Central Florida

Find similar works at: https://stars.library.ucf.edu/etd

University of Central Florida Libraries http://library.ucf.edu

Part of the Computer Sciences Commons

STARS Citation

Rahmatizadeh, Rouhollah, "Learning robotic manipulation from user demonstrations" (2017). *Electronic Theses and Dissertations*. 5680.
https://stars.library.ucf.edu/etd/5680

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.

University of Central Florida

STARS
Showcase of Text, Archives, Research & Scholarship

www.manaraa.com

LEARNING ROBOTIC MANIPULATION FROM USER DEMONSTRATIONS

by

ROUHOLLAH RAHMATIZADEH
M.S. University of Central Florida, 2014
B.S. Sharif University of Technology, 2012

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida

Fall Term
2017

Major Professor: Ladislau Boloni

المنارة للاستشارات
www.manaraa.com

© 2017 Rouhollah Rahmatizadeh

www.manaraa.com

ABSTRACT

Personal robots that help disabled or elderly people in their activities of daily living need to be able to autonomously perform complex manipulation tasks. Traditional approaches to this problem employ task-specific controllers. However, these must to be designed by expert programmers, are focused on a single task, and will perform the task as programmed, not according to the preferences of the user. In this dissertation, we investigate methods that enable an assistive robot to learn to execute tasks as demonstrated by the user. First, we describe a learning from demonstration (LfD) method that learns assistive tasks that need to be adapted to the position and orientation of the user's head. Then we discuss a recurrent neural network controller that learns to generate movement trajectories for the end-effector of the robot arm to accomplish a task. The input to this controller is the pose of related objects and the current pose of the end-effector itself. Next, we discuss how to extract user preferences from the demonstration using reinforcement learning. Finally, we extend this controller to one that learns to observe images of the environment and generate joint movements for the robot to accomplish a desired task. We discuss several techniques that improve the performance of the controller and reduce the number of required demonstrations. One of this is multi-task learning: learning multiple tasks simultaneously with the same neural network. Another technique is to make the controller output one joint at a time-step, therefore to condition the prediction of each joint on the previous joints. We evaluate these controllers on a set of manipulation tasks and show that they can learn complex tasks, overcome failure, and attempt a task several times until they succeed.

www.manaraa.com

To my parents and whomever taught me something!

www.manaraa.com

ACKNOWLEDGMENTS

I gratefully thank my advisor who made my stay at UCF a memorable experience. I also would like to thank the members of my committee, Dr. Kenneth O. Stanley, Dr. Guo-Jun Qi, and Dr. Peter Hancock for their time and valuable comments.

I would also like to thank my labmates, collaborators, and friends for their support and friendship and the discussions about interesting ideas. Special thanks to my love Zahoora without whom I would have probably ended up finishing my educations with just a Master's degree!

المنارة للاستشارات
www.manaraa.com

TABLE OF CONTENTS

www.manaraa.com

www.manaraa.com

المنارة للاستشارات
www.manaraa.com

LIST OF FIGURES

المنارة للاستشارات

www.manaraa.com

www.manaraa.com

www.manaraa.com

LIST OF TABLES

المنارة للاستشارات

www.manaraa.com

CHAPTER 1: INTRODUCTION

Every day people do different Activities of Daily Living (ADLs) such as self-feeding, dressing, grooming, and personal hygiene. However, many disabled people and elderly are not able to perform ADLs independently. Assistive robotics, whether in the form of wheelchair mounted robotic arms or mobile robots with manipulators, promises to improve the independence and quality of life of the disabled and the elderly. Particularly, autonomous robots can be of great help to users by performing the ADLs for them. While most current systems rely on remote control, there are ongoing research efforts to make assistive robots more autonomous. This includes the identification of the objects, grasping and manipulation executed on behalf and in collaboration with the disabled user [20, 48, 9].

One approach to make robots autonomous is to hand-engineer a controller that is specific to a certain task. However, this approach requires a lot of time from the robot programmer for designing each controller. In addition, the designed controller cannot adapt itself based on the preferences of each user. On the other hand, Learning from Demonstration (LfD) is a technique in which users can demonstrate different tasks to the robot without having any knowledge about how to program robots. In this case, the learned controller is indeed based on the preferences of the user. Although LfD does not require large number of interactions with the environment for learning as in Reinforcement Learning (RL), it still needs relatively large number of demonstrations from humans.

In this dissertation, we discuss different approaches to learn from user demonstrations with as few demonstrations as possible. First we show that it is possible to learn from a very few number of demonstrations if we consider a class of tasks such as the ones performed with respect to a person's head. Second, we show that if we have perfect knowledge about the environment and sufficient

www.manaraa.com

number of demonstrations, without making simplifying assumptions about the structure of the task, we can generate robot arm trajectories that enable the robot to perform the task. For this purpose, we design a simulation environment to train and test the robot on human demonstrations. Then we show that the trained policy can be used in a real-world scenario where the pose of each object is known similar to the simulated environment. For learning to generate trajectory movements, we use recurrent neural networks (RNNs). These networks can remember the past events and predict the future based on those events. We show that RNNs can generate very smooth trajectories based on the current position and orientation of relevant objects. In addition, we show how to use reinforcement learning to extract the desired preferences of the user from a set of demonstrations with mixed preferences.

In the third part of this thesis, we extend the previous method by using images as the input to the neural network and also using multiple tricks to make the system more sample-efficient. The proposed approach combines the data from multiple tasks to learn a single recurrent neural network policy. The policy takes as input images of the environment and a task selector one-hot vector, and predicts the joints of the robot in the next time-step. For processing the visual input, we use convolutional layers that are shared with the encoder part of an autoencoder which reconstructs the input images.

Since we do not make any assumption about the input images and the type of task that the robot needs to learn, the approach needs relatively large number of training samples which is not currently easy to obtain in the robotics domains. To make the approach more sample efficient, we use a single neural network and train it with all the data that we have from some related tasks. This way, the patterns in one task can be captured and used in other tasks. In addition, we predict only one joint at each time-step. As a result, the output neurons are shared among different joints. These aggressive parameter sharing mechanisms reduce the total number of parameters in the network, reduce overfitting, and have a regularizing effect.

المنارة للاستشارات
www.manaraa.com

One of the challenges in LfD is that humans might occasionally make mistakes during the demonstrations. These mistakes might seem unwanted in the first glance. However, we argue that they help to train a more robust model that can overcome its own mistakes since it has seen similar mistakes in the training. On the other hand, we do not want the model to repeat the same mistakes very often. Therefore, we use a trick to diminish the probability of mistakes and generate a trajectory that is very robust and at the same time overcome the mistakes if they happen.

The remainder of this dissertation is organized as follows. Chapter 2 describes related works. In Chapter 3 we describe an approach to learn human head dependent tasks from a few demonstrations. In Chapter 4 we explain our method for learning trajectories using recurrent neural networks given object poses as input. Chapter 5 extends the network architecture to accept images as input and learn multiple tasks all at once. We conclude and talk about the future work in Section 6. This thesis includes the research works by the author [56, 55, 57].

www.manaraa.com

CHAPTER 2: RELATED WORK

Two main approaches to learning robot behaviors are Learning from Demonstration (LfD) and Reinforcement Learning (RL). In LfD, the user demonstrates the tasks to the robot while in RL the robot tries to find the optimal policy by gathering experience from its interactions with the environment. In LfD the challenge is to extend these demonstrations to unseen situations [5]. LfD has achieved many successful applications such as autonomous helicopter maneuvers [1], playing table tennis [13], object manipulation [51], and making coffee [65]. This success comes at the expense of user's time to demonstrate the tasks to the robot. On the other hand, RL approaches enable the robot to acquire different skills by itself [38, 52].

In some domains, gathering relatively large number of examples is possible in the simulation environment(e.g. [2, 3]). However, the challenge in many LfD applications is to use a minimum number of examples to learn a task. The examples may include situations in which the geometry of the objects change from the demonstration scene to the test scene. In this case, non-rigid registration can be used to map the camera input points from the training scene onto the test scene [63]. This registration is later used to adapt the trajectory of the robot arm captured during the demonstration to the testing situation. It turns out that non-rigid registration works well as long as the environment does not change too much such that a valid registration could not be found. For example, when person's head rotates $180°$, most of the points of the demonstration are not visible anymore, hence, finding a warping function fails.

Another method to capture the critical aspects of demonstrated trajectories and maintain them during the test is investigated in [76]. Their method handles new positions of the involved objects while avoiding new obstacles in the scene by using motion planning algorithms. A method that utilizes averaging to generalize the trajectory to the scenes where the object position is changed

www.manaraa.com

is proposed by [60]. Neither method considers the orientation of the objects, thus they cannot be applied to situations where a non-symmetric object rotates.

Another approach is to model the recorded movements using a set of differential equations in the dynamic movement primitives framework [51]. In another work this framework is adapted to generate trajectories that end at goals at different positions while avoiding obstacles. Also, [12] uses a Hidden Markov Model to capture the constraints of demonstrated trajectory. They use a set of pre-defined landmarks for each trajectory to track the changes from demonstration to test. More recently, [30] worked on improving the demonstrated trajectories based on user preferences. We can also consider force information in LfD. For instance, [61] proposed a LfD framework teaching force-based manipulation tasks to robots.

Both RL and LfD approaches need relatively large number of examples for training if we do not enter simplifying assumptions to the learning algorithm. To overcome this problem, some researchers proposed cloud-based and crowd-sourced data collection techniques [34, 22, 16]. Another way to reduce the number of necessary demonstrations is to hand-engineer task-specific features [11, 12]. Other researchers have tried to reduce the number of required examples by transfer learning or multi-task learning [69, 39, 58, 46, 77]. An early work considered learning a neural network policy on multiple related tasks with backpropagation [14]. It is possible to learn this kind of policies by considering both the state and the task as input to the policy [17]. Neural network policies can be decomposed to different modules where there are some task-specific modules and some robot-specific modules [18]. It is also shown that sharing the parameters of a neural network among different tasks not only improves the results, but also it is even better to train the model using the data of multiple related tasks instead of using the same amount of data from the original task [53].

Deep neural networks have proven to be very powerful in learning very complex functions. Different approaches are proposed to model human neurons [59, 29], however, a very simplified model

www.manaraa.com

of human neuron is shown to be a sufficient building block of these networks. Although these networks can be fooled by some input images [68], there are ways to avoid this [32, 31]. Deep networks are being successfully used to map robot's visual input to control commands [43, 54, 4, 44, 45]. In Chapter 5, we propose an end-to-end deep neural network architecture that can learn multiple tasks with low amount of data compared to a single-task network. Our architecture predicts each joint of the robot at one time-step. By this approach, the weights are shared among different joints of the robot, therefore learning from small number of examples without easily overfitting is realized. This neural network architecture with some modifications is also used to predict taxi demand throughout large city [28, 75].

Our architecture utilizes the power of Recurrent Neural Networks (RNNs), an effective tool to model and reproduce patterns in sequential data. Some of successful applications of RNNs include handwriting generation [25], language modeling [33], machine translation [66], speech recognition [24], visual recognition [19], and image captioning [73]. Although recurrent neural networks had been proposed as early as the 1980s, early versions suffered from the difficulty of training over sequential data, due to the difficulty to account for events occurring at different times in the training data sequences (the "vanishing error gradient" problem). Succeeding versions of RNNs such as LSTMs [26] feature explicit gating mechanism that helped in storing and retrieving information over long time periods. In recent years, similar mechanisms were proposed such as Gated Recurrent Units (GRU) [15].

المنارة للاستشارات
www.manaraa.com

CHAPTER 3: TRAJECTORY ADAPTATION

Learning a wide variety of tasks from user demonstrations might be difficult since the learning algorithm does not know anything about the task and needs to learn everything from scratch. If we do not target a specific class of tasks, we cannot provide the learning algorithm with some simplifying assumptions to be able to learn from a few number of demonstrations. Therefore, one approach to make LfD more sample efficient is to have some class of tasks in mind and try to generalize from a very small number of examples by putting our intuition and assumptions into the learning algorithm.

In this chapter we consider a subset of ADL tasks that are dependent on the head pose of the user. For instance, consider feeding (using forks, spoons, glasses, bringing bottles of juice or medication to the mouth), personal grooming (combing the hair, shaving, brushing teeth) as well as other ADLs such as reading a book or participating in a video chat. In these tasks the assistive robot cannot simply reproduce pre-programmed or rigidly demonstrated trajectories. The trajectory must be dependent on and adapting to the current head pose of the user. For instance, when feeding the user, the robot needs to bring the fork to the mouth of the user – other trajectories are ineffective and potentially dangerous.

Adapting to the current head pose is not a problem if the trajectory had been calculated from scratch using a formal model of the task. Unfortunately, many ADLs are difficult to describe formally - they might depend on the environment and the preferences of the user. One of the most desirable ways to teach a robot to perform an ADL is LfD - a technique in which the task is demonstrated to the robot either by manually guiding its arm or by teleoperation. One of the advantages of LfD is that the user can demonstrate the task according to their own preferences. Ideally, from a small number (possibly, just one) demonstration, the robot should be able to generalize the learned

www.manaraa.com

trajectories to a new environment or state. Due to the increase in the problem complexity, previous LfD implementations that adapt the demonstrated trajectories to new situations usually do not generalize to the 3D pose of the objects in the environment.

In this chapter, we propose a method to adapt demonstrated trajectories to situations in which the pose of the human head has changed. Our method relies on a few example demonstrations of a task, and using geometrical transformations, adapts the trajectories to situations where the position and the orientation of the head has changed. Figure 3.1 shows an example of how the arm of a Rethink Robotics Baxter robot needs to adapt the trajectory of the arm bringing food to the user's mouth as the user moves and rotates his head.

