LEARNING TO MANIPULATE NOVEL OBJECTS FOR ASSISTIVE ROBOTS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Jaeyong Sung

May 2017

LEARNING TO MANIPULATE NOVEL OBJECTS FOR ASSISTIVE ROBOTS

Jaeyong Sung, Ph.D.

Cornell University 2017

The ability to reason about different modalities of information, for the purpose of physical interaction with objects, is a critical skill for assistive robots. For a robot to be able to assist us in our daily lives, it is not feasible to train each robot for a large number of tasks with all instances of objects that exist in human environments. Robots will have to generalize their skills by jointly reasoning with various sensor modalities such as vision, language and haptic feedback. This is an extremely challenging problem because each modality has intrinsically different statistical properties. Moreover, even with expert knowledge, manually designing joint features between such disparate modalities is difficult.

In this dissertation, we focus on developing learning algorithms for robots that model tasks involving interactions with various objects in unstructured human environments — especially on novel objects and scenarios that involve sequences of complicated manipulation. To this end, we develop algorithms that learn shared representations of multimodal data and model full sequences of complex motions. We demonstrate our approach on several different applications: understanding human activities in unstructured environment, synthesizing manipulation sequences for under-specified tasks, manipulating novel appliances, and manipulating objects with haptic feedback.

BIOGRAPHICAL SKETCH

Jaeyong Sung was born in Seoul, South Korea. He received Bachelor of Science in Computer Science with a minor in Applied Mathematics from Cornell University in 2011. During undergraduate studies, he became fascinated by the field of robot learning while building a vision-based autonomous mobile robot. After graduating, he went on to pursue software engineering, building cloud computing solutions at Amazon Web Services. He returned to Cornell to pursue Ph.D. with the desire of contributing to advancing machine intelligence for robots.

ACKNOWLEDGEMENTS

I would like to thank my committee members — Ashutosh Saxena, J. Kenneth Salisbury, Bart Selman, François Guimbretière, Steve Marschner — for all their advice and guidance throughout the years. I am especially grateful to my advisor, Ashutosh Saxena, for his support and guidance. Since the time I met him as an undergraduate researcher, he has taught me how good research in robotics and machine learning should be done. I am also very thankful to J. Kenneth Salisbury. Since I have moved to Stanford University to carry on my research, he has been very supportive and has taught me to always look at problems from many different perspectives. I would also like thank Bart Selman, who has given lots of guidance and insightful comments throughout the years.

I am very grateful to all the lab members of Robot Learning lab at Cornell, Ashesh Jain, Ian Lenz, Dipendra Misra, Ozan Sener, Hema Koppula, Yun Jiang and Chenxia Wu, for stimulating discussions and for all the fun we had. Also, I would like to thank Seok Hyun Jin for the countless sleepless nights working together towards the deadlines. Also, thanks to all collaborators on different projects including Colin Ponce, Joshua Reicher, Kevin Lee. Lastly, I am very fortunate to have known Ian Lenz, who was passionate about robotics and was always fun to be with, and he will be missed dearly.

Thanks also to all the lab members of Salisbury Lab at Stanford. In particular, with Manuel Ahumada and Mark Stauber, I had lots of fun chatting about all kinds of things and learned so much about building robots.

On a more personal note, I would like to thank my parents and my brother for their love and support. Finally, I would like to thank my fiancé, Bojeong Kim, for her endless support and patience for many years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The ability to perform many tasks with objects in complex environment is one of the key challenges for robots stepping into human environment. In our daily lives, we encounter and interact with a variety of complex environments without much effort. For instance, if you are a chef, you will almost certainly be able to walk into any kitchen in the world and start cooking very quickly without much help, after a quick tour of the environment. On the other hand, for an assistive robot, it is extremely challenging to adapt and generalize their skills to a new environment.

In order for a robot to physically interact with a large variety of objects, the robot needs to jointly reason about different modalities of information. Most of the previous robot learning algorithms are designed to perform a single manipulation task very accurately or are designed to handle a small variation of the same scene. By designing a learning algorithm that can learn a shared representation of different modalities, I will demonstrate that a single model can enable a robot to perform more than 100 manipulation tasks on a large number of objects it has never seen before.

A robot has to be able to reason about various types of modalities, including vision, natural language, geometry, motions, and even haptic feedback, that have intrinsically different statistical properties. By jointly reasoning about different modalities of information, a robot can model the interplay of tasks with objects in different environments. For instance, when a robot encounters an appliance it has never seen before, the robot should be able to come up with a basic strategy of actions on how to manipulate the appliance, by observing its shape and reading the instruction manual provided with the appliance. Especially if the robot has a lot of prior experience with various appliances, the robot should be able to generalize to the novel object. By developing models that can jointly reason about different sensory inputs and its prior experiences, a robot can learn to generalize a large variety of required motions for manipulation, including how to approach, grasp and interact with the object.

The focus of this dissertation is to develop algorithms that can reason about multimodal data, enabling robots to model large number of tasks on different objects in complex environments. Our robot collects data from different sensory inputs such as point-cloud, natural language, haptic sensor, RGB-D, and a database of manipulation trajectories. The models introduced in this dissertation are able to learn shared representations of these multimodal data in order to accomplish full sequences of complex manipulation motions. Figure 1.1 shows few of the tasks we test our robots on: planning a manipulation trajectory for a novel appliance with point-cloud and natural language, understanding human activity from RGB-D camera, and learning to represent haptic feedback of objects. In following sections, we describe in detail the challenges of a robot interacting in a human environment.

1.1 Unstructured Environment

The unstructured nature of human environments, such as homes and offices, poses a great challenge to assistive robots both in perception and interaction. In particular, we address the following aspects of many challenges:

- **Large variety of different scenarios.** Unlike a factory floor, there are large

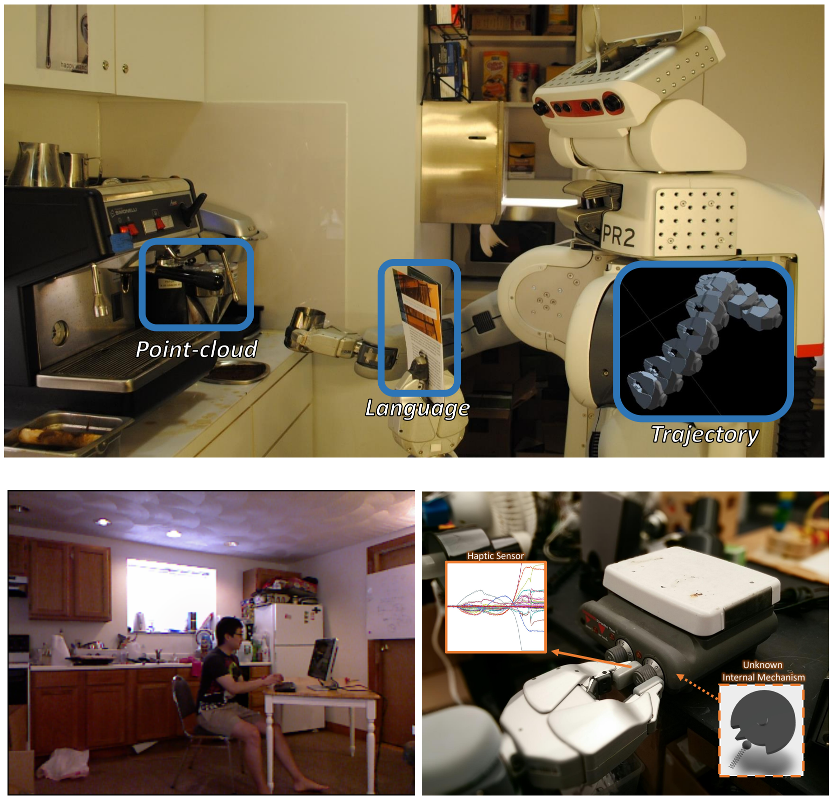


Figure 1.1: Examples of various tasks which we test our algorithms. Robots have to understand many novel appliances and humans in unstructured human environment and interact with them.

number variations of scenarios robots have to reason about in order to appropriately perform different tasks. For example, if a robot sees that a person in need of assistance heads towards a couch and asks the robot to bring a magazine from a table, it has to be able to respond to different scenarios such as when a magazine is under a mug or when a really thin magazine requires to be slid off to the side of the table to be picked up.

- **Large variety of objects and required motions.** Furthermore, there are large varieties of objects and their instances. Every household has different types of appliances (*e.g.* espresso machine, toaster, waffle maker and deep fryer). An espresso machine alone has thousands of variations

> of models with different capabilities, different shapes, and different sizes. Even for the same task of frothing milk, some machines have knobs that need to be rotated while some other machines have levers that need to be pulled and need to be held onto. Thus, in order to successfully manipulate these objects, the robot has to reason about infinitely many potential movements with different approach strategies, different grasp strategies, different types of interactions, and different release strategies.

Although there are significant developments in low-level controllers for robots, it is not feasible to build a single sequence or a single set of rules that can account for all kinds of different situations and manipulate all different types of objects. Even for a simple task such as boiling a water, depending on types of available tools, one may need to use a stove top, a microwave or an electric kettle. In Chapter 3, in order to accomplish manipulation tasks that are under-specified in instructions, we propose an algorithm that dynamically unrolls graphical models to sequence next motion primitives.

Robots are especially clueless when they encounter an object they have never seen before. Because every object type (*e.g.* espresso machine) can have such large variations, simply relying on scene understanding techniques, that can label or segment the object out of the scene, are not much useful for robots in manipulating them. Instead, based on an idea that many objects share similarly operated parts, we introduce an algorithm that learns to transfer manipulation trajectories across objects that serve completely different purpose. For instance, a robot that knows how to flush a urinal is now able to operate a similar handle on an espresso machine that is operated similar to the urinal handle (Chapter 5).