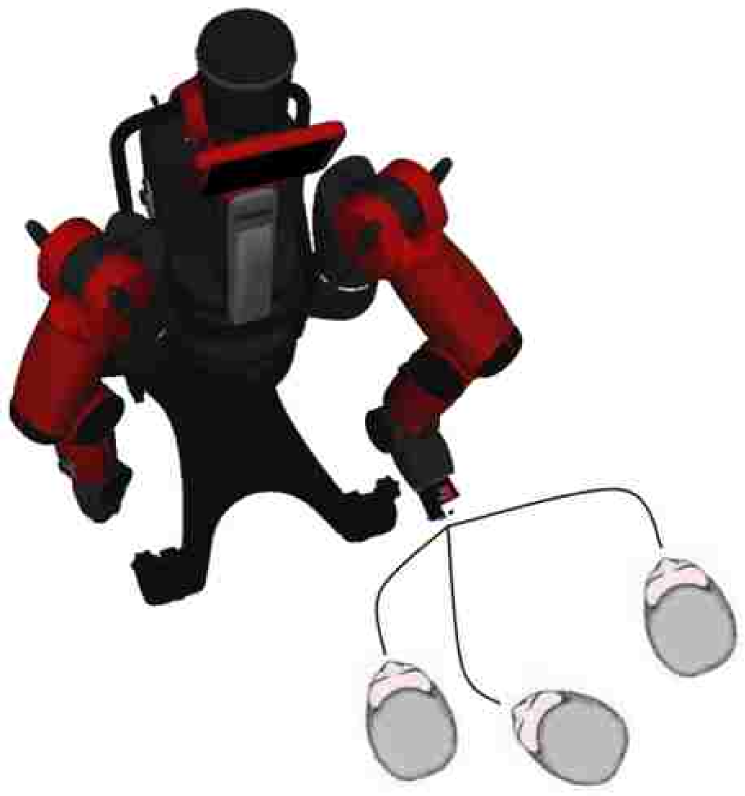


Figure 3.1: Trajectory adaptation of the arm of the Baxter robot in a feeding task. As the head pose changes, the robot arm must traverse trajectories of different shape and length.

www.manaraa.com

In learning from demonstration, we start by performing the task for the robot and recording the executed trajectory. A task can be executed using different trajectories based on the state of the environment. For example, a robot performing a head pose dependent ADL task might use either its right or left arm depending on which one is closer to the head. Therefore, we record trajectories augmented with the state of the environment at every time step. For the purpose of the work described in this chapter, the state consists of only the head pose. However, a complete sequence of RGB-D frames from the camera can be considered as the state if we have enough example demonstrations.

The trajectory of the robot arm is usually represented in *joint space* by keeping track of the value of each joint at each time step. This trajectory can also be shown in *task space* in which the pose of the end-effector is stored. We use task space trajectory in order to be able to transform it in 3D space.

Each trajectory might be achieved by executing a few demonstrations, using dynamic time warping [62] to synchronize them, and then do averaging on the resulting trajectories.

For each task, we record N demonstrations $D = \{d_1 \dots d_i \dots d_N\}$. A demonstration $d_i = \{E, Q\}$ consists of Q, the state of environment including head pose H, and $E = [e_1 \dots e_t \dots e_T]$ a set of end-effector poses e_t at time $t = [1 \dots T]$. Pose $e_t = [X, Y, Z, \phi, \alpha, \psi]$ is the vector containing the position and orientation (roll, pitch, yaw) of the end-effector with respect to origin. Similarly, we show the set of head poses during the demonstration by $H = [h_1 \dots h_t \dots h_T]$ in which h_t is the head pose at time t.

Naturally, the robot is able to retrace a learned trajectory, which would work fine provided that the evolution of the head poses during test are the same as during the demonstration. The challenge we

www.manaraa.com

are trying to solve is that the head pose during test time is different from the demonstration ones: $H' = [h_1 \dots h_{t'} \dots h_{T'}]$ where $h_{t'}$ is the head pose at time t'. The goal is to find the corresponding end-effector trajectory during the test time $E' = [e_1 \dots e_{t'} \dots e_{T'}]$. Note that not only the head poses might be different, but also the total time of the trajectory t' might be different at test time than the time t it took at demonstration. For example, consider the scenario shown in Figure 3.2 in which during the demonstration the arm traverses a straight trajectory towards the head. During the test, however, the trajectory takes longer to be executed since the head moves simultaneously. This change in trajectory execution time might occur because we prioritize maintaining the end-effector pose relative to the head pose rather than progressing in execution of the trajectory.

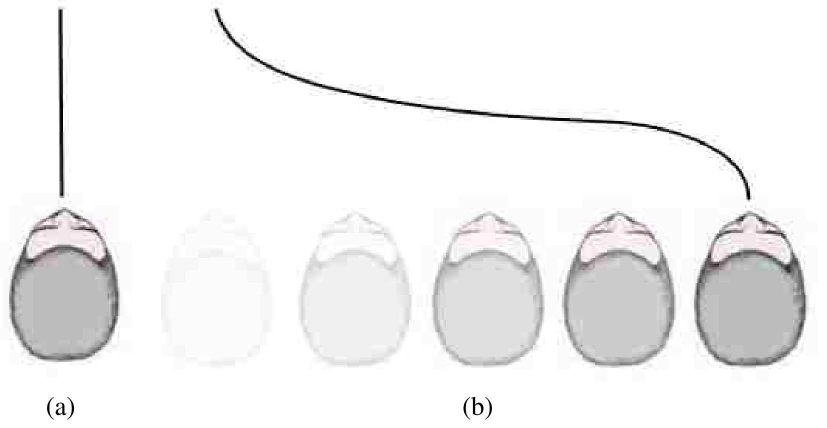


Figure 3.2: The duration of executed trajectory at test might be different from the demonstration. Demonstrated trajectory (a) is a straight trajectory towards the head. The desired trajectory at test time (b) is longer since the head moves during the execution.

In many tasks, the robot should respond to the movements of the head. For example, consider the task of holding a book for a person such that the person can easily read it. When the person rotates his head, the position and the orientation of the book should be changed accordingly. This problem of capturing the relationship between the head pose and the end-effector pose, can be captured using the method proposed in [10]. However, this approach needs dozens of demonstrations from

المنارة للاستشارات
www.manaraa.com

different orientations and positions. If we want to minimize the number of these demonstrations, they need to be designed by an expert such that they capture different aspects of this relationship.

Gathering dozens of demonstrations per task is not easy in assistive robotics where the task must be adapted to the preferences and environment of each user. In the next section, we will explain a solution to overcome this problem by using as little as a single trajectory to accomplish difficult tasks.

Method

In this section, we explain the steps to adapt the trajectories from the demonstration to the test situation. The real-time head pose of the user is collected using a Kinect sensor mounted on the robot. We use this information to transfer each waypoint of a demonstrated trajectory to make a new trajectory. The input of this transformation is the 3D pose of the end-effector, so the result would be a 3D trajectory of the end-effector. Therefore, in order for the robot to be able to execute the trajectory, we convert the trajectory from task space to joint space using inverse kinematics.

Let us start by defining the notations used in the remaining of this section. Each pose in a 3D world can be uniquely described by its position and orientation. We define operators $p(x)$ and $r(x)$ which decompose pose x into a translation vector and a rotation matrix respectively. We also define $p(\Delta(x, y)) = p(x) - p(y)$ to be a translation vector from x to y and $r(\Delta(x, y)) = r^{-1}(y)r(x)$ will be the difference between two rotation matrices.

للاستشارات المنارة
www.manaraa.com

Finding changes in the head pose

The head pose is extracted using the random regression forests method proposed by [21]. In practice, we found the output of this method to be relatively noisy. By taking advantage of the fact that we are recording a continuous scene at fixed intervals, we applied an Exponentially Moving Average (EMA), a common noise reduction technique for time-series data:

$$h_t = \alpha \tilde{h_t} + (1 - \alpha) h_{t-1} \tag{3.1}$$

where $\tilde{h_t}$ is the noisy head pose, and h_t is the filtered head pose by considering previous head poses with more emphasis on the most recent ones. The discount factor α controls how much weight we give to the old data, which is set to 0.2 in our experiments.

Each demonstration consists of multiple trajectories. At each time step, we need to select from the demonstrated trajectories one that is "closest" to the test situation. For this purpose, we use a K-Nearest Neighbor (KNN) classifier to decide which head pose in the demonstrated trajectories is closest to the current head pose at this time step. Based on this prediction, we select a waypoint of the corresponding trajectory at the current time step to be translated. The distance measure used in KNN is as follows:

$$d(h_t, h_{t'}) = \|p(\Delta(h_t, h_{t'}))\| + \beta \|r(\Delta(h_t, h_{t'}))\| \tag{3.2}$$

where $h_{t'}$ is the head pose at test time step t', h_t is the head pose at demonstration time step t, and the parameter β controls how much weight we give to the orientation compared to the position. Note that the selection of the trajectory occurs at each time step during the execution of the trajectory. In other words, at each time step, we consider the head pose in each trajectory and select the closest one, then we go to the next time step. This approach is justified by the requirement

www.manaraa.com

for the system to be able to respond in real-time to head pose changes during the execution of the trajectory. In most of the previous works (e.g. [42, 63]) this evaluation is preprocessed before the execution of the trajectory; hence, the system does not react to the changes in the environment. In next section we explain how to transfer the end-effector pose e_t to match the new head pose $h_{t'}$.

$$d_{exe} = \left\{ d_i \left| i = \min_{h_t \in D} |h_t - h'_{t'}| \right. \right\} \tag{3.3}$$

Transferring end-effector poses

The objective in this part is to calculate $p(e_{t'})$ and $r(e_{t'})$ which is position and orientation of robot's end-effector at test time t' based on $p(h_t)$, $r(h_t)$, $p(h_{t'})$, $r(h_{t'})$, $p(e_t)$ and $r(e_t)$. The assumption is that the end-effector should maintain its previous pose with respect to the head. To simplify the problem, let us divide the changes in the pose of the head to changes in its position and changes in its orientation. If the head only moves without any change in its orientation, the desired pose for the end-effector can be achieved by the same head translation:

$$p_{trans}(\Delta(e_{t'}, e_t)) = p(\Delta(h_t, h_{t'})) \tag{3.4}$$

On the other hand, if the head only rotates and does not move as shown in Figure 3.3, the end-effector's corresponding action will be a translation to keep the same position with respect to the head:

We define two operators $p(x)$ and $r(x)$ which decompose pose x into a translation vector and a rotation matrix respectively. In addition, for any arbitrary poses x and y, we show the difference between the positions by $p(\Delta(x, y)) = p(x) - p(y)$ and the difference between two rotations matrices by $r(\Delta(x, y)) = r^{-1}(y)r(x)$. We need to to find $e'_{t'}$ based on e_t, h_t, and $h'_{t'}$. When the

www.manaraa.com

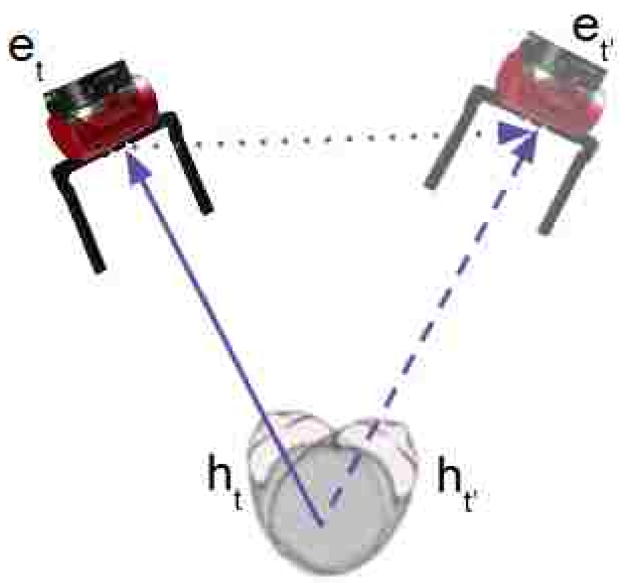


Figure 3.3: Illustration of equation 3.5. The solid vector is $p(\Delta(h_t, e_t))$, the dashed vector is $r(\Delta(h_{t'}, h_t))p(\Delta(h_t, e_t))$ and the dotted vector is $p_{rot}(\Delta(e_{t'}, e_t))$.

head rotates, the end-effector should move and rotate to stay in the same orientation with respect to head. For this purpose, we transform the end-effector to the center of the head, rotate with the head rotation, and move it back to its previous distance from the head:

$$p_{rot}(\Delta(e_{t'}, e_t)) = r(\Delta(h_{t'}, h_t))p(\Delta(h_t, e_t)) - p(\Delta(h_t, e_t)) \tag{3.5}$$

and a rotation to adjust the orientation.

$$r_{rot}(\Delta(e_{t'}, e_t)) = r(\Delta(h_{t'}, h_t))r(e_t) \tag{3.6}$$

المنارة للاستشارات
www.manaraa.com

Now we can calculate the overall translation and rotation matrix from e_t to $e_{t'}$ as

$$p_{overall}(\Delta(e_{t'}, e_t)) = p_{trans}(\Delta(e_{t'}, e_t)) + p_{rot}(\Delta(e_{t'}, e_t)) \tag{3.7}$$

$$r_{overall}(\Delta(e_{t'}, e_t)) = r_{rot}(\Delta(e_{t'}, e_t)) \tag{3.8}$$

in which $r(\Delta(h', h))$ is a rotation matrix, $p(\Delta(h, e))$ is the vector from the human head to the arm at any time. When the head moves in any direction, the end-effector should also move in the same direction. Therefore, we augment $prj(e, h)$ with the effect of head movement by adding $p(\Delta(h', h))$ to it to achieve the difference between the new position of the end-effector at test situation and its position at demonstration:

$$p(\Delta(e'_{t'}, e_t)) = prj(e, h) + p(\Delta(h', h)) \tag{3.9}$$

When the head rotates, the end-effector should also rotate to maintain its previous orientation with respect to the head. So, the difference between the new rotation of the end-effector and its previous rotation during the demonstration can be calculated as follows:

The calculations we have performed up to this point are finding the new pose of a single waypoint in a trajectory. In practice, however, we need to transform the whole trajectory so that it can perform the same task with respect to the new pose of the head. The first observation we make is that not all the points in the trajectory are required to be transformed equally: for instance, points closer to the head require should be translated more compared to the points far away from the head. To decide how much a point in a trajectory should be transformed, we use the following logistic

المنارة للاستشارات

www.manaraa.com

function as a Translation Factor (TF) for each waypoint based on their distance to the head:

$$TF(d) = \frac{1}{1 + e^{k(d-\hat{d})}} \tag{3.10}$$

where $d = p(\Delta(e_t, h_t))$ is the euclidean distance between the end-effector and the center of the head, $\hat{d}$ is the midpoint of transformation, and k is the slope of the transformation. The translation can vary from 0 when the end-effector is far enough from the head to 1 when the end-effector is near the head. The parameters $\hat{d}$ and k can be decided based on the task. Figure 3.4 shows a plot of function $TF(d)$ with parameters $k = 50$ and $\hat{d} = 0.3m$

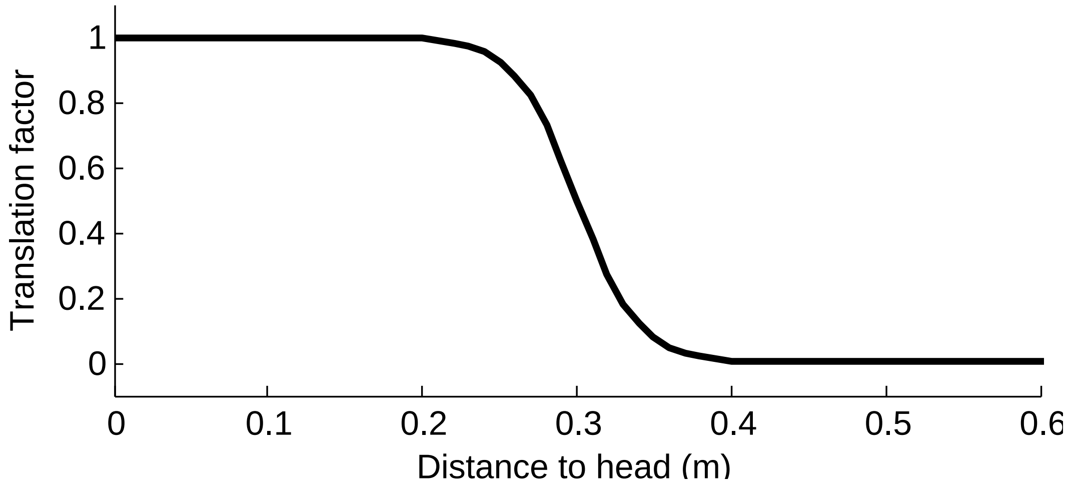


Figure 3.4: Translation factor shows how much a point in the trajectory should be transformed which is a function of distance between that point and the head.

Finally, we can achieve the new position and orientation of the end-effector using these formulas:

$$p(e_{t'}) = p(e_t) + p_{overall}(\Delta(e_{t'}, e_t)) \times TF(d) \tag{3.11}$$

www.manaraa.com

$$r(e_{t'}) = r(e_t) + r_{overall}(\Delta(e_{t'}, e_t)) \times TF(d) \tag{3.12}$$

Converting end-effector trajectory to joint angles

We represented each trajectory as a set of waypoints containing robot end-effector poses. Then, we transferred each waypoint from the training to test situation. Note that in both of these cases we excluded from the state the joints of the robot. However,

In order for the robotic arm to be able to execute the trajectory, a joint configuration should be found such that the end-effector reaches the desired pose. This can be achieved using inverse kinematics, which also needs to consider the obstacles in the environment (to prevent, for instance, the robot's elbow joint bump into obstacles while performing the transformed movement). Another problem is that of trajectory smoothness. We assume that the demonstration used a smooth movement to perform the task. It is possible, however, that the trajectory resulting from the transfer is not going to be smooth, because, as shown in Figure 3.2, the change in the head pose might insert new trajectory sequences. If the robot tries to make these corrections too quickly in an attempt to keep to the original schedule, it can result in a jerky movement. To mitigate this problem, before transferring the trajectory from task-space to joint-space, we interpolate between the successive end-effector poses which are far from each other in 3D space. Then, we transfer the trajectory to the joint space. We also designed a mechanism to control the speed of the arm by interpolating between successive joint configuration waypoints. This control is implemented at the joint level just before the command is sent to the robot to make sure that the trajectory is smooth and safe.

Note that the corresponding joint configurations might not be executed smoothly. There might be two consequent waypoints for the end-effector that close in 3D space, however, the corresponding joint configurations are far from each other. This causes the joint space trajectory to become jerky

www.manaraa.com

after transfer. Since the robotic arm is executing the trajectories in close proximity to human head, it is extremely important to consider safety measures and avoid any dangerous movements by the arm.

Let us assume that the arm is in configuration $J = \{j_1 \ldots j_i \ldots j_M\}$ at time-step t in which j_i is the configuration of ith joint and M is the total number of joints in the arm. The next desired joint configuration after probably using linear and spherical interpolation on the next desired pose and then doing inverse kinematics would be $J' = \{j'_1 \ldots j'_i \ldots j'_M\}$. We define $\Delta(J', J)$ as the vector of difference between each joint configuration.

Experimental validation

We have implemented our technique using the Baxter robot by Rethink Robotics. Baxter has a zero-force gravity compensation mode in which a user can steer the robot's arm to desired configurations. While the user is moving the arm, we record the joint configurations and also the pose of the end-effector at a frequency of 20Hz. This series of recorded end-effector poses augmented with the gripper status forms the trajectory of the arm.

For our experiments we considered three different ADLs:

- Holding a book for the user such that he can read it.
- Facilitating a video chat by recording the face of the user using a camera mounted on the wrist of the robot.
- Bringing a bottle of water close to the user's head.

The first two tasks demonstrate how our algorithm can adapt the demonstrated trajectory in real-

المنارة للاستشارات
www.manaraa.com

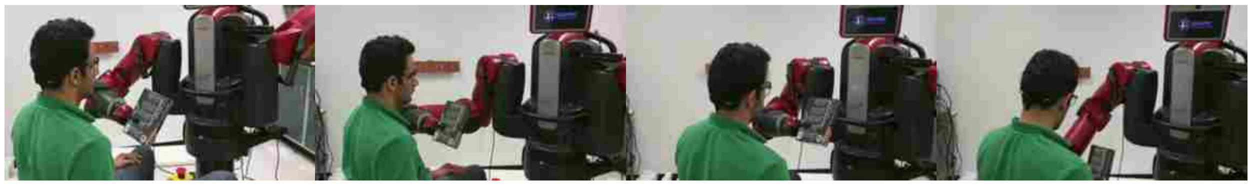


Figure 3.5: Sequence of images demonstrating the task of holding a book for the user to read. Notice that as the the user turns his head, the robot positions the book for a comfortable reading position at an appropriate reading distance.

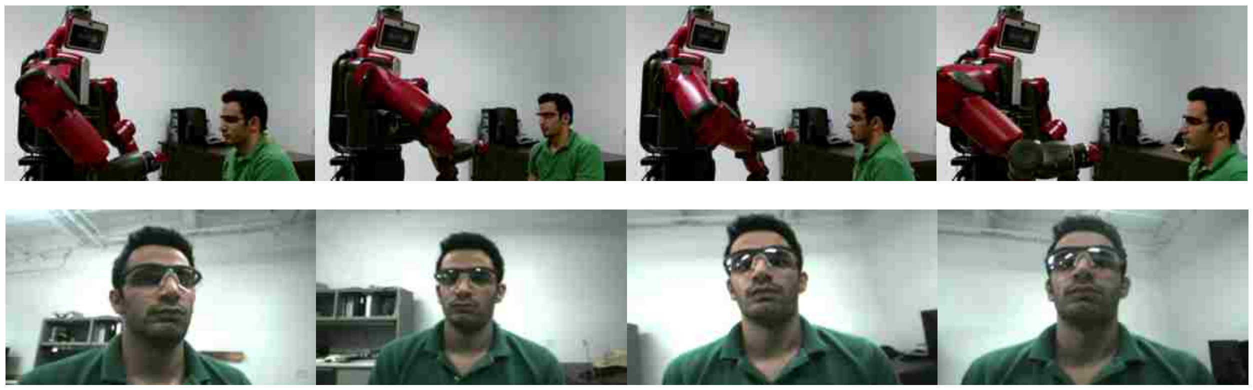


Figure 3.6: Sequence of images demonstrating the execution of the tasks of recording a video of subject's face for facilitating a video chat. The top row shows the relative position of the user and the Baxter robot, while the bottom row shows the video captured. Notice that although the user had moved around significantly, his face remains centered in the video stream.

time as the head moves. The third task is designed for testing execution of a trajectory and see how the trajectory adaptation happens as the end-effector comes closer to the head.

The experiments had been performed as follows. A human subject sits in front of the robot in such a way that the robot arm can reach his head. A Microsoft Kinect sensor mounted on the Baxter tracks the head pose of the user by capturing RGB-D frames. Based on the relative pose of the Baxter and the Kinect with respect to each other in the real world, we use a translation matrix to convert the points from Kinect coordinate system to the Robot's coordinate system. The

www.manaraa.com

captured frames are processed in real-time and the extracted head pose is recorded at the same rate as recording end-effector trajectory waypoints.

In the task of recording video from the person's face, the camera on the Baxter's arm is used. In this task, the head of the person is centered to the camera frame. We expected that by using our proposed method, the head should remain in center when the person moves or rotates his head.

Results and limitations

The sequence of images in Figure 3.5 shows how the robot performs the tasks of holding a book for the subject while Figure 3.6 shows the task of facilitating a video chat. In addition, the video of the robot executing the tasks can be found online[1]. In the tasks of holding a book and recording video, our algorithm could successfully adapt the demonstrated trajectory in real-time as the human subject was moving.

For normal operating conditions we have found that the robot was able to achieve all the three tasks successfully. We have, however, also identified some limitations of the trained trajectories. In the following we shall discuss these limitations and whether they can be ameliorated by future work.

Reacting to fast movement. Let us consider that task of recording a video of human face. The task started when the head was centered in the video frame. Then, the person started to move his head in arbitrary directions. We found that most of the time, the robot could find a proper pose for its arm so that the head remains centered in the video frame. However, if the head moved very fast, there could be a delay of several seconds before the robot arm caught up and re-centered the head in the camera. This delay could be reduced at the expense of faster movements by the robot arm. However, since abrupt movements in proximity of human head are dangerous, we preferred

[1]`https://youtu.be/BU775Wdd4JU`

www.manaraa.com

to keep the speed of the arm slow.

Moving outside of the range: For the video recording and book reading tasks, we have found that in some cases, when the head moved in a wide range, the robot arm stopped moving, because the desired pose for the end-effector was not reachable by the robot arm. In other words, the inverse kinematics algorithm could not find the joint configurations to put the end-effector in the desired pose. As the Baxter robot we used in these experiments is stationary, the range limitations are unavoidable. For mobile robots, we would need to perform concurrent planning for the base mobility and arm movement to avoid this problem.

Self-occlusion: One problem sometimes occurring during the execution of the video recording task was that to place the end-effector to the desired pose, the inverse kinematics finds a joint configuration that puts the arm between the camera and the head. In this case, view of the face is occluded by the arm, thus the head pose cannot be figured out. As a result, the algorithm stops moving the arm until a new information about the current head pose is received. In future work, we plan to introduce in the motion planning algorithms the ability to predict such occlusions and try to avoid them when possible.

Tuning the translation factor: In the task of bringing a bottle of water close to the human head, the translation factor TF plays an important role. If we set the parameter so that it is a large number even for points far from the head, the rate of unsuccessful trials will increase. On the other hand, if we tune it so that even the points close to the head are not translated completely, the bottle will not end up in a proper position close the the subject's head. Therefore, a trade-off needs to be made to tune this parameter.

Limitations of the head pose accuracy: Finally, another factor is that the method for extracting the head pose from the camera input is not reliable enough for some tasks which need the arm to

المنارة للاستشارات
www.manaraa.com

be very close to the head. For instance, the current algorithms are too imprecise for tasks such as feeding the user with a fork. In addition, sometimes the head might be occluded by the arm. Hence, in order to increase the safety, other mechanisms such as using force sensors must will need to be incorporated into the method.

المنارة للاستشارات
www.manaraa.com

CHAPTER 4: LEARNING FROM SIMULATED DEMONSTRATIONS

In this chapter, we propose an approach where the users demonstrate the tasks to be performed in a virtual environment that is similar to their living environment and objects. This allows the collection of a larger number of demonstrations under various scenarios. For each demonstration we collect the pose of the objects involved as well as the gripper. The collected trajectories are used to train a neural network, which will serve as the robot controller: at each timestep it receives as input the current state and predicts the next waypoint in the trajectory and open/closed status of the gripper. We also discuss a way to extract the preferences of users from the demonstrated trajectories using reinforcement learning. The general flow is illustrated in Figure 4.1.

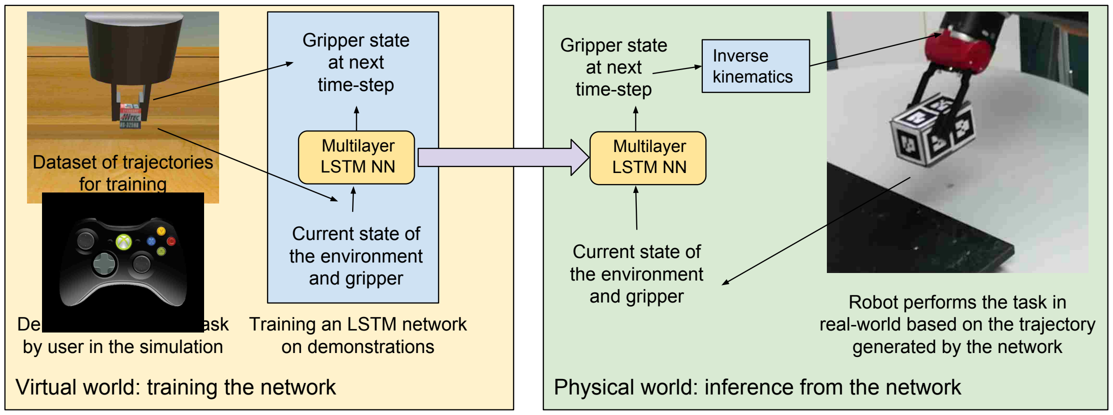


Figure 4.1: The general flow of learning from simulated demonstrations approach. The demonstrations of the ADL manipulation tasks are collected in a virtual environment. The collected trajectories are used to train the neural network controller. The trained controller is then transferred to the physical robot.

www.manaraa.com

Method

Collecting demonstrations in a virtual environment

The virtual environment

In the first step users demonstrate the ADLs to the robot in a virtual environment. To enable this we designed in the Unity3D game engine a virtual environment modeling a table with an attached shelf that can hold various objects. The virtual environment also contains a simple two-finger gripper that can be opened and closed to grasp and carry an object. The user can use the mouse and keyboard to open/close the gripper, as well as to move and rotate it in the 3D Cartesian space. Alternatively, a joystick can be used for the same tasks. Thus, the gripper has 7 degrees of freedom. The environment can also contain one or more movable objects, which have their own position and orientation. Unity3D simulates the basic physics of the real world including gravity, collision between objects and friction.

Trajectory representation

We represent the state of the virtual environment as the collection of the poses of the M movable objects $q = \{o_1 \dots o_M\}$. The pose of an object is represented by the vector containing the position and rotation quaternion with respect to the origin $o = [p_x, p_y, p_z, r_x, r_y, r_z, r_w]$. During each step of a demonstration, at time step t we record the state of the environment q_t and the pose of the end-effector augmented with the open/close status of the gripper e_t. Thus a full demonstration can be recorded as a list of pairs $d = \{(q_1, e_1) \dots (q_T, e_T)\}$. The duration of a trajectory T is determined by the moment when the user successfully finishes or abandons the experiment and it varies from demonstration to demonstration. The temporal resolution at which the states are recorded depend

المنارة للاستشارات
www.manaraa.com

on the requirements of the experiment.