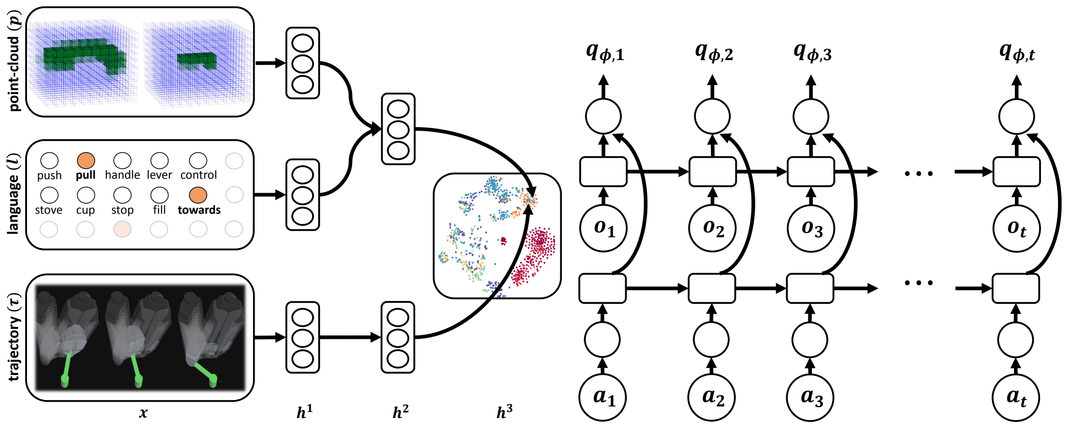


Figure 1.2: Various models encoding different multimodal data including point-cloud, natural language, manipulation trajectory, and tactile haptic feedback.

1.2 Learning Representation of Multimodal Data

People use many different sensory input — visual, auditory, and tactile — to accomplish different tasks in our daily lives. Humans are able to map from different sensory systems to same concepts using *common representation* of different modalities [29]. For example, we are able to correlate the appearance with feel of a banana, or a language instruction with a real-world action. For a robot to take advantage of such multimodal input, it should be able to find a good joint representation of different modalities. Finding a good representation of multimodal data has previously been found useful in many perceptual and interactive tasks including grasping [84], lip-audio understanding [111] and autonomous driving [189].

Obtaining a good common representation between different modalities is challenging for two main reasons. First, each modality might intrinsically have very different statistical properties — for example, most trajectory representations are inherently dense, while a bag-of-words representation of language is by nature sparse. This makes it challenging to apply algorithms designed for unimodal data, as one modality might overpower the others. Second, even with expert knowledge, it is extremely challenging to design joint features between such disparate modalities.

We introduce an algorithm that can find semantically meaningful representation of three distinct modalities of data. In Chapter 4, our deep multimodal embedding algorithm learns joint embedding of point-cloud, natural language, and manipulation trajectory using deep neural networks. These learned models provide semantically meaningful representation of modalities that allows robot to reason about different modalities in a shared embedding space. In Chapter 6, we also propose an algorithm that learns to jointly embed sequences of haptic feedback and sequences of previous actions. This model allows robot to dynamically adapt its strategy according to perceived feedback.

1.3 Applications

Such challenges of unstructured environment and multimodal data arise in many robotics tasks of perception, planning, and manipulation. We demonstrate our algorithms on several different real world problems robots encounter in unstructured human environments.

In order to handle such challenges of unstructured environment, we model the sequential nature of activities and tasks that involves sequences of different motions. For example, there is no single sequence of motions that can describe all activities that we would relate to as 'brushing teeth.' In Chapter 2, we show that robots can learn possible sequences of sub-activities by observing tasks performed by several people with a RGB camera and a depth camera. In Chapter 3, we then show that robots can dynamically plan a sequence of sub-activities (motion primitives) for different novel scenarios.

Furthermore, we present a general learning algorithm that can learn a joint embedding space of different modalities (Chapter 4). Using this algorithm, we learn a semantically meaningful representation of point-cloud, natural language, and motion trajectory, where motion trajectories consist of sub-sequences of motions representing approach, grasp, and interaction strategies. In Chapter 5, we utilize this learned embedding space to plan sequences of complex motions given visuals and natural language instruction manuals of novel objects. For instance, our algorithm enables our robot to even prepare a cup of latte without any prior experience of espresso machines or coffee grinders. Lastly, in Chapter 6, we show that our robot can also learn an appropriate representation of complex haptic feedback to influence transitions in sequences of motion trajectories.

1.4 First Published Appearances of Described Contributions

Most of the contributions presented in this thesis have first appeared as prior publications:

- Chapter 2: Sung, Ponce, Selman, and Saxena [146]
- Chapter 3: Sung, Selman, and Saxena [147]; Misra, Sung, Lee, and Saxena [101]
- Chapter 4: Sung, Lenz, and Saxena [150]; Sung, Jin, Lenz, and Saxena [149]

- Chapter 5: Sung, Jin, and Saxena [148]; Sung, Lenz, and Saxena [150]; Sung, Jin, Lenz, and Saxena [149]
- Chapter 6: Sung, Salisbury, and Saxena [151]

CHAPTER 2

UNSTRUCTURED HUMAN ACTIVITY DETECTION FROM RGBD IMAGES

Our goal is to enable robots to perform many tasks with different objects in complex environments, which requires modeling and understanding tasks performed in human environments. By observing humans carrying out tasks, robots can learn about interaction of tasks with objects and environment. In this chapter, we focus on modeling how humans perform different activities.

Being able to automatically infer the activity that a person is performing is essential in many applications, such as in personal assistive robotics. For example, if a robot could watch and keep track of how often a person drinks water, it could prevent the dehydration of elderly by reminding them. True daily activities do not happen in structured environments (e.g., with closely controlled background), but in uncontrolled and cluttered households and offices. Due to its unstructured and often visually confusing nature, detection of daily activities becomes a much more difficult task. In addition, each person has his or her own habits and mannerisms in carrying out tasks, and these variations in speed and style create additional difficulties in trying to detect and recognize activities. In this work, we are interested in reliably detecting daily activities that a person performs in a home or office, such as cooking, drinking water, brushing teeth, talking on the phone, and so on.

Most previous work on activity classification has focused on using 2D video (e.g., [115, 46]) or RFID sensors placed on humans and objects (e.g., [182]). The use of 2D videos leads to relatively low accuracy (e.g., 78.5% in [94]) even when there is no clutter. The use of RFID tags is generally too intrusive because it

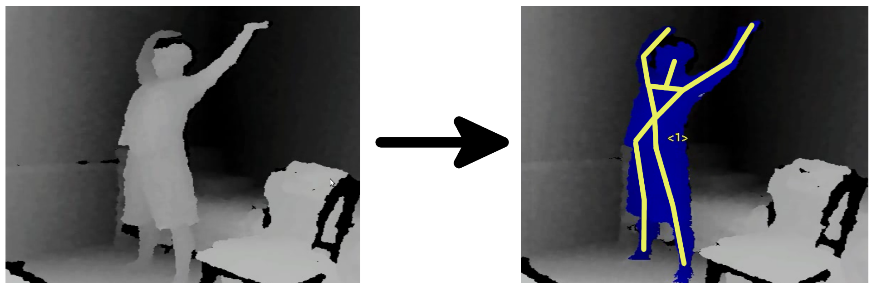


Figure 2.1: The RGBD data from the Kinect sensor is used to generate an articulated skeleton model. This skeleton is used along with the raw image and depths for estimating the human activity.

requires a placement of RFID tags on the people.

In this work, we perform activity detection and recognition using an inexpensive RGBD sensor (Microsoft Kinect). Human activities, despite their unstructured nature, tend to have a natural hierarchical structure; for instance, drinking water involves a three-step process of bringing a glass to one's mouth, tilting the glass and head to drink, and putting the glass down again. We can capture this hierarchical nature using a hierarchical probabilistic graphical model—specifically, a two-layered maximum entropy Markov model (MEMM). Even with this structured model in place, different people perform tasks at different rates, and any single graphical model will likely fail to capture this variation. To overcome this problem, we present a method of on-the-fly graph structure selection that can automatically adapt to variations in task speeds and style. Finally, we need features that can capture meaningful characteristics of the person. We accomplish this by using the PrimeSense skeleton tracking system [122] in combination with specially placed Histogram of Oriented Gradient [21] computer vision features. This approach enables us to achieve reliable performance in detection and recognition of common activities performed in typical cluttered human environments.

We evaluated our method on twelve different activities (see Figure 2.3) performed by four different people in five different environments: kitchen, office, bathroom, living room and bedroom. Our results show a precision/recall of 84.7%/83.2% in detecting the correct activity when the person was seen before in the training set and 67.9%/55.5% when the person was not seen before. We have also made the dataset and code available open-source at: `http://pr.cs.cornell.edu/humanactivities`

2.1 Related Work

There is a large body of previous work on human activity recognition. One common approach is to use space-time features to model points of interest in video [83, 27]. Several authors have supplemented these techniques by adding more information to these features [59, 179, 182, 94, 114, 135]. However, this approach is only capable of classifying, rather than detecting, activities. Other approaches include filtering techniques [129] and sampling of video patches [13]. Hierarchical techniques for activity recognition have been used as well, but these typically focus on neurologically-inspired visual cortex-type models [40, 137, 108, 124]. Often, these authors adhere faithfully to the models of the visual cortex, using motion-direction sensitive "cells" such as Gabor filters in the first layer [59, 115].

Another class of techniques used for activity recognition is that of the hidden Markov model (HMM). Early work by Brand et al. [15] utilize coupled HMMs to recognize two-handed activities. Weinland et al. [175] use an HMM together with a 3D occupancy grid to model human actions. Martinez-Contreras et al. [97] utilize motion templates together with HMMs to recognize human activities. As well as generative models like HMM, Lan et al. [82] employe a discriminative model which was aided by interaction analysis between people. Sminchisescu et al. [140] use conditional random fields (CRF) and maximum-entropy Markov models, arguing that these models overcome some of the limitations presented by HMMs. Notably, HMMs create long-term dependencies between observations and tries to model observations, which are already fixed at runtime. On the other hand, MEMM and CRF are able to avoid such dependencies and enables longer interaction among observations. However, the use of 2D videos leads to relatively low accuracies.

Other authors have worked on hierarchical dynamic Bayesian networks. Early work by Wilson and Bobick [178] extend HMM to parametric HMM for recognizing pointing gestures. Fine et al. [33] introduce hierarchical HMM, which was later extended by Bui et al. [16] to a general structure in which each child can have multiple parents. Truyen et al. [161] then developed a hierarchical semi-Markov CRF that could be used in partially observable settings. Liao et al. [93] apply hierarchical CRFs to activity recognition but their model requires many GPS traces and is only capable of off-line classification. Wang et al. [172] propose Dual Hierarchical Dirichlet Processes for surveillance of the large area. Among several others, the hierarchical HMM is the closest model of these to ours, but does not capture the idea that a single state may connect to different parents only for specified periods of time, as our model does. As a result, none of these models fit our problem of online detection of human activities in uncontrolled and cluttered environment. Since MEMM enables longer interaction among observations unlike HMM [140], the hierarchical MEMM allows us to take new observations and utilize dynamic programming to consider them in an online setting.