Manipulation tasks considered

The majority of ADLs involve object manipulation, and a very large majority of these involve objects located on horizontal surfaces such as tables. For our experiments we considered two manipulation tasks that are regularly found as components of ADLs: pick and place and pushing to a desired pose.

The *pick and place* task involves picking up a small box located on top of the table, and placing it into a shelf above the table. The robot needs to move the gripper from its initial random position to a point close to the box, open the gripper, position the fingers around the box, close the gripper, move towards the shelf in an orientation where it will not collide with the shelf, enter the shelf, and finally open the gripper to release the box.

The *pushing to desired pose* task involves moving and rotating a box of size $10 \times 7 \times 7$cm to a desired area only by pushing it on the tabletop. In this task, the robot is not allowed to grasp the object. The box is initially positioned in a way that needs to be rotated by 90° to fit inside the desired area which is 3cm wider than the box in each direction. The robot starts from an initial gripper position, moves the gripper close to the box and pushes the box at specific points at its sides to rotate and move it. If necessary, the gripper needs to circle around the box to find the next contact point.

Collected dataset

Using the virtual environment described above, we collected a series of demonstrations for both manipulation tasks. The demonstrations were collected from a single user, in the course of multiple

المنارة للاستشارات
www.manaraa.com

sessions. In each session, the user performed a series of demonstrations for each task. The quality of demonstrations varied: in some of them, the user could finish the task only after several tries. For instance, sometimes the grasp was unsuccessful, or the user dropped the object in an incorrect position and had to pick it up again. After finishing a demonstration, the user was immediately presented with a new instance of the problem, with randomly generated initial conditions. All the experiments had been recorded in the trajectory representation format presented above, at a recording frequency of 33Hz. However, we have found that the neural network controller can be trained more efficiently if the trajectories are sampled at a lower rate. We found a sampling rate of 4Hz to give the best results.

Although recording in the virtual environment allowed as to record significantly more demonstrations than it would have been possible with a physical robot, we have found that we needed ways to improve the number of training trajectories. We have done this by exploiting both the properties of the individual tasks and the capabilities of our trajectory recording technique.

First, we noticed that in the pick and place task the user can put the object to any location on the shelf. Thus we were able to generate new synthetic training data by shifting the existing demonstration trajectories parallel to the width of the shelf. As the pushing to desired pose task requires a specific coordinate and pose to succeed, this approach is not possible for the second task.

The second observation was that by recording the demonstration at 33Hz but presenting the training trajectories at only 4Hz, we have extra trajectory points. These trajectory points can be used to generate multiple independent trajectories at a lower resolution. The process of the trajectory generation by frequency reduction is shown in Figure 4.2.

Table 4.1 describes the size of the final dataset. As the table shows, the average number of waypoints in a trajectory for the pick and place task was 20.68, while the same number was 28.61 for the push to desired pose task. This data, based on the human input, shows that the second task was

www.manaraa.com

more difficult than the first *for the human operator*.

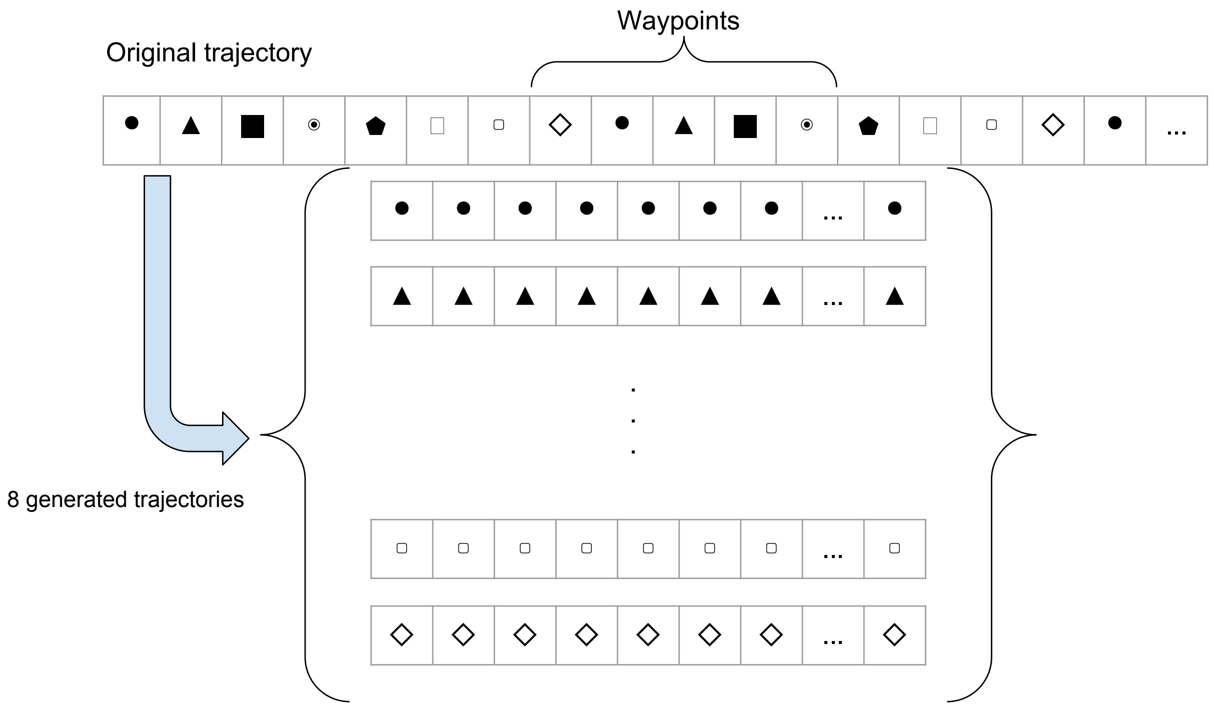


Figure 4.2: Creating multiple trajectories from a demonstration recorded at a higher frequency.

Table 4.1: The size of the datasets for the two studied tasks

Task	Pick and place	Push to pose
Raw demonstrations	650	1614
No. of demonstrations after shift	3900	-
No. of demonstrations after frequency reduction	31,200	12,912
Total no. of waypoints	645,198	369,477
Average demonstration length (waypoints)	20.68	28.61

The neural network based robot controller

The next step of our workflow is to design and train a neural network based robot controller that is able to generate robot trajectories. This controller takes as input the pose of the objects involved

www.manaraa.com

and the pose and open/close status of the gripper at time t. The output of the controller is a prediction of the pose and the open/closed status of the gripper at time $t + 1$. During training, this prediction is used to generate the error signal. During the deployment of the trained network, the prediction represents the desired pose of the end actuator which the robot needs to achieve through its inverse kinematics calculations.

In this series of experiments, we have used a separate controller for the pick and place and the push to desired pose tasks. These controllers have a similar network architecture but had been trained on the specific tasks. In the next chapter, we show that it is also possible to use a single neural network to learn all the manipulation tasks.

Let us discuss the choice of the architecture of the neural network. In particular, there are two important decision points: the nature of the network layers and the cost function used to train the network.

Our choice for this architecture is to use an LSTM recurrent neural network and rely on mixture density networks (MDNs) to predict the probability density of the output. The error signal, in this case, is based on the negative logarithm likelihood of the next target waypoint given the probability density implied by the MDN.

In order to determine whether our proposed approach represents a progress over the current state of the art, we need to compare it with current state of the art solutions. For instance, we can compare our solution with the more popular choice of using the mean squared error (MSE) as an error signal. Another term of comparison can be obtained by comparing our approach to other researchers solving similar tasks. For instance, [43] uses convolutional layers (for extracting poses from images) followed by feedforward layers to give a single prediction about the next waypoint in the trajectory. In this work we do not deal with robot vision, thus as the basis of comparison, we have also created an implementation that matches the structure of the network from [43] which

المنارة للاستشارات
www.manaraa.com

follow the convolutional layers.

Thus we have implemented four choices for the controller structure: LSTM with MDN, LSTM with MSE, feedforward network with MDN and feedforward network with MSE. Due to limited space, we will only describe the LSTM with MDN controller. For the other controllers, all the comparable choices had been similar.

The LSTM-MDN robot controller

Let us first justify the intuition behind the use of LSTM and MDN technologies in the implementation of our robot controller.

One of the first insights is that the solution to both manipulation tasks contain a series of individual movements which need to be executed in a specific sequence. Although both tasks can be solved in several different ways, the individual movements in them cannot be randomly exchanged. In order to successfully solve the task, the robot needs to choose and commit to a certain solution. While it is technically possible that this commitment will be encoded in the environment outside the robot, we conjecture that a robot controller that has a memory that can store these commitments will perform better. The requirement of a controller with a memory leads us to the choice of recurrent neural networks, in particular, one of the most widely used model, the LSTM [26].

LSTM can encode the useful information of the past in a single or multiple layers. Each layer accepts as input the output of the previous layer concatenated with the network input $x_t = \{e_t, q_t\}$. Each LSTM layer predicts its output based on its current input and its internal state. In addition, the LSTM updates its internal state at each time-step to be used in the next prediction. The concatenation of outputs of all layers will be used to predict the output of the network y_t. In our controller we are using three LSTM layers with 50 nodes each as shown in Figure 4.3.

www.manaraa.com

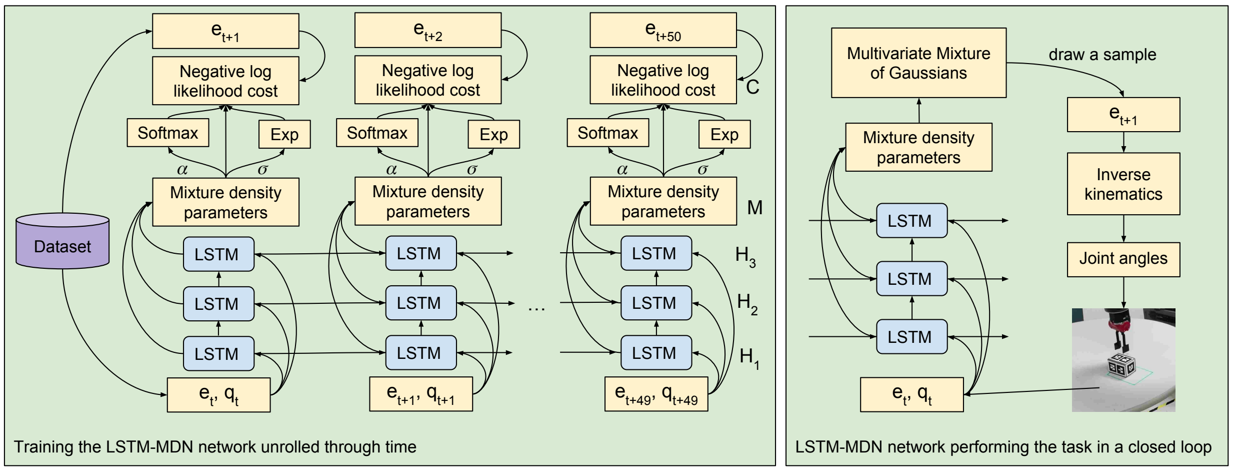

e_{t+1}
e_{t+2}
e_{t+50}
Negative log likelihood cost
C
Softmax
Exp
α
σ
Mixture density parameters
M
Dataset
LSTM
H_3
H_2
H_1
e_t, q_t
e_{t+1}, q_{t+1}
e_{t+49}, q_{t+49}
...
Training the LSTM-MDN network unrolled through time
Multivariate Mixture of Gaussians
draw a sample
Inverse kinematics
Joint angles
LSTM-MDN network performing the task in a closed loop

Figure 4.3: The training and evaluation phase. During the training the LSTM network is unrolled for 50 time-steps. The gripper pose and status (open/close) e_t and the pose of relevant objects q_t at time-step t is used as input and output of the network to calculate and backpropagate the error to update the weights. During the evaluation phase, the mixture density parameters are used to form a mixture of Gaussians and draw a sample from it. The sample is used to control the robot arm.

The second intuition applies to the choice of the output layer and error signal used for the training of the neural network. For both tasks we are considering there can be multiple solutions. For instance, for the push to pose task, the robot might need to push the box in a direction parallel with its diagonal. This can be achieved by either (a) first pushing the shorter side of the box followed by a push on the longer side or (b) the other way around. However, by averaging these two solutions we reach a solution where the grasper would try to push the corner of the box, leading to an unpredictable result. This leads us to the conjecture that a multi-modal error function would perform better than the unimodal MSE. The approach we chose is based on Mixture Density Networks (MDN) [8] which use the output of the network to predict the parameters of a mixture distribution. Once we have the parameters of the mixture distribution, we can draw a sample and use it as the output of the network (which, in our case is the next pose of the gripper). Unlike the model with MSE cost which is deterministic, this approach can model stochastic behaviors to be executed by the robot. The probability density of the next waypoint can be modeled using a linear

المنارة للاستشارات
www.manaraa.com

combination of Gaussian kernel functions

$$p(y|x) = \sum_{i=1}^{m} \alpha_i(x) g_i(y|x) \tag{4.1}$$

where $\alpha_i(x)$ is the mixing coefficient, $g_i(y|x)$ is a multivariate Gaussian, and m is the number of kernels. Note that both the mixing coefficients and the Gaussian kernels are conditioned on the complete history of the inputs till current timestep $x = \{x_1 \ldots x_t\}$. This is because the concatenation of the output of all layers which is used to estimate the mixing coefficients and Gaussian kernels is itself a function of x. The Gaussian kernel is of the form

$$g(y|x) = \frac{1}{(2\pi)^{c/2}\sigma_i(x)} \exp\left\{-\frac{\|y - \mu_i(x)\|^2}{2\sigma_i(x)^2}\right\} \tag{4.2}$$

where the vector $\mu_i(x)$ is the center of ith kernel. We do not calculate the full covariance matrices for each component, since this form of Gaussian mixture model is general enough to approximate any density function [47].

The parameters of the Gaussian kernels $\mu_i(x)$, $\sigma_i(x)$ and mixing coefficients $\alpha_i(x)$ are represented by the layer M in Figure 4.3. To accomplish this, layer M needs to have one neuron for each parameter. Thus layer M will have a width $(c + 2) \times m$, containing $c \times m$ neurons for $\mu_i(x)$, m neurons for $\sigma_i(x)$, and another m neurons for $\alpha_i(x)$. This layer is fully connected to the concatenation of layers H_1, H_2 and H_3.

To satisfy the constraint $\sum_{i=1}^{m} \alpha_i(x) = 1$, the corresponding neurons are passed through a softmax function. The neurons corresponding to the variances $\sigma_i(x)$ are passed through an exponential function and the neurons corresponding to the means $\mu_i(x)$ are used without any further changes.

www.manaraa.com

Finally, we can define the error in terms of negative logarithm likelihood

$$E_{MDN} = -ln \left\{ \sum_{i=1}^{m} \alpha_i(x) g_i(y|x) \right\} \tag{4.3}$$

Training the controller. The network model is implemented in Blocks [72] framework that is built on top of Theano [70]. The network is unrolled for 50 time steps. All the parameters are initialized uniformly between -0.08 to 0.08 following the recommendation by [66]. Stochastic gradient descent with mini-batches of size 10 is used to train the network. RMSProp [71] with initial learning rate of 0.001 and decay of 0.99-0.999 (based on number of examples) is used to divide the gradients by a running average of their recent magnitude. In order to overcome the exploding gradients problem, the gradients are clipped in the range [-1, 1]. We use 80% of the data for training and keep the remaining 20% for validation. We stop the training when the validation error does not change for 20 epochs.

Using reinforcement learning to adapt the policy according to user preferences

As we discussed before, even demonstrations collected from a single user might have examples where the same task is demonstrated in several different styles. If the demonstrations had been collected from multiple users, they will naturally have a mix of various styles of executing tasks. Most of these ways or styles are equivalent from the point of view of performance. A robot trained using LfD will choose randomly at execution time the style of execution to follow.

However, sometimes we need the robot to behave according to a particular style of performing a task so that it matches the preferences of the user. A possible approach is to use reinforcement learning. To accomplish this goal, we use the policy gradient algorithm [67] to push the policy towards our desired style. Let us assume that users can specify some reward r_t at each time-step of

المنارة للاستشارات
www.manaraa.com

trajectory τ that shows how it fits their preferences. For a parameterized policy π_θ (in our method, parameters are the weights of a neural network), the expected return can be defined as

$$J(\theta) = \mathbb{E}\Big[\sum_{t\geq 1} \gamma^t r_t | \pi_\theta\Big] \tag{4.4}$$

where γ is the discount factor. We want to maximize the expected return. According to the REINFORCE algorithm [74], we can estimate the gradient using the formula

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 1} r(\tau) \nabla_\theta \, log \; \pi_\theta \tag{4.5}$$

where $r(\tau)$ is the expected reward for the trajectory τ. However, we can reduce the variance of the gradient by considering only the future reward and discount the distant rewards

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 1} \Big(\sum_{t'\geq t} \gamma^{t'-t} \, r_{t'} \Big) \nabla_\theta \, log \; \pi_\theta \tag{4.6}$$

This means that if a trajectory is based on the preferences of the user (returns high reward quicker), increase the probability of its actions being selected by the policy. Training the neural network using this approach is very similar to the supervised method we explained in the previous part. We use the data gathered from the user and multiply the cost of each waypoint in the trajectory by its expected future reward.

Transferring the controller to the real robot

The last step of the process is to transfer the trained controller to a real robot. We have used a Rethink Robotics Baxter robot for this purpose. Transferring the controller to the physical robot opened several new challenges. As the controller provides only the next pose of the end-effector,

www.manaraa.com

the controller had been augmented with inverse kinematics calculations to calculate the trajectory in the Baxter robot arms' joint space.

Another challenge is that while in the virtual world we had perfect knowledge of the pose of the end-effector and all the objects in the environment, we needed to acquire this information through sensing. In this chapter we do not deal with computer vision: our controller architecture only performs robot arm control. To supplant the missing vision component, we relied on a Microsoft Kinect sensor and objects annotated with markers to track the pose of the objects. One of the problems with this approach is that the robot arm might occlude the view of the sensor. The Kinect sensor was placed close to the table to reduce the chance of occlusion, however, occlusions may still occur if the robot's arm is placed between the object and the Kinect.

Another challenge is the fact that the waypoints generated by the controller are relatively far away, leading to a jerky motion. While generating the trajectory at each time-step, we use interpolation in joint space to fill in the gap between the current waypoint and previous one and make the robot's movement smooth and slow.

Finally, we found that the trajectory described by the controller can not always be executed by the Baxter arm at the same timestep as in the virtual environment. Sometimes it takes longer (a few seconds) for the Baxter to reach the desired next end-effector pose and sometimes it is quite fast (a few milliseconds). Therefore, we use a dynamic execution rate to wait between execution of each waypoint. Concretely, the algorithm waits for .2sec and checks if the difference between the current pose of the gripper and the predicted one is below a certain threshold. If yes, it commands the robot to go to the next waypoint, otherwise it waits in a loop until the end-effector reaches the desired pose or timeout occurs which means that the end-effector cannot reach that pose (either because inverse kinematic fails or a collision occurs).

المنارة للاستشارات
www.manaraa.com

Experimental validation

In the following we describe the results of a series of experiments performed in the virtual and physical environment. Our goals with the experimentation were to verify two hypotheses:

- Hypothesis 1: We conjecture that for the manipulation tasks we consider, the choices of LSTM for the neural network architecture and MDN for the error signal confer advantages over feedforward networks and MSE respectively.
- Hypothesis 2: We conjecture that the proposed architecture combined with policy gradient techniques can learn manipulation policies that perform the task according to preferences of the user.
- Hypothesis 3: We conjecture that the proposed architecture allows us to transfer a controller learned in a virtual environment to the control of a physical robot executing a real-world version of the same task.

Validating hypothesis 1: Comparing network architectures in the virtual world

In order to validate hypothesis 1, in addition to our proposed architecture involving LSTM and MDN we have implemented and trained all the other combinations of the proposed approaches. Thus we trained four different neural network based robot controllers of the following architectures and parametrization:

- FeedForward-MSE: 3 layers of fully connected feedforward network with 100 neurons in each layer and mean squared error as the cost function.

المنارة للاستشارات
www.manaraa.com

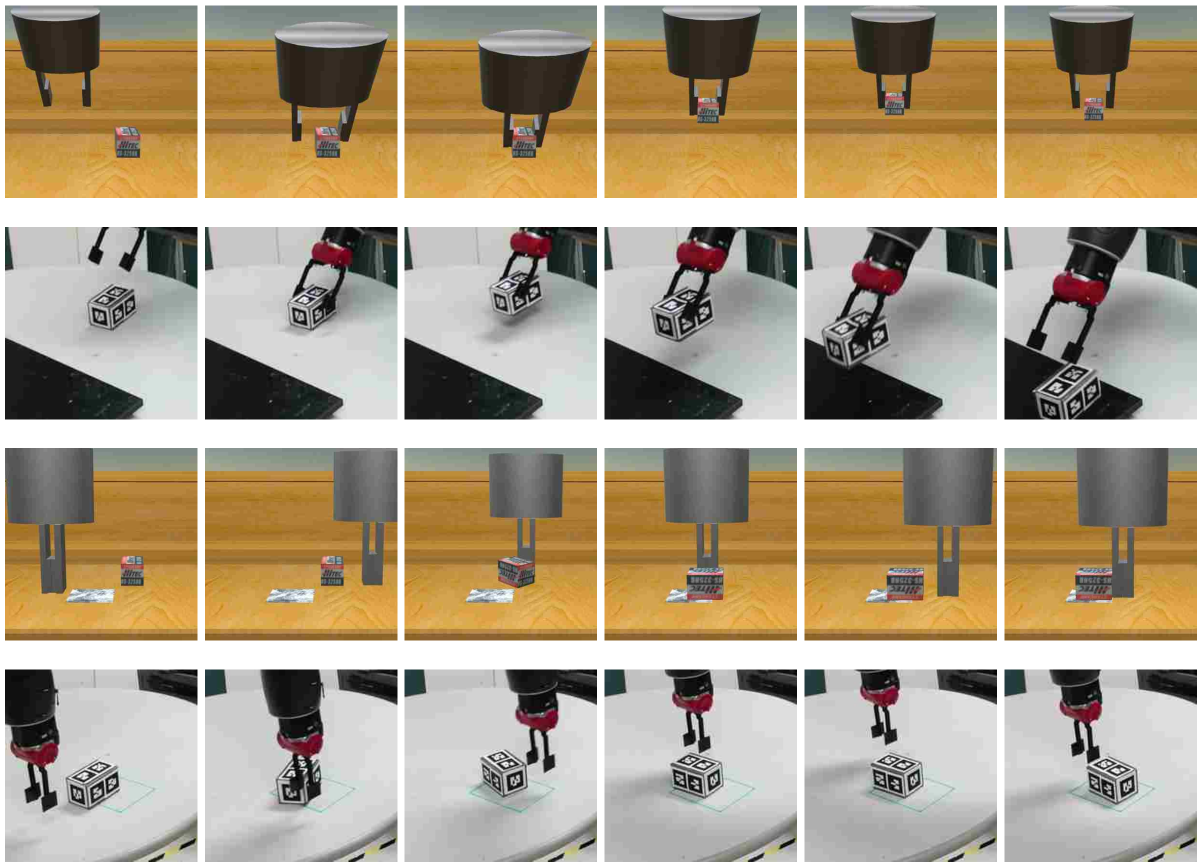


Figure 4.4: A sequence of images showing the autonomous execution of pick and place in simulation (first row), pick and place in real world (second row), pushing in simulation (third row), and pushing in real world (fourth row). The robot is controlled by a mixture density network with 3 layers of LSTM.

- LSTM-MSE: 3 layers of LSTM with 50 memory states in each layer and mean squared error as the cost function.
- FeedForward-MDN: Mixture density network containing 3 fully connected feedforward layers with 100 neurons in each layer. The mixture contains 20 Gaussian kernels.
- LSTM-MDN: Mixture density network containing 3 layers of LSTM with 50 memory states in each layer. The mixture contains 20 Gaussian kernels.

المنارة للاستشارات
www.manaraa.com

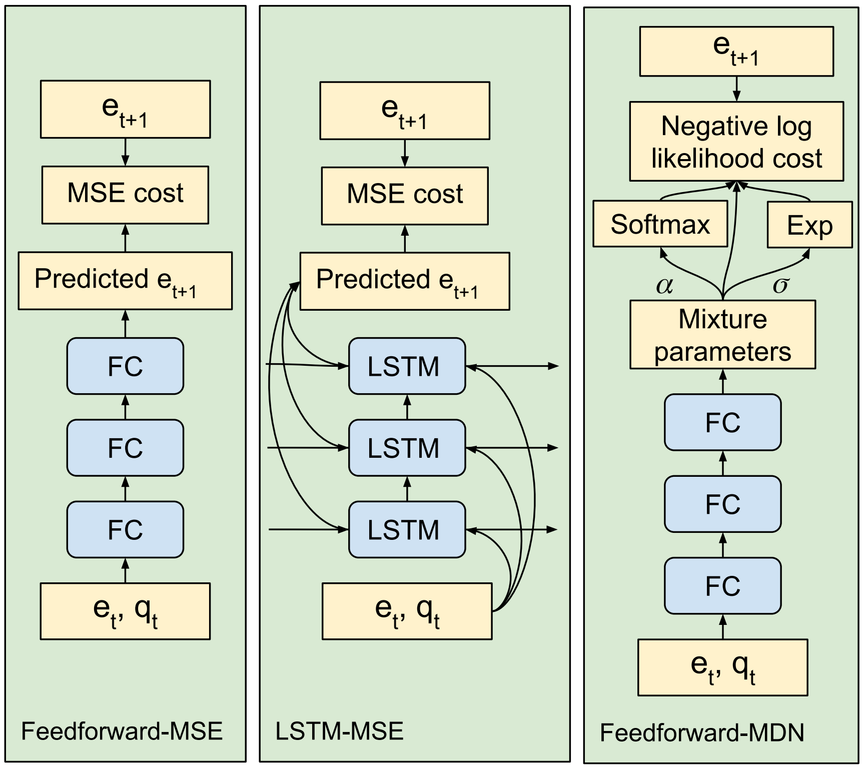


Figure 4.5: Baseline network architectures compared to the LSTM-MDN approach.

The LSTM-MDN network was illustrated in Figure 4.3. The training phase of other baseline approaches are shown in Figure 4.5.

Each network had been separately trained for the pick and place and the push to desired pose respectively, in effect creating 8 different controllers. The resulting controllers had been tested in the virtual environment as follows. The virtual robot had to perform randomly generated tasks 20 times. If it can not complete the task in a limited time (1 minute for the first task and 2 minutes for the second one), we count the try as a failure and reset the position of the box. The performance of the evaluated networks are shown in Table 4.2.

Feedforward-MSE. The feedforward-MSE network with a Markov assumption was not able to

www.manaraa.com

Table 4.2: Performance comparison of different approaches in a virtual environment.

Controller	**Pick and place**	**Push to pose**
Feedfoward-MSE	0%	0%
LSTM-MSE	85%	0%
Feedforward-MDN	95%	15%
LSTM-MDN	**100%**	**95%**

complete the tasks even once. In the pick and place task, it learns to follow the box and sometimes close the gripper, however, it stops there and does not continue towards the shelf. The reason might be that the user usually pauses after closing the gripper to see if the grasp is successful or not. Since the gripper does not move for a few waypoints and then continues to move, the memoryless network fails to determine when the gripper should stop and when it should continue towards the shelf. What this model learns is basically the average of next waypoints without considering the past trajectory.

LSTM-MSE. This network learns a deterministic policy to predict the next waypoint based on the past trajectory by taking the average of next waypoints in the example demonstrations. Therefore, in the pick and place task that there is usually only one solution to the problem, it works relatively fine. However, in the pushing task that usually there are multiple solutions that the user has chosen to perform arbitrarily, the model takes their average which might not be a valid solution.

Feedforward-MDN. This method finds a probability distribution of the next waypoint without considering the past. Therefore, the trajectory does not look smooth as in the case of using LSTM especially when there are different solutions as in the case of pushing task. However, it tries and fails multiple times until it sometimes successfully performs the task.

LSTM-MDN. This method learns to successfully perform the tasks in most of the cases since it does

المنارة للاستشارات
www.manaraa.com

not have the limitations of the other methods, i.e. it predicts the complete probability distribution of the next waypoint considering the past. One interesting observation was that it learns to recover from failure; for instance, when the grasp fails in the pick and place task, the gripper does not continue towards the shelf without the box, instead it tries the grasp one more time. Similarly, in the pushing task, when the force to the box was not enough to put it into the desired location, it tries again. These behaviors are learned by the network based on the performance of the user in similar situations.