Various robotic systems have used activity recognition before. Theodoridis et al. [156] use activity recognition in robotic systems to discern aggressive activities in humans. Li et al. [92] discuss the importance of non-verbal communication between human and robot and developed a method to recognize simple activities that are nondeterministic in nature, while other works have focused on developing robots that utilizes activity recognition to imitate human activities [24, 95]. However, we are more interested here in assistive robots. Assistive robots are robots that assist humans in some task. Several types of assistive robots exist, including socially assistive robots that interact with another person in a non-contact manner, and physically assistive robots, which can physically help people [31, 153, 113, 89, 60, 74].

2.2 Our Approach

We use a supervised learning approach in which we collected ground-truth labeled data for training our model. Our input is RGBD images from a Kinect sensor, from which we extract certain features that are fed as input to our learning algorithm. We train a two-layered maximum-entropy Markov model which will capture different properties of human activities, including their hierarchical nature and the transitions between sub-activities over time.

2.2.1 Features

We can recognize a person's activity by looking at his current pose and movement over time, as captured by a set of features. The input sensor for our robot is a RGBD camera (Kinect) that gives us an RGB image as well as depths at each pixel. In order to compute the human pose features, we describe a person by a rigid skeleton that can move at fifteen joints (see Figure 2.1). We extract this skeleton using a tracking system provided by PrimeSense [122]. The skeleton is described by the length of the links and the joint angles. Specifically, we have the three-dimensional Euclidean coordinates of each joint and the orientation matrix of each joint with respect to the sensor. We compute features from this data as follows.

Body pose features. The joint orientation is obtained with respect to the sensor. However, we are interested in true pose, which is invariant of sensor location. Therefore, we transform each joint's rotation matrix so that the rotation is given with respect to the person's torso. For 10 joints, we convert each rotation matrix to half-space quaternions in order to more compactly represent the joint's orientation. (A more compact representation would be to use Euler angles, but they suffer from representation problem called gimbal lock [136].) Along with these joint orientations, we would like to know whether the person is standing or sitting, and whether or not person is leaning over. Such information is observed from the position of each foot with respect to the torso ($3 * 2$) by using the head and hip joints to compute the angle of the upper body against vertical. We have $10 * 4 + 3 * 2 + 1 = 47$ features for the body pose.

Hand Position. Hands play an especially important role in carrying out many activities, so information about what hands are doing can be quite powerful. In particular, we want to capture information such as "the left hand is near the stomach" or "the right hand is near the right ear." To do this, we compute the position of the hands with respect to the torso, and with the respect to the head in the local coordinate frame. Though we capture the motion information as described next, in order to emphasize hand movement, we also observe hand position over last 6 frames and record the highest and lowest vertical hand position. We have $2 * (6 + 2) = 16$ features for this.

Motion Information. Motion information is also important for classifying a person's activities. We select nine frames spread out over the last three seconds, spaced as follows: $\{-5, -9, -14, -20, -27, -35, -44, -54, -65\}$, where the numbers refer to the frames chosen. Then, we compute the joint rotations that have occurred between each of these frames and the current frame, represented as half-space quaternions (for the 11 joints with orientation information). This gives. $9 * 11 * 4 = 396$ features. We refer to body pose, hand and motion features as "skeletal features".

Image and point-cloud features. Much useful information can be derived directly from the raw image and point cloud as well. We use the Histogram of Oriented Gradients (HOG) feature descriptors [21], which gives 32 features that count how often certain gradient orientations are seen in specified bounding boxes of an image. Although this computation is typically performed on RGB or grayscale images, we can also view the depth map as a grayscale image and compute the HOG features on that. We have two HOG settings that we use. In the "simple HOG" setting, we find the bounding box of the person in the image, and compute RGB and depth HOG features for that bounding box, for a total of 64 features. In the "skeletal HOG" setting, we use the extracted skeleton model to find the bounding boxes for the person's head, torso, left arm, and right arm, and we compute the RGB and depth HOG features for each of these four bounding boxes, for a total of 256 features. In this chapter's primary result,

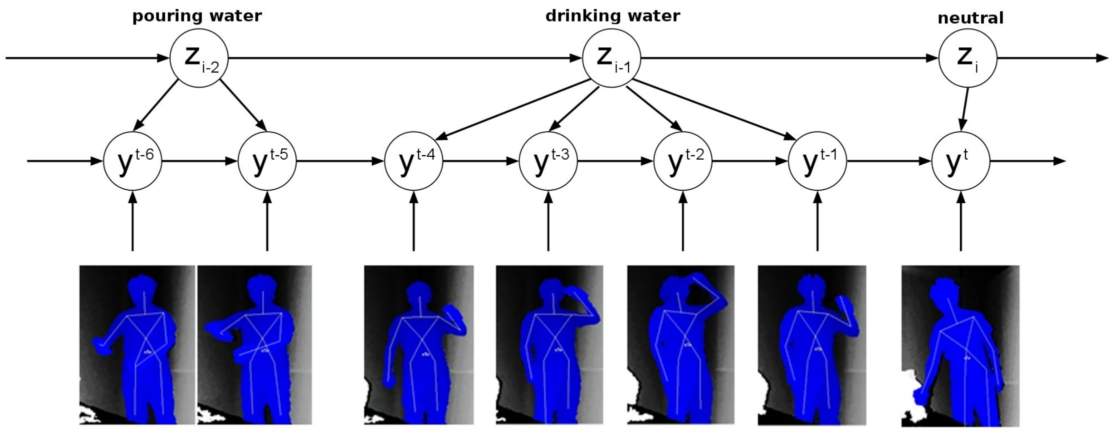


Figure 2.2: Our two-layered MEMM model.

we use the “skeletal HOG” setting.

2.2.2 Model Formulation

Human activity is complex and dynamic, and therefore our learning algorithm should model different nuances in human activities, such as the following.

First, an activity comprises a series of sub-activities. For example, the activity “brushing teeth” consists of sub-activities such as “squeezing toothpaste,” “bringing toothbrush up to face,” “brushing,” and so forth. Therefore for each activity (represented by $z \in Z$), we will model sub-activities (represented by $y \in Y$). We will train a hierarchical Markov model where the sub-activities y are represented by a layer of hidden variables (see Figure 2.2).

For each activity, different subjects perform the sub-activities for different periods of time. It is not clear how to associate the sub-activities to the activities. This implies that the graph structure of the model cannot be fixed in advance.

We therefore determine the connectivity between the z and the y layers in the model during inference.

Model. Our model is based on a maximum-entropy Markov model (MEMM) [98]. However, in order to incorporate the hierarchical nature of activities, we use a two-layered hierarchical structure, as shown in Figure 2.2.

In our model, let x^t denote the features extracted from the articulated skeleton model at time frame t. Every frame is connected to high-level activities through the mid-level sub-activities. Since high-level activities do not change every frame, we do not index them by time. Rather, we simply write z_i to denote the i^{th} high-level activity. Activity i occurs from time $t_{i-1}+1$ to time t_i. Then $\{y^{t_{i-1}+1}, ..., y^{t_i}\}$ is the set of sub-activities connected to activity z_i.

2.2.3 MEMM with Hierarchical Structure

As shown in Figure 2.2, each node z_i in the top layer is connected to several consecutive nodes in the middle layer $\{y^{t_{i-1}+1}, ..., y^{t_i}\}$, capturing the intuition that a single activity consists of a number of consecutive sub-activities.

For the sub-activity at each frame y^t, we do not know a priori to which activity z_i it should connect at the top layer. Therefore, our algorithm must decide when to connect a middle-layer node y^t to top-layer node z_i and when to connect it to next top-layer node z_{i+1}. We show in the next section how selection of graph structure can be done through dynamic programming. Given the graph structure, our goal is to infer the z_i that best explains the data. We do this by modeling the joint distribution $P(z_i, y^{t_{i-1}+1} \cdots y^{t_i} | O_i, z_{i-1})$ where $O_i = x^{t_{i-1}+1}, ..., x^{t_i}$, and for each z_i, we find the set of y^t's that maximize the joint probability. Finally, we choose the z_i that has the highest joint probability distribution.

Learning Model. We use a Gaussian mixture model to cluster the original training data into separate clusters, and consider each cluster as a sub-activity, rather than manually labeling sub-activities for each frame. We constrain the model to create five clusters for each activity, and then combine all the clusters for a certain location's activities into a single set of location specific clusters. In addition, we also generate a few clusters from the negative examples, so that our algorithm becomes robust to not detecting random activities. Specifically, for each classifier and for each location, we create a single cluster from each of the activities that do not occur in that location.

Our model consists of the following three terms:

- $P(y^t|x^t)$**:** This term models the dependence of the sub-activity label y^t on the features x^t. We model this using the Gaussian mixture model we have built. The parameters of the model are estimated from the labeled training data using maximum-likelihood.
- $P(y^{t_i-m}|y^{t_i-m-1}, z_i)$ (where $m \in \{0, ..., (t_i - t_{i-1} - 1)\}$). A sequence of sub-activities describes the activities. For example, we can say the sequence "squeezing toothpaste," "bringing toothbrush up to face," "actual brushing," and "putting toothbrush down" describes the activity "brushing teeth." If we only observe "bringing toothbrush up to face" and "putting toothbrush down," we would not refer to it as "brushing teeth." Unless the activity goes through a specific set of sub-activities in nearly the same sequence, it should probably not be classified as the activity. For all the activities except *neutral*, the table is built from observing the transition of posterior

probability for soft cluster of Gaussian mixture model at each frame.

However, it is not so straightforward to build $P(y^{t_i-m}|y^{t_i-m-1}, z_i)$ when z_i is *neutral*. When a sub-activity sequence such as "bringing toothbrush to face" and "putting toothbrush down" occurs, it does not correspond to any known activity and so is likely to be *neutral*. It is not possible to collect data of all sub-activity sequences that do not occur in our list of activities, so we rely on the sequences observed from non-*neutral* activities. If N denotes *neutral* activity, then $P(y^{t_i-m}|y^{t_i-m-1}, z_i = N) \propto 1 - \sum_{z_i \neq N} P(y^{t_i-m}|y^{t_i-m-1}, z_i)$.