Validating hypothesis 2: Using policy gradient to capture user preferences

Let us consider the pick and place task, which requires the robot arm to pick up a box from the table and put it on the shelf. The shelf is the same width as the table, and the task does not require any specific position of where to put the box. In some demonstrations, the users put the box at the same x coordinate on the shelf where it was on the table. In other demonstrations, the box was moved more towards the center of the shelf. All these demonstrations were a correct way to execute the task. We found that the trained controller choose to put the box at the shelf at a more or less random position.

Let us now consider that a user expresses a preference towards putting the box towards the middle of the shelf. In reinforcement learning terms, by assigning for the final action of the task a reward equal with the negative distance d from the center of the shelf on the x coordinate. Then we train the network using policy gradient-based reinforcement learning on the same data we gathered for supervised LfD approach. In the following table, we compare the average distance from the center for the original and RL-refined version over 100 random cases.

المنارة للاستشارات
www.manaraa.com

Controller	**Distance from center on x axis**
LfD only	9.1cm
LfD refined with policy gradient RL	6.07cm

Let us now discuss the results and the limitations of this approach. Clearly, the approach was successful on introducing a new criteria based on which the robot controller chooses its behavior. Although the error is still high, we expect that using more demonstrations will make the robot more accurate as reinforcement learning in general needs more examples to work well compared to supervised learning.

We have also experimented with a number of different preferences a user might express both for the pick and place and the push to pose task. For instance, another reasonable example of a preference would be to avoid certain positions on the table (this would have applications when transferring to a new environment). We did not succeed in converging to a policy that both executes the original task and satisfies these preferences. One reason for this might be that the original demonstrations did not contain any behavior that might serve as the basis of trajectories which might be needed in this case. Our experiments match the current consensus of the deep reinforcement learning community that the RL training for specific criteria is more an art than a science with predictable results. Further investigating this problem is a future work for our group.

Validating hypothesis 3: Comparing performance in the virtual and physical world

Once we have established that the LSTM-MDN approach yields the best performance on both tasks, we proceeded to verify that the LSTM-MDN controller trained on virtual demonstrations can actually perform on the physical robot. To verify this we subjected both the virtual and the physical robots to the same tasks. The sequence of images in the Figure 4.4 shows the controller

المنارة للاستشارات
www.manaraa.com

acting autonomously for the pick and place and pushing to pose tasks in the virtual and physical environments respectively. A video of the same experiments is available online[1].

We have found that indeed, in most cases, the physical robot had been successful on executing both tasks. This is not because the virtual and physical worlds are highly similar. The size of the gripper of the Baxter robot is different from the one in the virtual world. The friction coefficients are very different, and the physics simulation in the virtual world is also of limited accuracy. Even the size and shape of the box used in the physical experiments is not an exact match of the ones in the simulation, and the physical setup suffered from camera calibration problems. Overall, the number of things that can go wrong, is much higher in the physical world.

What we found remarkable is that, in fact, during the experiments many things went wrong or changed from the training data - however, the robot was often able to recover from them. This shows how the model is robust in handling deviation from what it has seen. The network contains different solutions for a case that can apply if others fail. For instance, in the pushing task, to rotate the box, it tries to touch the corner of the box and push it. If the box did not move since the gripper passed it without a touch, it tries again but this time from a point closer to the center of the box. This gives the network some tolerance to slight variations in the size of the box from the simulation to the real world.

After verifying that the successful completion of the task in the physical world is possible, we ran the same series of 20 experiments in the physical world as well. The success rates in the virtual and physical worlds are compared in Table 4.3.

As expected, the success rate was lower in the physical world for both tasks. Some of the reasons behind the lower success rate is obvious: for instance, in the physical world there is an inevitable

[1]`https://youtu.be/9vYlIG2ozaM`

المنارة للاستشارات
www.manaraa.com

Table 4.3: Performance comparison of LSTM-MDN network in the virtual and physical worlds.

Environment	**Pick and place**	**Push to pose**
Virtual world	100%	95%
Physical world	80%	60%

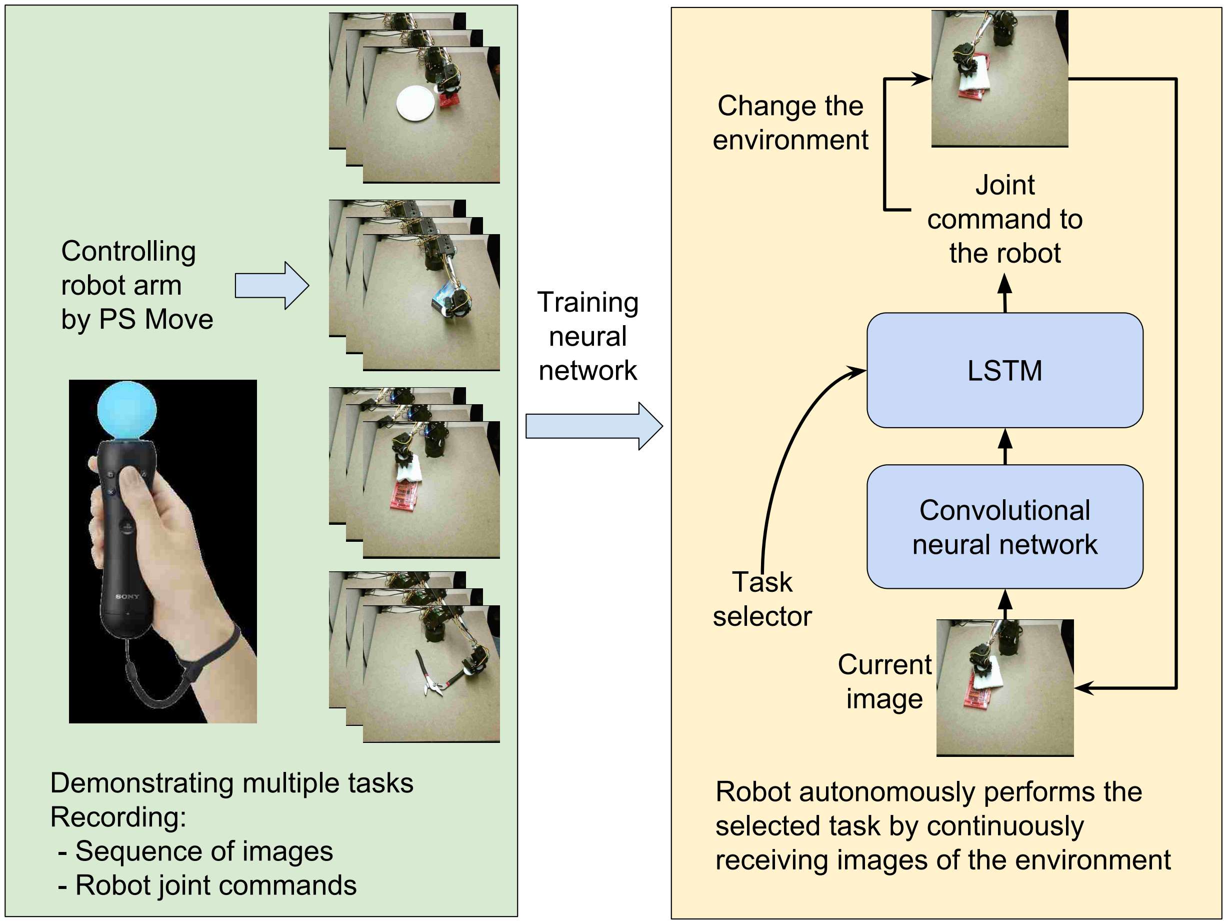
noise in the position of the objects and the end effector. Some of the noise is a consequence of limited sensor accuracy (such as the calibration of the Kinect sensor) and effector performance. Some of the noise is due to the way in which we acquired the positional information through a Kinect sensor: if during the manipulation the robot arm occluded the view of the object to the Kinect sensor, we temporarily lost the ability to track the object.

Another reason for the lower performance in the physical world is due to the differences in the size, shape, physical attributes such as friction, etc. of the gripper and objects between the simulation and real world. For instance, for the push to pose task, the friction between the object and the table determines the way the object moves when pushed. This creates a bigger difference between the virtual and the physical environment compared to the pick and place task, where after a successful grasp the robot is essentially in control of the environment. Thus, the push to pose task shows a stronger decrease in success rate when moving to the physical world.

المنارة للاستشارات
www.manaraa.com

CHAPTER 5: LEARNING MULTIPLE TASKS

In this chapter we discuss a multi-task learning from demonstration mechanism that works using raw images as input. In this approach, a single recurrent neural network can generate robot arm trajectories to perform different manipulation tasks. The task is decided by inputting a task selector one-hot vector to the network. An overview of our approach is illustrated in Figure 5.1.



Controlling robot arm by PS Move
Demonstrating multiple tasks
Recording:
- Sequence of images
- Robot joint commands
Training neural network
Change the environment
Joint command to the robot
LSTM
Convolutional neural network
Task selector
Current image
Robot autonomously performs the selected task by continuously receiving images of the environment

Figure 5.1: Overview of our approach to multi-task learning

www.manaraa.com

Method

Task demonstration and data collection

Different approaches can be considered to enable a human to control a robotic arm. For some robotic arms such as Baxter's there is a zero-gravity mode in which the user can grab the arm and freely move it to demonstrate a task. However, in this approach the user will appear in the recorded images of the scene and might confuse the trained model when he is not present during the evaluation phase. Another approach to control the arm is to use mouse, keyboard, or Xbox controller. However, we did not find these approaches intuitive and convenient for the user. Therefore, we used a Playstation Move or a Leap Motion controller that provides fast tracking of the position and the orientation of user's hand. This information is used to find joint configurations of the robot arm so that its end-effector follows user's hand. This approach can generate natural arm movements for accomplishing a task.

We ask the user to demonstrate some related manipulation tasks with a two finger robotic arm. During the demonstration we record the commands sent to the robot as well as 128×128 RGB images of the scene at a frequency of 33Hz. Then we down-sample the trajectories to reach a frequency of 4Hz. This way we create redundant trajectories with different starting point offsets [55]. For instance, if we have a high frequency trajectory $\{t_1, t_2, \ldots\}$, after down-sampling it by a factor of 8, we will have 8 trajectories $\{t_1, t_9, t_{17}, \ldots\}$, $\{t_2, t_{10}, t_{18}, \ldots\}$, etc. We found this data augmentation technique to be useful in regularizing the network.

www.manaraa.com

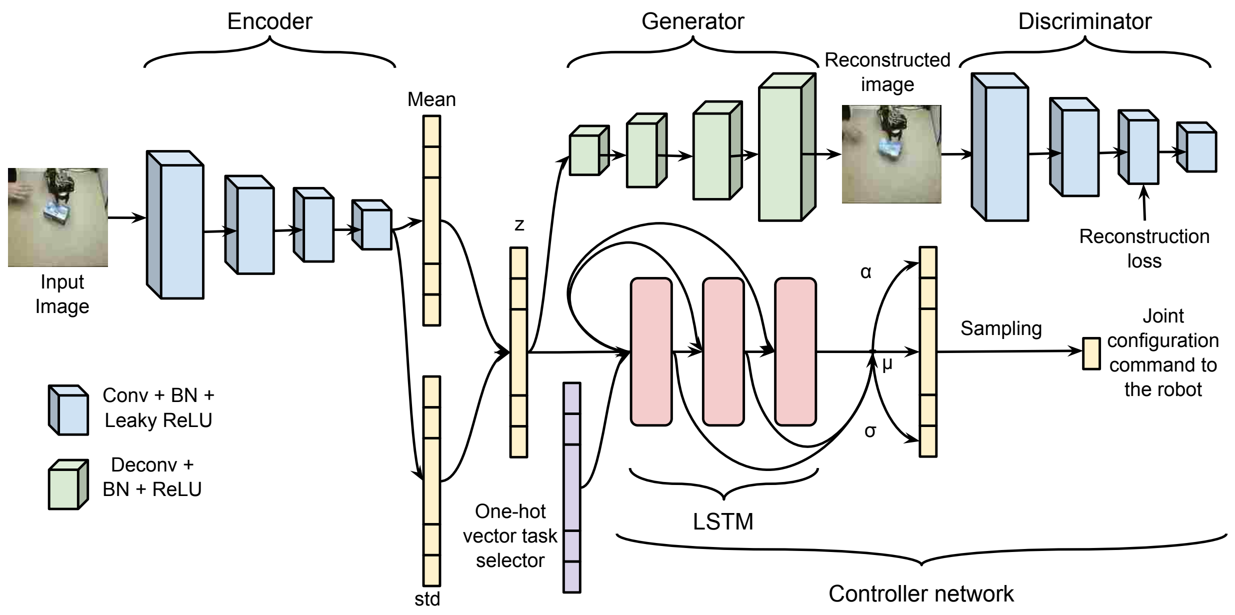


Figure 5.2: Our proposed architecture for multi-task robot manipulation learning. The neural network consists of a controller network that outputs joint commands based on a multi-modal autoregressive estimator and a VAE-GAN autoencoder that reconstructs the input image. The encoder is shared between the VAE-GAN autoencoder and the controller network and extracts some shared features that will be used for two tasks (reconstruction and controlling the robot).

Neural Network Architecture

Our network architecture is illustrated in Figure 5.2. Convolutional layers augmented with batch normalization [27] process the input images and map them to a low dimensional feature representation according to the VAE-GAN approach [41]. VAE-GAN tries to encode input images based on the idea of Variational Autoencoders [35] and reconstruct realistic images based on the idea of Generative Adverserial Networks [23]. In VAE-GAN approach, a discriminator is added to the generator to discriminate the reconstructed images with the real images. However, instead of directly comparing the image pixels that causes uncertainty to appear in the form of blurriness, the extracted features of the real and reconstructed images after the third convolutional layer of the

www.manaraa.com

discriminator are compared together. On the bottom half of the Figure 5.2, we have a controller network where the extracted visual features are combined with a task selector one-hot vector and fed into 3 layers of layer normalized [6] LSTM [26] to generate joint commands to control the robot.

Human demonstrations can be very inconsistent, even for the same task, so a unimodal predictor, such as a Gaussian distribution, will average out dissimilar motions. By using a multi-modal predictor, we can capture all of the modes in the demonstrations without excessive averaging. However, simple multi-modal distributions such as mixtures of Gaussians [8] provide a number of modes that scales linearly with the number of parameters. Therefore, by using a multi-modal autoregressive estimator similar to Neural Autoregressive Distribution Estimator (NADE) [40], we increase the number of modes that the model can represent exponentially with the number of steps of the autoregressive model. While autoregressive estimators usually discretize the output, we use mixture of Gaussians to predict the entire probability distribution of the output, providing a rich and expressive class of distributions.