- $P(z_i|z_{i-1})$. The activities evolve over time. For example, one activity may be more likely to follow another, and there are brief moments of *neutral* activity between two non-*neutral* activities. Thus, we can make a better estimate of the activity at the current time if we also use the estimate of the activity at previous time-step. Unlike other terms, due to difficulty of obtaining rich data set for maximum likelihood estimation, $P(z_i|z_{i-1})$ is set manually to capture these intuitions.

Inference. Consider the two-layer MEMM depicted in Figure 2.2. Let a single z_i activity node along with all the y^t sub-activity nodes connected directly to it and the corresponding x^t feature inputs be called a *substructure* of the MEMM graph. Given an observation sequence $O_i = x^{t_{i-1}+1}, ..., x^{t_i}$ and a previous activity z_{i-1}, we wish to compute the joint probability $P(z_i, y^{t_{i-1}+1} \cdots y^{t_i}|O_i, z_{i-1})$:

$$\begin{aligned}
&P(z_i, y^{t_{i-1}+1} \cdots y^{t_i}|O_i, z_{i-1}) \\
&\qquad = P(z_i|O_i, z_{i-1}) P(y^{t_{i-1}+1} \cdots y^{t_i}|z_i, O_i, z_{i-1}) \\
&\qquad = P(z_i|z_{i-1}) \cdot \prod_{t=t_{i-1}+2}^{t_i} P(y^t|y^{t-1}, z_i, x^t) \cdot \sum_{y^{t_{i-1}}} P(y^{t_{i-1}+1}|y^{t_{i-1}}, z_i, x^{t_{i-1}+1}) P(y^{t_{i-1}})
\end{aligned}$$

We have all of these terms except $P(y^t|y^{t-1}, z_i, x^t)$ and $P(y^{t_{i-1}+1}|y^{t_{i-1}}, z_i, x^{t_{i-1}+1})$. Both terms can be derived as

$$P(y^t|y^{t-1}, z_i, x^t) = \frac{P(y^{t-1}, z_i, x^t|y^t)P(y^t)}{P(y^{t-1}, z_i, x^t)}$$

We make a naive Bayes conditional independence assumption that y^{t-1} and z_i are independent from x^t given y^t. Using this assumption, we get:

$$P(y^t|y^{t-1}, z_i, x^t) = \frac{P(y^t|y^{t-1}, z_i)P(y^t|x^t)}{P(y^t)}$$

We have fully derived $P(z_i, y^{t_{i-1}+1} \cdots y^{t_i}|O_i, z_{i-1})$:

$$\begin{aligned} P(z_i, y^{t_{i-1}+1} \cdots y^{t_i}|O_i, z_{i-1}) =& P(z_i|z_{i-1}) \\ & \cdot \sum_{y^{t_{i-1}}} \frac{P(y^{t_{i-1}+1}|y^{t_{i-1}}, z_i)P(y^{t_{i-1}+1}|x^{t_{i-1}+1})}{P(y^{t_{i-1}+1})} P(y^{t_{i-1}}) \\ & \cdot \prod_{t=t_{i-1}+2}^{t_i} \frac{P(y^t|y^{t-1}, z_i)P(y^t|x^t)}{P(y^t)} \end{aligned}$$

Note that this formula can be factorized into two terms where one of them only contains two variables.

$$P(z_i, y^{t_{i-1}+1} \cdots y^{t_i}|O_i, z_{i-1}) = \mathcal{A} \cdot \prod_{t=t_{i-1}+2}^{t_i} \mathcal{B}(y^{t-1}, y^t)$$

Because the formula has factored into terms containing only two variables each, this equation can be easily and efficiently optimized. We simply optimize each factor individually, and we obtain:

$$\max P(z_i, y^{t_{i-1}+1} \cdots y^{t_i}|O_i, z_{i-1}) = \max_{y^{t_{i-1}+1}} \mathcal{A} \cdot \max_{y^{t_{i-1}+2}} \mathcal{B}(y^{t_{i-1}+1}, y^{t_{i-1}+2}) \cdots \max_{y^{t_i}} \mathcal{B}(y^{t_i-1}, y^{t_i})$$

2.2.4 Graph Structure Selection

Now that we can find the set of y^t's that maximize the joint probability $P(z_i, y^{t_{i-1}+1} \cdots y^{t_i}|O_i, z_{i-1})$, the probability of an activity z_i being associated with the

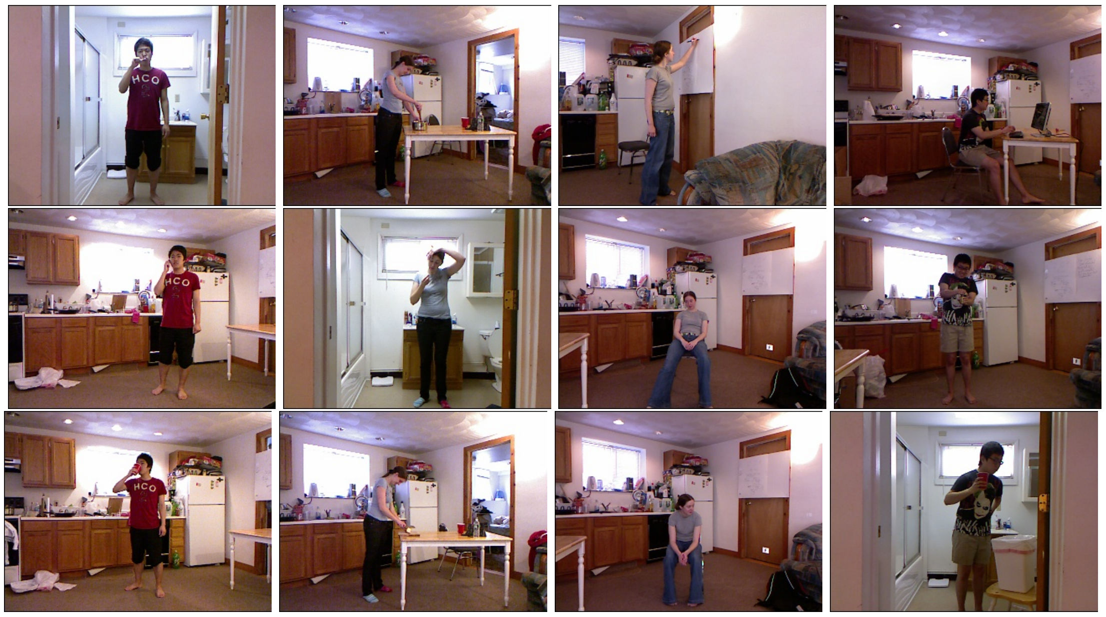


Figure 2.3: Samples from our dataset. Row-wise, from left: brushing teeth, cooking (stirring), writing on whiteboard, working on computer, talking on phone, wearing contact lenses, relaxing on a chair, opening a pill container, drinking water, cooking (chopping), talking on a chair, and rinsing mouth with water.

i^{th} substructure and the previous activity, we wish to use that to compute the probability of z_i given all observations up to this point. However, to do this, we must solve the following problem: for each observation y^t, we must decide to which high-level activity z_i it should be connected (see Figure 2.2). For example, consider the last y node associated with the "drinking water" activity in Figure 2.2. It's not entirely clear if that node really should connect to the "drinking water" activity, or if it should connect to the following "neutral" activity. Deciding with which activity node to associate each y node is the problem of hierarchical MEMM graph structure selection.

Unfortunately, we cannot simply try all possible graph structures. To see why, suppose we have a graph structure at time $t-1$ with a final high-level node z_i, and then are given a new node y^t. This node has two "choices": it can either connect to z_i, or it can create a new high-level node z_{i+1} and connect to that one. Because every node y^t has this same choice, if we see a total of n mid-level nodes, then there are 2^n possible graph structures.

We present an efficient method to find the optimal graph structure using dynamic programming. The method works, in brief, as follows. When given a new frame for classification, we try to find the point in time at which the current high-level activity started. So we pick a time t', and say that every frame after t' belongs to the current high-level activity. We have already computed the optimal graph structure for the first t' time frames, so putting these two subgraphs together give us a possible graph structure. We can then use this graph to compute the probability that the current activity is z. By trying all possible times $t' < t$, we can find the graph structure that gives us the highest probability, and we select that as our graph structure at time t.

The Method of Graph Structure Selection. Now we describe the method in detail. Suppose we are at some time t; we wish to select the optimal graph structure given everything we have seen so far. We will define the graph structure inductively based on graph structures that were chosen at previous points in time. Let $G_{t'}$ represent the graph structure that was chosen at some time $t' < t$. Note that, as a base case, G_0 is always the empty graph.

For every $t' < t$, define a candidate graph structure $\tilde{G}_t^{t'}$ consisting of $G_{t'}$ (the graph structure capturing the first t' timeframes), followed by a single substructure from time $t' + 1$ to time t connected to a single high-level node z_i. Note that this candidate graph structure sets $t_{i-1} = t'$ and $t_i = t$. Given the set of candidate structures $\{\tilde{G}_t^{t'} | 1 \leq t' < t\}$, the plan is to find the graph structure and high-level activity $z_i \in Z$ to maximize the likelihood given the set of observations so far.

Let O be the set of all observations so far. Then $P(z_i|O;\tilde{G}_t^{t'})$ is the probability that the most recent high-level node i is activity $z_i \in Z$, given all observations so far and parameterized by the graph structure $\tilde{G}_t^{t'}$. We initially set $P(z_0|O;G_0)$ to a uniform distribution. Then, through dynamic programming, we have $P(z_{i-1}|O;G_{t'})$ for all $t' < t$ and all $z \in Z$ (details below). Suppose that, at time t, we choose the graph structure $\tilde{G}_t^{t'}$ for a given $t' < t$. Then the probability that the most recent node i is activity z_i is given by

$$\begin{aligned}
P(z_i|O;\tilde{G}_t^{t'}) &= \sum_{z_{i-1}} P(z_i, z_{i-1}|O;\tilde{G}_t^{t'}) \\
&= \sum_{z_{i-1}} P(z_{i-1}|O;\tilde{G}_t^{t'})P(z_i|O, z_{i-1};\tilde{G}_t^{t'}) \\
&= \sum_{z_{i-1}} P(z_{i-1}|O;G_{t'})P(z_i|O_i, z_{i-1}) \qquad (2.1)
\end{aligned}$$

The two factors inside the summation are terms that we know, the former due to dynamic programming, and the latter estimated by finding maximum of $P(z_i, y^{t_{i-1}+1}\cdots y^{t_i}|O_i, z_{i-1})$, described in the previous section.

Thus, to find the optimal probability of having node i be a specific activity z_i, we simply compute

$$P(z_i|O;G_t) = \max_{t'<t} P(z_i|O;\tilde{G}_t^{t'})$$

We store $P(z_i|O;G_t)\ \forall z_i$ for dynamic programming purposes (Equation 2.1). Then, to make a prediction of an activity at time t, we compute

$$\begin{aligned}
\text{activity}_t &= \operatorname*{argmax}_{z_i} P(z_i|O) \\
&= \operatorname*{argmax}_{z_i} \max_{t'<t} P(z_i|O;\tilde{G}_t^{t'})
\end{aligned}$$

Optimality. We show that this algorithm is optimal by induction on the time t. Suppose we know the optimal graph structure for every time $t' < t$. This is certainly true at time $t = 1$, as the optimal graph structure at time $t = 0$ is the empty graph. The optimal graph structure at time t involves a final high-level node z_i that is connected to $1 \leq k \leq t$ mid-level nodes.

Suppose the optimal structure at time t has the high-level node connected to $k = t - t'$ mid-level nodes. Then what graph structure do we use for the first t' nodes? By the induction hypothesis, we know the optimal graph structure $G_{t'}$ for the first t' nodes. That is, $G_{t'}$ is the graph structure that maximizes the probability $P(z_{i-1}|O)$. Because z_i is conditionally independent of any high-level node before z_{i-1}, the graph structure before z_{i-1} does not affect z_i. Similarly, the graph structure before z_{i-1} obviously does not depend on the graph structure after z_{i-1}. Therefore, the optimal graph structure at time t is $\tilde{G}_t^{t'}$, the concatenation of $G_{t'}$ to a single substructure of $t - t'$ nodes.

We do not know what the correct time $0 \leq t' < t$ is, but because we try all, we are guaranteed to find the optimal t', and therefore the optimal graph structure.

Complexity. Let n and m be the number of activities and sub-activities, respectively, and let t be the time. Space complexity for the dynamic programming algorithm is $O(n \cdot t)$ since we store 1-d array of size t for each activity. At each timeframe, we must compute the optimal graph structure. By setting a maximum substructure size of $T \ll t$, dynamic programming requires n activities to be checked for each of T possible sizes. Each check requires a computation of $P(z_i, y^{t_{i-1}+1} \cdots y^{t_i}|O_i, z_{i-1})$, which takes $O(m \cdot T)$ time. Thus, each timeframe requires $O(n \cdot m \cdot T^2)$ computation time. We do this computation for each of t timeframes, for an overall time complexity of $O(n \cdot m \cdot T^2 \cdot t)$.

2.3 Experiments

Data. We used the Microsoft Kinect sensor, which outputs an RGB image together with aligned depths at each pixel at a frame rate of 30Hz. It produces a 640x480 depth image with a range of 1.2m to 3.5m. The sensor is small enough for it to be mounted on inexpensive mobile ground robots.

We considered five different environments: office, kitchen, bedroom, bathroom, and living room. Three to four common activities were identified for each location, giving a total of twelve unique activities (see Table 2.1). Data was collected from four different people: two males and two females. None of the subjects were otherwise associated with this project (and hence were not knowledgeable of our models and algorithm). We collected about 45 seconds of data for each activity from each person. The data was collected in different parts of regular household with no occlusion of arms and body from the view of sensor. When collecting, the subjects were given basic instructions on how to carry out the activity, such as "stand here and chop this onion," but were not given any instructions on how the algorithm would interpret their movements. (See Figure 2.3.)

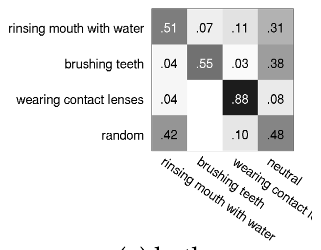
Our goal is to perform human activity *detection*, i.e., our algorithm must be able to distinguish the desired activities from other random activities that people perform. To that end, we collected *random* activities by asking the subject to act in a manner unlike any of the previously performed activities. The *random* activity contains sequence of random movements ranging from a person standing still to a person walking around and stretching his or her body. Note that *random* data was only used for testing.

For testing, we experimented with two settings. In the "new person" setting, we employed leave-one-out cross-validation to test each person's data; i.e. the model was trained on three of the four people from whom data was collected, and tested on the fourth. In the other "have seen" setting of the experiment, the model was given data about the person carrying out the same activity. To achieve this setting, we halved the testing subject's data and included one half in the training data set. So, even though the model had seen the person do the activity at least once, they had not seen the testing data itself.

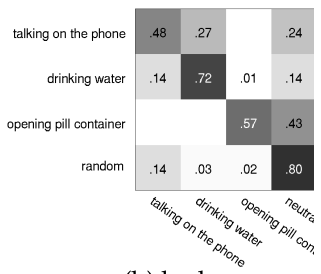
Finally, to train the model on both left-handed and right-handed people without needing to film them all, we simply mirrored the training data across the virtual plane down the middle of the screen. We have made the data available at: `http://pr.cs.cornell.edu/humanactivities/`

Models. We compared two-layered MEMM against two models, naive classifier based on SVM and one-level MEMM. Both models were trained on full set of features we have described earlier.

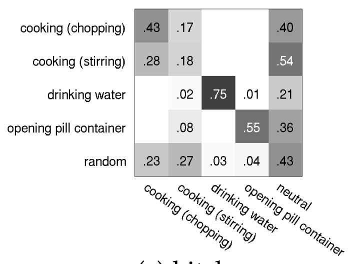
- *Baseline: Naive Classifier.* As the baseline model, we used a multi-class support vector machine (SVM) as a way to map features to corresponding activities. Here SVM is used to map the features to the high-level activities directly.
- *One-level MEMM.* This is a one-level MEMM model which builds upon the naive classifier. $P(y^t|x^t)$ is computed by fitting a sigmoid function to the output of the SVM. Transition probabilities between activities, $P(y^t|y^{t-1})$, use the same table we have built for full model, which in that model is called $P(z_i|z_{i-1})$. Using $P(y^t|x^t)$ and $P(y^t|y^{t-1})$, we compute the probability that the person is engaged in activity j at time t.



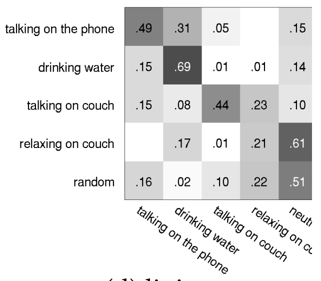
(a) bathroom



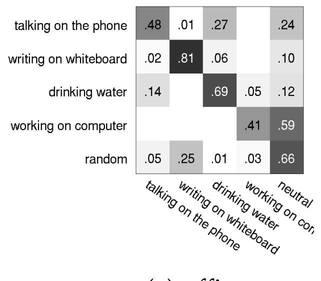
(b) bedroom



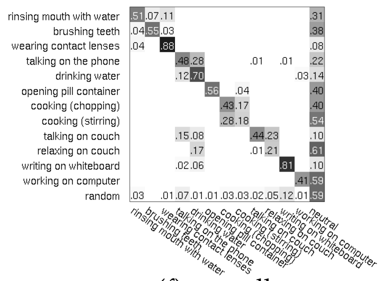
(c) kitchen



(d) living room



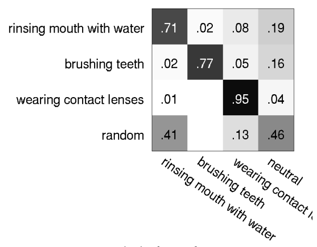
(e) office



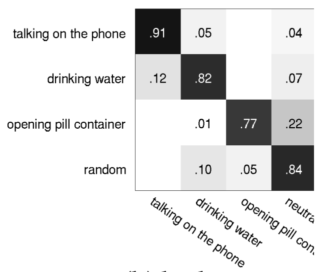
(f) overall

Figure 2.4: Leave-one-out cross-validation confusion matrix for each location with the full model in the "new person" setting, using skeletal features and skeletal HOG features. The *neutral* activity denotes that the algorithm estimates that the person is either not doing anything or that the person is engaged in some other activity that we have not defined. The last matrix (bottom-right) shows the results aggregated over all the locations.

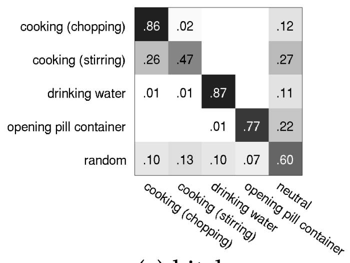
- *Hierarchical MEMM.* We ran our full model with a few different sets of input features in order to show how much improvement our selection of features brings compared to the set of features that solely relies on images. We tried using "simple HOG" features (using a person's full bounding box) with just RGB image data, "simple HOG" features with both RGB and depth data, and skeletal features with the "skeletal HOG" features for both RGB and depth data.



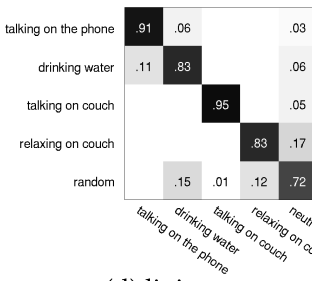
(a) bathroom



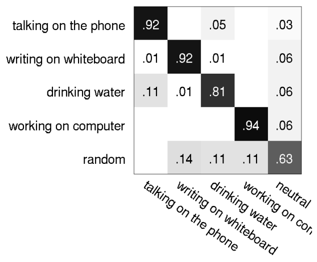
(b) bedroom



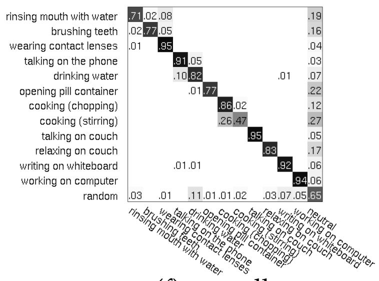
(c) kitchen



(d) living room



(e) office



(f) overall

Figure 2.5: Same format as Figure 2.4 except it is in the "have seen" setting.

2.3.1 Results and Discussion

Table 2.1 shows the results of the naive classifier, one-level MEMM and our full two-layered model for the "have seen" and "new person" settings. The precision and recall measures are used as metrics for evaluation. Our model was able to detect and classify with a precision/recall measure of 84.7%/83.2% and 67.9%/55.5% in "have seen" and "new person" settings, respectively. It is not surprising that the model performs better in the "have seen" setting, as it has seen that person's body type and mannerisms before.