A closer look at the architecture shows that we have a VAE-GAN autoencoder that shares its encoder with the visual feature extractor of a controller network that sends commands to the robot. The encoder tries to fully reconstruct the images while the controller network will try to focus on some relevant features from the image such as the pose of the gripper and relevant objects. This competition/collaboration between these two networks will result in a more regularized visual feature extractor. This idea is similar to the semi-supervised learning with generative models [36] where they use a generative model via the VAE decoder and discriminative training via the action branch to improve sample efficiency. However, in contrast to this work, we observe an improvement in generalization simply from including the reconstruction objective, without including any additional unlabeled data.

www.manaraa.com

Note that the extracted features from the encoder are in the form of a probability distribution that is encouraged to be close to the unit Gaussian by a KL-divergance penalty in the loss function. The noise in the LSTM input caused by sampling from the encoded latent features helps to regularize the LSTM. In addition, we use dropout [64] with a probability of 0.5 to further avoid overfitting.

Training the Network

The error signal that is used to train the network using back-propagation is based on the idea of Mixture Density Networks (MDN) [8]. In this approach, the output of the LSTM network is used to predict the parameters of a multi-modal mixture distribution. However, we do not predict all the outputs (joint configurations) at the same time-step of LSTM. Instead, we factor the J-dimensional distribution of joint configurations y(x) into a product of one-dimensional distributions, in this order: base, shoulder, elbow, ..., and gripper. The probability distribution in this approach is modeled using a linear combination of Gaussian kernel functions of the form

$$p(y|x) = \sum_{i=1}^{m} \alpha_i(x) g_i(y|x) \tag{5.1}$$

in which $\alpha_i(x)$ is the mixing coefficient, $g_i(y|x)$ is a multivariate Gaussian, and m is the number of kernels. At each time-step, the output is $y = y_t^j$, the current joint to be predicted and input is $x = \{x_1^1 \dots x_t^{<j}\}$, the encoded history of observations and predictions of the network before predicting the current joint. The Gaussian kernel is of the form

$$g(y|x) = \frac{1}{(2\pi)^{c/2}\sigma_i(x)} \exp\left\{-\frac{\|y - \mu_i(x)\|^2}{2\sigma_i(x)^2}\right\} \tag{5.2}$$

where the vector $\mu_i(x)$ is the center of ith kernel. We do not calculate the full covariance matrices for each component, since this form of Gaussian mixture model is general enough to approximate

www.manaraa.com

any density function [47].

In the network architecture, we use skip connections from the input to all LSTM layers [25]. Then the output of all LSTM layers are concatenated together and then fully connected to another layer with width $3 \times m$. This layer contains m neurons for $\mu_i(x)$, m neurons for $\sigma_i(x)$, and another m neurons for $\alpha_i(x)$. To satisfy the constraint $\sum_{i=1}^{m} \alpha_i(x) = 1$, the corresponding neurons are passed through a softmax function. The neurons corresponding to the variances $\sigma_i(x)$ are passed through an exponential function and the neurons corresponding to the means $\mu_i(x)$ are used without any further changes. Finally, we can define the error in terms of negative logarithm likelihood

$$E = -ln \left\{ \sum_{i=1}^{m} \alpha_i(x) g_i(y|x) \right\} \tag{5.3}$$

Except the latent space size that is set to 256, other parameters of the autoencoder are set and initialized according to the original paper [41]. All other parameters including LSTM parameters are initialized uniformly between -0.08 to 0.08 following the recommendation by [66]. Each LSTM layer has 100 memory cells and is connected to a mixture of Gaussians with 50 components. We first unroll and train the network using sequences of 5 time-steps and batch size of 100 examples. Then we fine-tune the LSTM layers using sequences of 50 time-steps and mini-batches of size 128. In the fist phase of training, Adam optimizer [37] is used while for fine tuning LSTM layers, RMSProp [71] with initial learning rate of 0.005 and decay of 0.999. In order to overcome the exploding gradients problem, the gradients are clipped in the range [-1, 1]. During the fine-tuning, the learning rate is decreased by a factor of 2 in every 100 epochs.

المنارة للاستشارات
www.manaraa.com

Executing the Policy at Test Time

During the test time, the trained neural network controller generates robot joint commands in a loop by observing the environment. The LSTM predicts each joint one by one and when the predictions of all the joints are available, the robot takes an action, and another image is recorded and fed into the controller. As we mentioned before, the user occasionally makes mistakes while demonstrating the task. However, it is desired that the robot does not repeat those mistakes. We can reduce the rate of mistakes by introducing some bias towards higher probability areas of the distribution while sampling from the probability distribution [25]. While sampling, we use the new mixing coefficient

$$\alpha_i(x) = \frac{\exp(\alpha_i(x)(1+b))}{\exp(\sum_{i=1}^{m} \alpha_i(x)(1+b))} \tag{5.4}$$

and standard deviation

$$\sigma_i(x) = \exp(\sigma_i(x)(1+b)) \tag{5.5}$$

where bias parameter b is a real number between 0 to 10. When $b = 0$, there is no bias while $b = 10$ introduces the maximum bias where only a point with maximum probability is chosen. We found $b = 1$ to work well in our experiments.

Why this Architecture?

Why predicting the entire probability distribution? In many of the robotics tasks, there exist more than one solution to a problem. Interestingly, humans tend to perform the tasks in different ways. Even one user might take different approaches to solve a problem in each attempt. Therefore our model should be capable of dealing with datasets containing more than one possible output given the same input. One approach would be to predict a deterministic joint command and use mean squared error to minimize the error between the predicted command and the command demon-

www.manaraa.com

strated by the user. In this approach, if there are multiple solutions demonstrated by the user, the network will learn to predict the average of these commands. However, this behaviour might result in completely wrong behaviours. For instance, assume that the end-effector is supposed to rotate around an object without colliding with it to reach the other side of it. The user sometimes chose to rotate clockwise and sometimes counterclockwise. If our model learns an average of these two trajectories, the end-effector will go straight towards the object and collide with it instead of rotating around it either way. A solution to this problem is to predict the entire probability distribution of the next action and choose a sample from that distribution.

Why use recurrent neural networks? Another consideration while designing the network is that whether the input provides all the necessary information for choosing a motor command. In simple tasks this might be the case, however, some tasks need history of the previous waypoints in the trajectory in order to predict the next command. For instance, assume that the task is to pick up a pen from a pen holder and put it on the desk if the pen is currently inside the pen holder, otherwise pick up the pen from the desk and put it inside the pen holder. Consider an image in which the pen is gripped and is in the middle of the way between the desk and the pen holder. Is this image enough for telling the network whether to continue towards the pen holder or the desk? No. The network needs to know where the pen has been before in order to decide the next action. Another example can be when the human demonstrator needs to stop for a couple of time-steps to make sure the gripper is in the right place before grasping an object. The model needs to remember the number of time-steps that it has been waiting before continuing to move. This is the reason we use LSTM recurrent neural network that can store relevant information to be used later.

Why autoregressive density estimator? By using this density estimator, we condition the prediction of each joint on the prediction of previous joints. To have an understanding of why this is important, imagine a situation in which an object is about to be grasped. The network needs to predict whether the gripper should be closed or not in the next time-step. If we do not condition the prediction of

المنارة للاستشارات
www.manaraa.com

gripper on the prediction of other joints in the current time-step, the gripper might be closed before the end-effector is in a good grasping angle. Therefore, the grasp will fail. This idea of modeling the joint probability distribution of outputs by casting it as a product of conditional distributions is used in Pixel RNNs [50], Neural Autoregressive Distribution Estimator (NADE) [40], and fully visible neural networks [49, 7].

Experimental validation

Manipulation Tasks

In all of the following tasks the objects are placed in a random position and orientation within the reachability range of the robot arm. One of the challenges is that all the objects might get partially or completely occluded by the robot arm. In addition, because of the difficulty of the tasks and also the teleoperation method, the user often makes mistakes and tries again.

Task 1. In this task, the robot picks up a small bubble wrap and puts it into a small plate. This task is very challenging since the robot needs to be very accurate while picking up the thin and deformable bubble wrap that often gets completely occluded by the arm during the grasp. If the bubble wrap is placed inside the plate, we count it as a success.

Task 2. In this task, the robot needs to push a rounded plate to a certain area on the left side of its workspace. This task is challenging because the robot needs to accurately detect the position of the plate and push a point that moves the plate with the desired angle. In addition, if the plate ended up moving to an unexpected direction, the arm needs to push from a completely different contact point to fix the problem. If the plate is placed within an area with the radius of 3cm larger than the radius of the plate, we count it a success.

المنارة للاستشارات
www.manaraa.com

Task 3. In this task, the robot pushes a large box, which is too large to be grasped, and places it close to its base with a certain position and orientation. This task is challenging because the robot needs adjust the orientation of the box within a 20° error range and the position of the box within an area that is 2cm wider than the box in each direction. So, if for instance the robot pushes the box further to the right such that it exits the mentioned area, it needs to circle around the box without colliding with it to push it from the other side.

Task 4. In this task, water pump pliers are placed on the desk in the open position. The robot needs to close the pliers and orient them parallel to the table borders. In this task the convolutional layers need to detect the thin handles of the pliers so that LSTM can decide where to push to accomplish the task. The pliers needs to be completely closed while 20° of error is acceptable for its final orientation. Note that in this task the initial orientation of the pliers is in a way that the handles are closer than its head to the base of the robot arm.

Task 5. In this task, the robot needs to pick up a towel and rub a small screwdriver box to clean it. Although cleaning is not really important here, the motion needs to be as demonstrated. Therefore, if the robot successfully picks up the towel, rubs at least half of the screwdriver box, and places the towel back on the table, we count it as a success.

We demonstrate each task for 3 hours, which is equivalent to 909, 495, 431, 428, 398 times completion of each task for tasks 1-5, respectively. We use 80% of this data for training and keep the remaining 20% for validation.

Compared Methods

In this part we compare different different variations of the proposed method to see which one works best. The compared approaches are as follows:

www.manaraa.com

Single-task. The network architecture is as explained in Figure 5.2. However, we train it on the data of a single task. This means that the one-hot task selector vector will not be used. In addition, all the joints are predicted at the same time, therefore, it is not an autoregressive estimator.

Multi-task. This method is the same as single-task method, however, we train it on the data of all tasks and then we use the one-hot task selector vector to decide which task should be performed.

Multi-task autoregressive (no reconstruction). this approach utilized the autoregressive estimator and is trained on the data of all tasks. However, we exclude the VAE/GAN that adds the reconstruction to the error signal. This way we can see if the reconstruction part of the network really helps.

Multi-task autoregressive. this is the main approach that contains autoregressive estimator and reconstruction error, and also is trained on the data of all tasks.

Results

First we show how the autoencoder can reconstruct the input images in Figure 5.3. This shows that all the objects and the arm itself are captured and encoded quite well in most cases. Therefore, the LSTM has useful information to generate a trajectory to accomplish the task. Videos of example demonstrations and also autonomous performance of different tasks by the robot can be watched online[1]. Note that to get an accurate impression of the results, it is strongly recommended that the reader watches the accompanying videos.

In order to quantitatively evaluate the performance of our method, we allow the robot to try each task 25 times. If it cannot accomplish the task in a limited time (45 seconds for task 1, 60 seconds

[1]`https://www.youtube.com/playlist?list=PL9FNn4TWSzQ418GWZsK1NWmNJRe_Q0ipg`

المنارة للاستشارات
www.manaraa.com

for tasks 2-4, and 75 seconds for task 5), we count the try as a failure place the objects in a new random pose and repeat the experiment. Note that we do not stop the controller while we are resetting the experiment since this has also been the case during the demonstrations. It is interesting that the robot learned to go to the default state when it finishes the task since this was the preference of the user while demonstrating. Table 5.1 shows the difference in performance of compared methods.

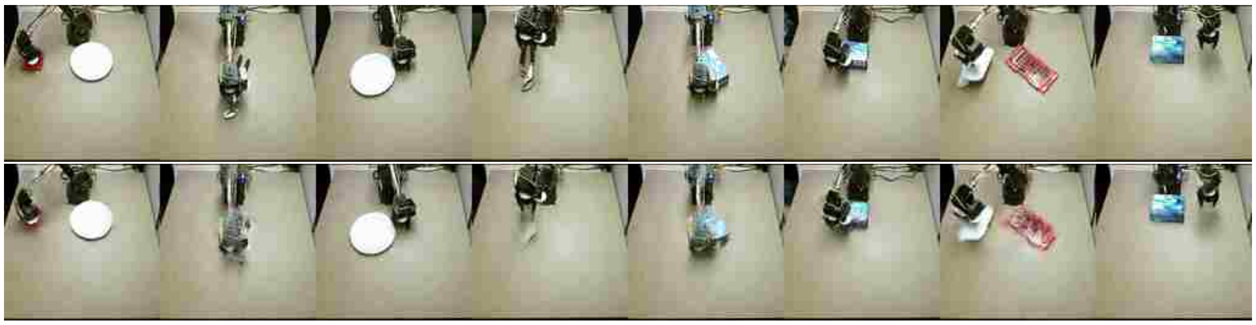


Figure 5.3: Original input images to the network are shown on the top row. For each original image, the corresponding reconstructed image by the autoencoder is shown in the bottom row.

Table 5.1: Performance comparison of different methods. The numbers are the percentile rate of successfully accomplishing the tasks.

Method	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Task 5**
Single-task	36%	16%	44%	16%	8%
Multi-task	16%	20%	52%	64%	20%
Multi-task autoregressive (no reconstruction)	12%	72%	56%	48%	16%
Multi-task autoregressive	**76%**	**80%**	**88%**	**76%**	**88%**

Now we describe and analyze the performance of each method:

Single-task. Training this network is difficult since it overfits very easily. Our attempts to avoid or

المنارة للاستشارات
www.manaraa.com

delay overfitting by increasing the dropout ratio or making the network smaller did not improve the results. The proposed model is very powerful and it does not have any assumption about the task or the shape of objects that are involved in each task. This is good since we can train the model on a wide variety of tasks. However, we need large number of demonstrations to successfully learn a single task. The controller trained using this approach generates some movements that are often similar to the demonstrations. However, they are not accurate enough to finish the task most of the time. Sometimes abrupt movements send the objects to an unseen configuration or out of the reachability range of the robot.

Multi-task. This model works much better than the previous model since there is more data that leads to more stable training. The common patterns among different tasks can be well captured since there is enough data from different tasks. For instance, consider a time when the gripper is close to an object (bubble wrap or paper towel) and the model needs to decide if the gripper should be closed or not in the next time-step. There is large variance in the data of one task due to imperfect demonstrations and model. However, when grasping is an element of two different tasks, the data for this pattern is doubled which leads to a better prediction.