We found that both the naive classifier and one-level MEMM were able to classify well when a frame contained distinct characteristics of an activity, but performed poorly when characteristics were subtler. For example, for tasks like 'rinsing mouth' and 'drinking water' that does not always have an apparent characteristics in every frame, these models consistently performed much lower in different locations as shown in Table 2.1. The one-layer MEMM was able to perform better than the naive classifier, as it naturally captures important temporal properties of motion. Our full two-layer MEMM, however, is able to capture the hierarchical nature of human activities in a way that neither the naive classifier nor the one-layer MEMM can do. As a result, it performed the best of all three models.

The comparison of feature sets on our full model shows that the features we use are much more robust compared to features that rely on RGB and/or Depth.

In the "have seen" setting, the HOG on RGB images are capable of capturing powerful information about a person. However, when seeing a new person, changes in clothing and background can cause confusion especially in uncontrolled and cluttered backgrounds, as shown by relatively low precision/recall value of 33.1%/23.5%. The skeletal features along with HOG on depth, while sometimes less informative than the HOG on images, are both more robust to changes in people. Thus, by combining skeletal features, skeletal HOG image features, and skeletal HOG depth features, we simultaneously achieved good accuracy in the "new person" setting and very good accuracy in the "have seen" setting.

Figure 2.4 and Figure 2.5 show the confusion matrices between the activities in "new person" and "have seen" setting when using skeletal features and "skeletal HOG" image and depth features. When it did not classify correctly, it usually chose the *neutral* activity, which is typically not as bad as choosing a wrong "active" activity. When we look at the confusion matrices, we see that many of the mistakes are actually reasonable in that the algorithm confuses them with very similar activities. For example, cooking-chopping and cooking-stirring are often confused, rinsing mouth with water is confused with brushing teeth, and talking on the couch is confused with relaxing on the couch. For these tasks, the skeleton tracker was not robust enough to provide accurate tracking of the arm and the hand motions.

Another strength of our model is that it correctly classifies *random* data as *neutral* most of the time, as shown in the bottom row of the confusion matrices. This means that it is able to distinguish whether the provided set of activities actually occurs or not—thus our algorithm is not likely to misfire when a person is doing some new activity that the algorithm has not seen before. Also, since we trained on both the regular and mirrored data, the model performs well with both left- and right-handed people.

However, there are some limitations to our method. First, our data only included cases in which the person was not occluded by an object; our method does not model occlusions and may not be robust to such situations. Second, some activities require more contextual information other than simply human pose. For example, knowledge of objects being used could help significantly in making human activity recognition algorithms more powerful in the future.

2.4 Conclusion

In order for a robot to perform tasks in a complex environment, the robot first has to be able to model various activities in unstructured human envrionments. In this chapter, we considered the problem of detecting and recognizing activities that humans perform in unstructured environments such as homes and offices. We used an inexpensive RGBD sensor (Microsoft Kinect) as the input sensor, the low cost of which enables our approach to be useful for applications such as smart homes and personal assistant robots. We presented a two-layered maximum entropy Markov model (MEMM). This MEMM modeled different properties of the human activities, including their hierarchical nature, the transitions between sub-activities over time, and the relation between sub-activities and different types of features. During inference, our algorithm exploited the hierarchical nature of human activities to determine the best MEMM graph structure. We tested our algorithm extensively on twelve different activities performed by four different people in five different environments, where the test activities were often interleaved with random activities not belonging to these twelve categories. It achieved good detection performance in both settings, where the person was and was not seen before in the training set, respectively.

Table 2.1: Results of naive classifier, one-level MEMM model, and our full model in each location. The table shows precision and recall scores for all of our models. Note that the test dataset contains *random* movements (in addition to the activities considered), ranging from a person standing still to walking around while waving his or her hands. RGB HOG and RGBD HOG refers to "simple HOG".

		"New Person"									
		Naive Classifier		One-layer MEMM		Full Model					
						RGB HOG		RGBD HOG		Skel.+Skel HOG	
Location	Activity	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec
bathroom	rinsing mouth	77.7	49.3	71.8	63.2	42.2	73.3	49.1	97.3	51.1	51.4
	brushing teeth	64.5	20.5	83.3	57.7	50.7	30.8	73.4	16.6	88.5	55.3
	wearing contact lens	82.0	89.7	81.5	89.7	44.2	40.6	52.5	59.5	78.6	88.3
	Average	74.7	53.1	78.9	70.2	45.7	48.2	58.3	57.8	72.7	65.0
bedroom	talking on the phone	82.0	32.6	82.0	32.6	0.0	0.0	15.6	8.8	63.2	48.3
	drinking water	19.2	12.1	19.1	12.1	0.0	0.0	3.0	0.1	70.0	71.7
	opening pill container	95.6	65.9	95.6	65.9	60.6	34.8	33.8	36.5	95.0	57.4
	Average	65.6	36.9	65.6	36.9	20.2	11.6	17.4	15.2	76.1	59.2
kitchen	cooking (chopping)	33.3	56.9	33.2	57.4	56.1	90.0	59.9	74.2	45.6	43.3
	cooking (stirring)	44.2	29.3	45.6	31.4	58.0	4.0	94.5	11.1	24.8	17.7
	drinking water	72.5	21.3	71.6	23.9	0.0	0.0	91.8	23.9	95.4	75.3
	opening pill container	76.9	6.2	75.8	6.2	83.6	33.5	54.1	35.0	91.9	55.2
	Average	56.8	28.4	56.6	29.7	49.4	31.9	75.1	36.1	64.4	47.9
living room	talking on the phone	69.7	0.9	83.3	25.0	0.0	0.0	31.0	11.8	51.5	48.5
	drinking water	57.1	53.1	52.8	55.8	0.0	0.0	1.2	0.0	54.3	69.3
	talking on couch	71.5	35.4	57.4	91.3	42.7	59.4	53.2	63.2	73.2	43.7
	relaxing on couch	97.2	76.4	95.8	78.6	0.0	0.0	100.0	21.5	31.3	21.1
	Average	73.9	41.5	72.3	62.7	10.7	14.9	46.4	24.1	52.6	45.7
office	talking on the phone	60.5	31.0	60.6	31.5	17.5	6.7	2.7	0.6	69.4	48.2
	writing on whiteboard	47.1	73.3	45.2	74.1	41.2	25.1	94.0	97.0	75.5	81.3
	drinking water	41.1	12.4	51.2	23.2	0.0	0.0	0.0	0.0	67.1	68.8
	working on computer	93.5	76.8	93.5	76.8	100.0	11.9	100.0	29.0	83.4	40.7
	Average	60.5	48.4	62.6	51.4	39.7	10.9	49.2	31.7	73.8	59.8
Overall Average		**66.3**	**41.7**	**67.2**	**50.2**	**33.1**	**23.5**	**49.3**	**33.0**	**67.9**	**55.5**

		"Have Seen"					
		Naive Classifier		One-layer MEMM		Full Model Skel.+Skel HOG	
Location	Activity	Prec	Rec	Prec	Rec	Prec	Rec
bathroom	rinsing mouth	73.3	49.7	70.7	53.1	61.4	70.9
	brushing teeth	81.5	65.1	81.5	75.6	96.7	77.1
	wearing contact lens	87.8	71.9	87.8	71.9	79.2	94.7
	Average	80.9	62.2	80.0	66.9	79.1	80.9
bedroom	talking on the phone	70.2	67.2	70.2	69.0	88.7	90.8
	drinking water	64.1	31.6	64.1	39.6	83.3	81.7
	opening pill container	48.7	52.3	48.7	54.8	93.3	77.4
	Average	61.0	50.4	61.0	54.5	88.4	83.3
kitchen	cooking (chopping)	78.9	28.9	78.9	29.0	70.3	85.7
	cooking (stirring)	44.6	45.8	44.6	45.8	74.3	47.3
	drinking water	52.2	51.5	52.2	52.4	88.8	86.8
	opening pill container	17.9	62.4	17.9	62.4	91.0	77.4
	Average	48.4	47.2	48.4	47.4	81.1	74.3
living room	talking on the phone	34.1	67.7	34.1	67.7	88.8	90.6
	drinking water	80.2	48.7	71.0	53.8	80.2	82.6
	talking on couch	91.4	50.7	91.4	50.7	98.8	94.7
	relaxing on couch	95.7	96.5	95.7	96.5	86.8	82.7
	Average	75.4	65.9	73.1	67.2	88.7	87.7
office	talking on the phone	80.4	52.2	80.4	52.2	87.6	92.0
	writing on whiteboard	42.5	59.3	42.5	59.3	85.5	91.9
	drinking water	53.4	36.7	53.4	36.7	82.3	81.5
	working on computer	89.2	69.3	89.2	69.3	89.5	93.8
	Average	66.4	54.4	66.4	54.4	86.2	89.8
Overall Average		**66.4**	**56.0**	**65.8**	**58.1**	**84.7**	**83.2**

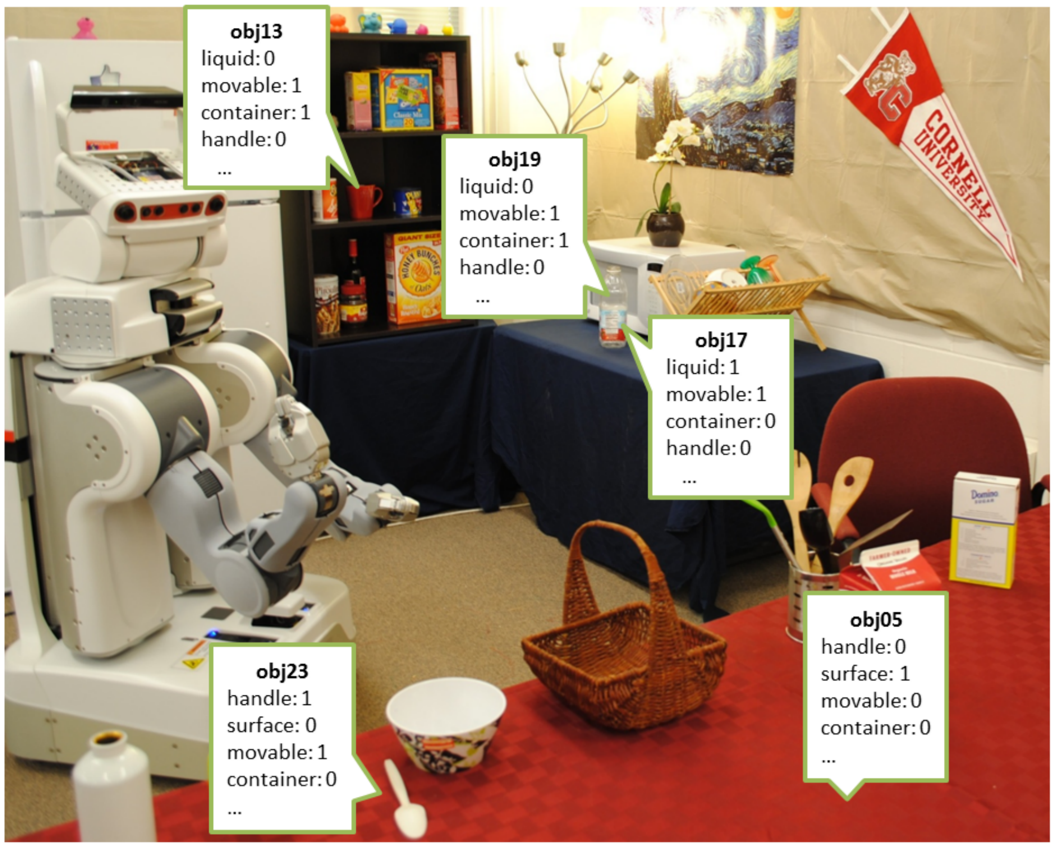
CHAPTER 3

SYNTHESIZING MANIPULATION SEQUENCES FOR UNDER-SPECIFIED TASKS USING UNROLLED MARKOV RANDOM FIELDS

When a robot performs a task in a complex environment, it should be able to plan its motions according to the context the environment. We previously introduced an algorithm that models human activities as sequences of sub-activities through observation. In this chapter, we focus on synthesizing a sequence of symbolic motion primitives by taking contexts of various objects into account.

When interacting with a robot, users often under-specify the tasks to be performed. For example in Figure 3.1, when asked to `pour` something, the robot has to infer which cup to pour into and a complete sequence of the navigation and manipulation steps—moving close, grasping, placing, and so on.

This sequence not only changes with the task, but also with the perceived state of the environment. As an example, consider the task of a robot fetching a magazine from a desk. The method to perform this task varies depending on several properties of the environment: for example, the robot's relative distance from the magazine, the robot's relative orientation, the thickness of the magazine, and the presence or the absence of other items on top of the magazine. If the magazine is very thin, the robot may have to slide the magazine to the side of the table to pick it up. If there is a mug sitting on top of the magazine, it would have to be moved prior to the magazine being picked up. Thus, especially when the details of the manipulation task are under-specified, the success of executing the task depends on the ability to detect the object and on the ability to sequence the set of *primitives* (navigation and manipulation controllers) in



Task: pour (obj17)
Inferred Sequence:
`move_close(obj13)` → `grasp(obj13)` → `move_close(obj05)` → `place_above(obj13,obj05)` → `release(obj13)` → `move_close(obj19)` → `grasp(obj19)` → `move_close(obj13)` → `hold_above(obj19,obj13)` → `follow_traj_pour(obj19,obj13)`

Figure 3.1: Figure showing our Kodiak PR2 in a kitchen with different objects labeled with attributes. To accomplish the under-defined task of `pour(obj17)`, it has to first find the mug (obj13) and carry it to the table (obj05) since it is dangerous to pour liquid in a tight shelf. Once the mug is on the table, it has to bring the liquid by the container (obj19) and then finally pour it into the mug.

various ways in response to the environment.

In recent years, there have been significant developments in building low-level controllers for robots [157] as well as in perceptual tasks such as object detection from sensor data [72, 61, 181]. In this work, our goal is to, given the environment and the task, enable robots to sequence the navigation and manipulation primitives. Manually sequencing instructions is not scalable because of the large variety of tasks and situations that can arise in unstructured environments.

In this work, we take an attribute-based representation of the environment, where each object is represented with a set of attributes, such as their size, shape-related information, presence of handles, and so forth. For a given task, there are often multiple objects with similar functions that can be used to accomplish the task, and humans can naturally reason and choose the most suitable object for the given task [69]. Our model, based on attribute representation of objects, is similarly capable of choosing the most suitable object for the given task among many objects in the environment.

We take a dynamic planning approach to the problem of synthesizing, in the right order, the suitable primitive controllers. The best primitive to execute at each discrete time step is based on a score function that represents the appropriateness of a particular primitive for the current state of the environment. Conceptually, a dynamic plan consists of a loop containing a sequence of conditional statements each with an associated primitive controller or action. If the current environment matches the conditions of one of the conditional statements, the corresponding primitive controller is executed, bringing the robot one step closer to completing the overall task (example in Section 3.2). We will show how to generalize sequencing of primitives to make them more flexible and robust, by switching to an attribute-based representation. We then show how to unroll the loop into a graph-based representation, isomorphic to a Markov Random Field. We then train the parameters of the model by maximum margin learning method using a dataset comprising many examples of sequences.

We evaluated our model on 127 controller sequences for five under-specified manipulation tasks generated from 13 environments using 7 primitives. We show that our model can predict suitable primitives to be executed with the correct arguments in most settings. Furthermore, we show that, for five high-level tasks, our algorithm was able to correctly sequence 70% of the sequences in different environments.

The main contributions of this chapter are:

- We propose the use of attribute-based representation of an environment for task planning.
- We infer the sequence of steps where goals are under-specified and have to be inferred from the context.
- We represent a dynamic plan as a graph by unrolling the loop into a Markov Random Field.

3.1 Related Work

There is a large body of work in task planning across various communities. We describe some of them in the following categories.

Manual Controller Sequencing. Many works manually sequence different types of controllers to accomplish specific types of tasks. Bollini et al. [14] develop an end-to-end system which can find ingredients on a tabletop and mix them uniformly to bake cookies. Others used pre-programmed sequences for tea serving and carrying humans in healthcare robotics [109, 106]. These approaches however cannot scale to large number of tasks when each task requires its own complicated rules for sequencing controllers and assumes a controlled environment, which is very different from actual human households, where objects of interest can appear anywhere in the environment with a variety of similar objects.

Beetz et al. [8] retrieve a sequence for "making a pancake" from online websites but assumes an environment with correct labels and a single choice of object for the task. Human experts can generate finite state machines for robots but this again requires explicit labels (e.g. AR tags) [112]. Our work addresses these problems by representing each object in the environment as a set of attributes which is more robust than labeling the individual object [32, 30, 81]. In our recent work [102], we learn a sequence given a natural language instruction and object labels, where the focus is to learn the grounding of the natural language into the environment.

Learning Activities from Videos. In the area of computer vision, several works [184, 185, 146, 73] consider modeling the sequence of activities that humans perform. These works are complementary to ours because our problem is to infer the sequence of controllers and not to label the videos.

Symbolic Planning. Planning problems often rely on symbolic representation of entities as well as their relations. This has often been formalized as a deduction [45] or satisfiability problem [67]. A plan can also be generated hierarchically by first planning abstractly, and then generating a detailed plan recursively [65]. Such approaches can generate a sequence of controllers that can be proven to be correct [64, 9]. Symbolic planners however require encoding every precondition and effect of each operation, which will not scale in human environments where there are large variations. Such planners also require domain description for each planning domain including the types of each object (e.g., pallet crate - surface, hoist surface - locatable) as well as any relations (e.g., on x:crate y:surface, available x:hoist). The preconditions and effects can be learned directly from examples of recorded plans [183, 190] but this method suffers when there is noise in the data [190], and also suffers from the difficulty of modeling real world situations with the PDDL representation [183].

Such STRIPS-style representation also restricts the environment to be represented with explicit labels. Though there is a substantial body of work on labeling human environments [72, 79], it still remains a challenging task. A more reliable way of representing an environment is representing through attributes [32, 30]. An attribute-based representation even allows classification of object classes that are not present in the training data [81]. Similarly, in our work, we represent the environment as a set of attributes, allowing the robot to search for objects with the most suitable attributes rather than looking for a specific object label.

Predicting Sequences. Predicting sequences has mostly been studied in a Markov Decision Process framework, which finds an optimal policy given the reward for each state. Because the reward function cannot be easily specified in many applications, inverse reinforcement learning (IRL) learns the reward function from an expert's policy [110]. IRL is extended to Apprenticeship Learning based on the assumption that the expert tries to optimize an unknown reward function [1]. Most similar to our work, the Max-Margin Planning frames imitation learning as a structured max-margin learning problem [125]. However, this has only been applied to problems such as 2D path planning, grasp prediction and footstep prediction [126], which have much smaller and clearer sets of states and actions compared to our problem of sequencing different controllers. Co-Active Learning for manipulation path planning [57], where user preferences are learned from weak incremental feedback, does not directly apply to sequencing different controllers.

Both the model-based and model-free methods evaluate state-action pairs. When it is not possible to have knowledge about all possible or subsequent states (*full backup*), they can rely on *sample backup* which still requires sufficient sample to be drawn from the state space [36]. However, when lots of robot-object interactions are involved, highly accurate and reliable physics-based robotic simulation is required along with reliable implementation of each manipulation controllers. Note that each of the manipulation primitives such as grasping are still not fully solved problems. For example, consider the scenario where the robot is grasping the edge of the table and was given the instruction of `follow_traj_pour(table,shelf)`. It is unclear what should occur in the environment and becomes challenging to have reliable simulation of actions. Thus, in the context of reinforcement learning, we take a maximum margin based approach to learning the weight for $\mathbf{w}^T\phi(s, a)$ such that it maximizes the number of states where the expert outperforms other policies, and chooses the action that maximizes $\mathbf{w}^T\phi(s, a)$ at each time step. The key in our work is representing task planning as a graph-based model and designing a score function that uses attribute-based representation of environment for under-specified tasks.

3.2 Our Approach

We refer to a sequence of *primitives* (low-level navigation and manipulation controllers) as a *program*. To model the sequencing of primitives, we first represent each object in the environment with a set of attributes as described in Section 3.3.2. In order to make programs generalizable, primitives should have the following two properties. First, each primitive should specialize in an atomic operation such as moving close, pulling, grasping, and releasing. Second, a primitive should not be specific to a single high-level task. By limiting the role of each primitive and keeping it general, many different manipulation tasks can be accomplished with the same small set of primitives, and our approach becomes easily adaptable to different robots by providing implementation of primitives on the new robot.