Multi-task autoregressive (no reconstruction). The vision part of this model is not well trained especially in the tasks where the objects are smaller or occluded more often. This is probably because the training examples are not enough to train the visual feature extractor of the controller. In the tasks where the objects are larger and vision is the easy part (e.g., task 3), the robot achieves acceptable results.

Multi-task autoregressive. This model achieves the best results compared to the rest. Our observation was that this model generates very smooth trajectories that take a reasonable path in different situations. Interestingly, this model has also learned to fix its own mistakes most of the time. Take a look at Figure 5.4 for an example of mistake-fixing behaviour. To understand why this model

المنارة للاستشارات
www.manaraa.com

works better consider the following example. When the gripper is close to an object that needs to be grabbed, the event for closing the gripper might be triggered. In this model the uncertainty is diminished since we delay the prediction of the gripper to the future when all other joints are already known. Therefore, the model knows better if the gripper will have a good contact with the object for a more stable grasp or not. Hence, the chances of a more successful grasp increases.

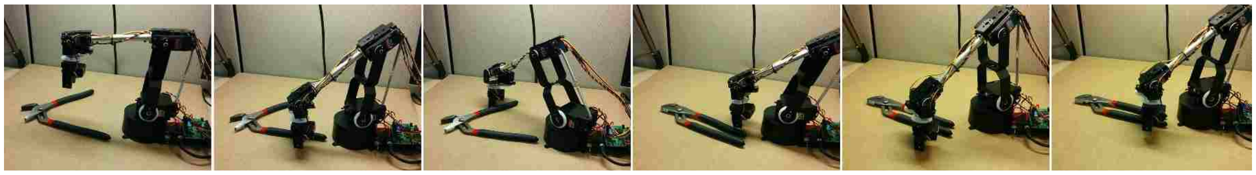


Figure 5.4: The robot makes mistakes but fixes them. Sequence of images from left to right shows the performance of the robot. The task here is to close and orient the water pump pliers. When the pliers are pushed more than expected, the robot turns around and pushes the other side to move the pliers back to the desired orientation.

www.manaraa.com

CHAPTER 6: CONCLUSIONS

In this dissertation, we discussed methods to learn from user demonstrations with the goal of making robots autonomous. First, we showed that if we consider only a subset of simple tasks that are performed with respect to person's head, we can learn using very few demonstrations. In the second part, we addressed this problem: can we learn a task-agnostic controller from user demonstrations if we have perfect knowledge about the environment? We proposed to use a neural network architecture for this learning task. We evaluated this approach in simulation and also in real world where the pose of each object is known. We also showed how to push the generated trajectory towards desired user preferences using reinforcement learning. In the third part, we extended the discussed architecture to a more general one where the input of the neural network is images of the environment instead of object poses. We showed that this end-to-end training approach works well on complex manipulation tasks. We also showed that a single network can learn multiple tasks at the same time and it improves the success rate. This improvement is achieved because most of the manipulation tasks have common building blocks such as grasping and pushing. In addition, we showed that conditioning the prediction of each joint on the previously predicted joint is very important and effective.

By providing sufficient demonstrations to the proposed architecture, we could learn complicated tasks. One direction for the future work can be to find the limitations of this approach, i.e. which tasks the robot does not learn. Another direction can be to gather demonstrations from multiple users and learn to perform the task according to the preferences of each user. In this scenario, the preferences of the user should be provided to the network in some way such as using a feature vector.

المنارة للاستشارات
www.manaraa.com

LIST OF REFERENCES

[1] Pieter Abbeel, Adam Coates, and Andrew Y Ng. “Autonomous helicopter aerobatics through apprenticeship learning”. In: *International Journal of Robotics Research (IJRR)* (2010).

[2] Pooya Abolghasemi et al. “A real-time technique for positioning a wheelchair-mounted robotic arm for household manipulation tasks”. In: *AAAI Workshop on artificial intelligence applied to assistive technologies and smart environments (ATSE-16)*. 2016.

[3] Pooya Abolghasemi et al. “Real-time placement of a wheelchair-mounted robotic arm”. In: *Robot and Human Interactive Communication (RO-MAN), 2016 25th IEEE International Symposium on*. IEEE. 2016, pp. 1032–1037.

[4] Pulkit Agrawal et al. “Learning to poke by poking: Experiential learning of intuitive physics”. In: *arXiv preprint arXiv:1606.07419* (2016).

[5] Brenna D Argall et al. “A survey of robot learning from demonstration”. In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).

[7] Yoshua Bengio and Samy Bengio. “Modeling High-Dimensional Discrete Data with Multi-Layer Neural Networks.” In: *Advances in neural information processing systems (NIPS)*. Vol. 99. 1999, pp. 400–406.

[8] Christopher M Bishop. “Mixture density networks”. In: *Technical Report* (1994).

[9] Mario Bollini, Jennifer Barry, and Daniela Rus. “Bakebot: Baking cookies with the PR2”. In: *IROS PR2 workshop: results, challenges and lessons learned in advancing robots with a common platform*. 2011.

المنارة للاستشارات
www.manaraa.com

[10] Sylvain Calinon, Tohid Alizadeh, and Darwin G Caldwell. “On improving the extrapolation capability of task-parameterized movement models”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013, pp. 610–616.

[11] Sylvain Calinon, Florent Guenter, and Aude Billard. “On learning, representing, and generalizing a task in a humanoid robot”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 37.2 (2007), pp. 286–298.

[12] Sylvain Calinon et al. “Handling of multiple constraints and motion alternatives in a robot programming by demonstration framework.” In: *IEEE International Conference on Humanoid Robots (Humanoids)*. Citeseer. 2009, pp. 582–588.

[13] Sylvain Calinon et al. “Learning and reproduction of gestures by imitation”. In: *IEEE Robotics & Automation Magazine* 17.2 (2010), pp. 44–54.

[14] Rich Caruana. “Learning Many Related Tasks at the Same Time with Backpropagation.” In: *Advances in neural information processing systems* (1995), pp. 657–664.

[15] Kyunghyun Cho et al. “On the properties of neural machine translation: Encoder-decoder approaches”. In: *arXiv preprint arXiv:1409.1259* (2014).

[16] Christopher Crick et al. “Human and robot perception in large-scale learning from demonstration”. In: *International conference on Human-robot interaction*. ACM. 2011, pp. 339–346.

[17] Marc Peter Deisenroth et al. “Multi-task policy search for robotics”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 3876–3881.

[18] Coline Devin et al. “Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer”. In: *arXiv preprint arXiv:1609.07088* (2016).

المنارة للاستشارات
www.manaraa.com

[19] Jeff Donahue et al. “Long-term recurrent convolutional networks for visual recognition and description”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[20] Felix Endres, Jeff Trinkle, and Wolfram Burgard. “Learning the dynamics of doors for robotic manipulation”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013, pp. 3543–3549.

[21] Gabriele Fanelli et al. “Random Forests for Real Time 3D Face Analysis”. In: *International Journal of Computer Vision* 101.3 (2013), pp. 437–458.

[22] Maxwell Forbes et al. “Robot programming by demonstration with crowdsourced action fixes”. In: *Second AAAI Conference on Human Computation and Crowdsourcing*. 2014.

[23] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[24] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech recognition with deep recurrent neural networks”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2013, pp. 6645–6649.

[25] Alex Graves. “Generating sequences with recurrent neural networks”. In: *arXiv preprint arXiv:1308.0850* (2013).

[26] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[27] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[28] J. Xu et al. “A Sequence Learning Model with Recurrent Neural Networks for Taxi Demand Prediction”. In: *IEEE Conference on Local Computer Networks Networks (LCN)*. 2017.

المنارة للاستشارات
www.manaraa.com

[29] Amirhossein Jabalameli and Aman Behal. “A constrained linear approach to identify a multi-timescale adaptive threshold neuronal model”. In: *IEEE International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. IEEE. 2015, pp. 1–6.

[30] Ashesh Jain et al. “Learning Trajectory Preferences for Manipulators via Iterative Improvement”. In: *Neural Information Processing Systems (NIPS)*. 2013, pp. 575–583.

[31] Navid Kardan and Kenneth O Stanley. “Fitted Learning: Models with Awareness of their Limits”. In: *arXiv preprint arXiv:1609.02226* (2016).

[32] Navid Kardan and Kenneth O Stanley. “Mitigating fooling with competitive overcomplete output layer neural networks”. In: *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE. 2017, pp. 518–525.

[33] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. “Visualizing and understanding recurrent networks”. In: *arXiv preprint arXiv:1506.02078* (2015).

[34] Ben Kehoe et al. “Cloud-based robot grasping with the Google object recognition engine”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2013, pp. 4263–4270.

[35] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[36] Diederik P Kingma et al. “Semi-supervised learning with deep generative models”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 3581–3589.

[37] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[38] Jens Kober, J Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

المنارة للاستشارات
www.manaraa.com

[39] George Konidaris and Andrew Barto. “Autonomous shaping: Knowledge transfer in reinforcement learning”. In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 489–496.

[40] Hugo Larochelle and Iain Murray. “The Neural Autoregressive Distribution Estimator.” In: *AISTATS*. Vol. 1. 2011, p. 2.

[41] Anders Boesen Lindbo Larsen et al. “Autoencoding beyond pixels using a learned similarity metric”. In: *International Conference on Machine Learning*. 2016, pp. 1558–1566.

[42] Alex X. Lee et al. “A non-rigid point and normal registration algorithm with applications to learning from demonstrations”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 935–942.

[43] Sergey Levine et al. “End-to-end training of deep visuomotor policies”. In: *Journal of Machine Learning Research* 17.39 (2016), pp. 1–40.

[44] Sergey Levine et al. “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection”. In: *arXiv preprint arXiv:1603.02199* (2016).

[45] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[46] Michael G Madden and Tom Howley. “Transfer of experience between reinforcement learning environments with progressive difficulty”. In: *Artificial Intelligence Review* 21.3 (2004), pp. 375–398.

[47] Geoffrey J McLachlan and Kaye E Basford. “Mixture models. Inference and applications to clustering”. In: *Statistics: Textbooks and Monographs, New York: Dekker* 1 (1988).

[48] Stephen Miller et al. “A geometric approach to robotic laundry folding”. In: *The International Journal of Robotics Research* 31.2 (2012), pp. 249–267.

المنارة للاستشارات

www.manaraa.com

[49] Radford M Neal. “Connectionist learning of belief networks”. In: *Artificial intelligence* 56.1 (1992), pp. 71–113.

[50] Aaron Van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. “Pixel Recurrent Neural Networks”. In: *International Conference on Machine Learning*. 2016, pp. 1747–1756.

[51] Peter Pastor et al. “Learning and generalization of motor skills by learning from demonstration”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2009, pp. 763–768.

[52] Jan Peters and Stefan Schaal. “Reinforcement learning of motor skills with policy gradients”. In: *Neural networks* 21.4 (2008), pp. 682–697.

[53] Lerrel Pinto and Abhinav Gupta. “Learning to Push by Grasping: Using multiple tasks for effective learning”. In: *arXiv preprint arXiv:1609.09025* (2016).

[54] Lerrel Pinto and Abhinav Gupta. “Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 763–768.

[55] Rouhollah Rahmatizadeh et al. “Learning real manipulation tasks from virtual demonstrations using LSTM”. In: *arXiv preprint arXiv:1603.03833* (2016).

[56] Rouhollah Rahmatizadeh et al. “Trajectory Adaptation of Robot Arms for Head-Pose Dependent Assistive Tasks.” In: *FLAIRS conference*. 2016, pp. 410–413.

[57] Rouhollah Rahmatizadeh et al. “Vision-Based Multi-Task Manipulation for Inexpensive Robots Using End-To-End Learning from Demonstration”. In: *arXiv preprint arXiv:1707.02920* (2017).

[58] Jan Ramon, Kurt Driessens, and Tom Croonenborghs. “Transfer learning in reinforcement learning problems through partial policy recycling”. In: *European Conference on Machine Learning*. Springer. 2007, pp. 699–707.

المنارة للاستشارات
www.manaraa.com

[59] David P Reichert and Thomas Serre. “Neuronal synchrony in complex-valued deep networks”. In: *arXiv preprint arXiv:1312.6115* (2013).

[60] Benjamin Reiner et al. “LAT: A simple Learning from Demonstration method”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2014, pp. 4436–4441.

[61] Leonel Rozo, Pablo Jiménez, and Carme Torras. “A robot learning from demonstration framework to perform force-based manipulation tasks”. In: *Intelligent Service Robotics* 6.1 (2013), pp. 33–51.

[62] Hiroaki Sakoe and Seibi Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE Transactions on Acoustics, Speech and Signal Processing* 26.1 (1978), pp. 43–49.

[63] John Schulman et al. “Learning from demonstrations through the use of non-rigid registration”. In: *International Symposium on Robotics Research (ISRR)*. 2013.

[64] Nitish Srivastava et al. “Dropout: A simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[65] Jaeyong Sung, Seok Hyun Jin, and Ashutosh Saxena. “Robobarista: Object Part-based Transfer of Manipulation Trajectories from Crowd-sourcing in 3D Pointclouds”. In: *International Symposium on Robotics Research (ISRR)*. 2015.

[66] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems (NIPS)*. 2014, pp. 3104–3112.

[67] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

www.manaraa.com

[68] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *arXiv preprint arXiv: 1312.6199* (2013).

[69] Matthew E Taylor and Peter Stone. “Transfer learning for reinforcement learning domains: A survey”. In: *Journal of Machine Learning Research* 10.Jul (2009), pp. 1633–1685.

[70] Theano Development Team. “Theano: A Python framework for fast computation of mathematical expressions”. In: *arXiv e-prints* abs/1605.02688 (May 2016). URL: `http://arxiv.org/abs/1605.02688`.

[71] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural Networks for Machine Learning* 4 (2012).

[72] Bart Van Merriënboer et al. “Blocks and fuel: Frameworks for deep learning”. In: *arXiv preprint arXiv:1506.00619* (2015).

[73] Oriol Vinyals et al. “Show and tell: A neural image caption generator”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[74] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[75] J. Xu et al. “Real-Time Prediction of Taxi Demand Using Recurrent Neural Networks”. In: *IEEE Transactions on Intelligent Transportation Systems* PP.99 (2017), pp. 1–10.

[76] Gu Ye and Ron Alterovitz. “Demonstration-guided motion planning”. In: *International Symposium on Robotics Research (ISRR)*. Vol. 5. 2011.

[77] Niloofar Yousefi, Michael Georgiopoulos, and Georgios C Anagnostopoulos. “Multi-Task Learning with Group-Specific Feature Space Sharing”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2015, pp. 120–136.

المنارة للاستشارات
www.manaraa.com