For illustration, we write a program for "throw garbage away" in Program 1. Most tasks could be written in such a format, where there are many `if` statements inside the loop. However, even for a simple "throw garbage away" task, the program is quite complex. Writing down all the rules that can account for the many different scenarios that can arise in a human environment would be quite challenging.

Program 1 is an example of what is commonly referred to as reactive or dynamic planning [130, 71]. In traditional deliberative planning, a planning algorithm synthesizes a sequence of steps that starts from the given state and reaches the given goal state. Although current symbolic planners can find optimal plan sequences consisting of hundreds of steps, such long sequences often break down because of unexpected events during the execution. A dynamic

Program 1 "throw garbage away."

```
Input: environment $e$, trash $a_1$
$gc := find\_garbage\_can(e)$
repeat
    if $a_1$ is in hand & $gc$ is close then
        release($a_1$)
    else if $a_1$ is in hand & far from $gc$ then
        move_close($gc$)
    else if $a_1$ is close & $a_1$ not in hand & nothing on top of $a_1$ then
        grasp($a_1$)
     ⋮
    else if $a_1$ is far then
        move_close($a_1$)
    end if
until $a_1$ inside $gc$
```

plan provides a much more robust alternative. At each step, the current state of the environment is considered and the next appropriate action is selected by one of the conditional statements in the main loop. A well-constructed dynamic plan will identify the next step required to bring the robot closer to the overall goal in any possible world state. In complex domains, dynamic plans may become too complicated. However, we are considering basic human activities, such as following a recipe, where dynamic plans are generally quite compact and can effectively lead the robot to the goal state. Moreover, as we will demonstrate, we can learn the dynamic plan from observing a series of action sequences in related environments.

In order to make our approach more general, we introduce a feature based representation for the conditions of `if` statements. We can extract some features from both the environment and the action that will be executed in the body of `if` statement. With extracted features ϕ and some weight vector $\mathbf{w}$ for each `if` statement, the same conditional statements can be written as $\mathbf{w}^T\phi$, since the environment will always contain the rationale for executing certain primitive. Such a feature-based approach allows us to re-write Program 1 in the form of Program 2.

Program 2 "throw garbage away."

```
Input: environment $e$, trash $a_1$
$gc := find\_garbage\_can(e)$
repeat
    $e_t$ := current environment
    if $w_1^T\phi(e_t,\text{release}(a_1)) > 0$ then
        release($a_1$)
    else if $w_2^T\phi(e_t,\text{move\_close}(gc)) > 0$ then
        move_close($gc$)
    ⋮
    else if $w_n^T\phi(e_t,\text{move\_close}(a_1)) > 0$ then
        move_close($a_1$)
    end if
until $a_1$ inside $gc$
```

Now all the `if` statements have the same form, where the same primitive along with same arguments are used in both the condition as well as the body of the `if` statement. We can therefore reduce all `if` statements inside the loop further down to a simple line which depends only on a single weight vector and a single joint feature map, as shown in Program 3, for finding the most suitable pair of primitive $\hat{p}_t$ and its arguments ($\hat{a}_{1,t}$, $\hat{a}_{2,t}$).

Program 3 "throw garbage away."

```
**Input:** environment $e$, trash $g_{a1}$
**repeat**
    $e_t$ := current environment
    $(\hat{p}_t, \hat{a}_{1,t}, \hat{a}_{2,t}) := \underset{p_t \in \mathcal{P}, a_{1,t}, a_{2,t} \in \mathcal{E}}{\operatorname{argmax}} w^T \phi(e_t, p_t(a_{1,t}, a_{2,t}))$
    execute $\hat{p}_t(\hat{a}_{1,t}, \hat{a}_{2,t})$
**until** $\hat{p}_t = done$
```

The approach taken in Program 3 also allowed removing the function $find\_garbage\_can(e)$. Both Program 1 and Program 2 require $find\_garbage\_can(e)$ which depends on semantic labeling of each object in the environment. The attributes of objects will allow the program to infer which object is a garbage can without explicit encoding.

Program 3 provides a generic representation of a dynamic plan. We will now discuss an approach to learning a set of weights. To do so, we will employ a graph-like representation obtained by "unrolling" the loop representing discrete time steps by different layers. We will obtain a representation that is isomorphic to a Markov Random Field (MRF) and will use a maximum margin based approach to training the weight vector. Our MRF encodes the relations between the environment, primitive and its arguments. Our empirical results show that such a framework is effectively trainable with a relatively small set of example sequences. Our feature-based dynamic plan formulation therefore offers an effective and general representation to learn and generalize from action sequences, accomplishing high-level tasks in a dynamic environment.

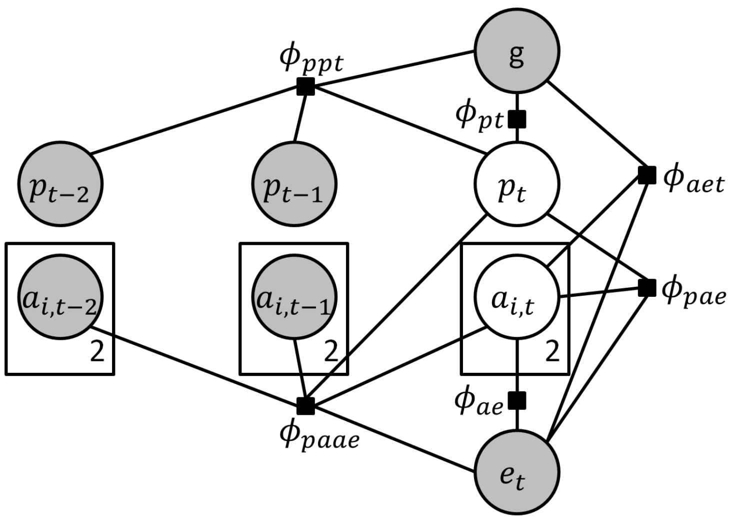


Figure 3.2: **Markov Random Field representation of our model** at discrete time step t. The top node represents the given task g, g_{a1}, g_{a2}. The second layer from the top represents the sequence of primitives, and the layer below represents the arguments associated with each primitive. And, the bottom node represents the environment represented with set of attributes.

3.3 Model Formulation

We are given a set of possible primitives $\mathcal{P}$ (navigation and manipulation controllers) to work with (see Section 3.4) and an environment $\mathcal{E}$ represented by a set of attributes. Using these primitives, the robot has to accomplish a manipulation task $g \in \mathcal{T}$. The manipulation task g is followed by the arguments $g_{a1}, g_{a2} \in \mathcal{E}$ which give a specification of the task. For example, the program "throw garbage away" would have a single argument which would be the object id of the object that needs to be thrown away.

At each time step t (i.e., at each iteration of the loop in Program 3), our environment e_t will dynamically change, and its relations with the primitive is represented with a joint set of features. These features include information about the physical and semantic properties of the objects as well as information about their locations in the environment.

Now our goal is to predict the best primitive $p_t \in \mathcal{P}$ to execute at each discrete time step, along with its arguments: $p_t(a_{1,t}, a_{2,t})$. We will do so by designing a score function $S(\cdot)$ that represents the correctness of executing a primitive in the current environment for a task.

$$S(g(g_{a1}, g_{a2}), e_t, p_t(a_{1,t}, a_{2,t})) = w^T \phi(g(g_{a1}, g_{a2}), e_t, p_t(a_{1,t}, a_{2,t}))$$

In order to have a parsimonious representation, we decompose our score function using a model isomorphic to a Markov Random Field (MRF), shown in Figure 3.2. This allows us to capture the dependency between primitives, their arguments, and environments which are represented by set of attributes. In the figure, the top node represents the given task and its arguments (g, g_{a1}, g_{a2}). The second layer from the top represents the sequence of primitives, and the layer below represents the arguments associated with each primitive. And, the bottom node represents the environment which is represented with set of attributes. Note that we also take into account the previous two primitives in the past, together with their arguments: $p_{t-1}(a_{1,t-1}, a_{2,t-1})$ and $p_{t-2}(a_{1,t-2}, a_{2,t-2})$.

Now the decomposed score function is:

$$S = \underbrace{S_{ae}}_{\text{args-env}} + \overbrace{S_{pt}}^{\text{prim-task}} + \underbrace{S_{aet}}_{\text{args-env-task}} + \overbrace{S_{pae}}^{\text{prim-args-env}} + \underbrace{S_{ppt}}_{\text{prim-prim(prev)-task}} + \overbrace{S_{paae}}^{\text{prim-args-args(prev)-env}}$$

The terms associated with an edge in the graph are defined as a linear function of its respective features ϕ and weights w:

$$S_{ae} = {w_{ae1}}^T \phi_{ae}(a_{1,t}, e_t) + {w_{ae2}}^T \phi_{ae}(a_{2,t}, e_t)$$

$$S_{pt} = {w_{pt}}^T \phi_{pt}(p_t, g)$$

Similarly, the terms associated with a clique in the graph are defined as a linear function of respective features ϕ and weights w:

$$\begin{aligned}
S_{aet} &= {w_{aet1}}^T \phi_{aet}(a_{1,t}, e_t, g) + {w_{aet2}}^T \phi_{aet}(a_{2,t}, e_t, g) \\
S_{pae} &= {w_{pae1}}^T \phi_{pae}(p_t, a_{1,t}, e_t) + {w_{pae2}}^T \phi_{pae}(p_t, a_{2,t}, e_t) \\
S_{ppt} &= {w_{ppt1}}^T \phi_{ppt}(p_{t-1}, p_t, g) + {w_{ppt2}}^T \phi_{ptt}(p_{t-2}, p_t, t) \\
S_{paae} &= \sum_{i,j\in(1,2),k\in(t-2,t-1)} {w_{paae_{ijk}}}^T \phi_{paae}(p_t, a_{i,k}, a_{j,t}, e_t)
\end{aligned}$$

Using these edge and clique terms, our score function S can be simply written in the following form, which we have seen in Program 3 with an extra term g for the task: $S(g(g_{a1}, g_{a2}), e_t, p_t(a_{1,t}, a_{2,t})) = w^T \phi(g(g_{a1}, g_{a2}), e_t, p_t(a_{1,t}, a_{2,t}))$.

3.3.1 Features

In this section, we describe our features $\phi(\cdot)$ for the different terms in the previous section.

Arguments-environment (ϕ_{ae}): The robot should be aware of its location and the current level of its interaction with objects (e.g., grasped), which are given as possible primitive arguments $a_{1,t}, a_{2,t}$. Therefore, we add two binary features which indicate whether each primitive argument is already grasped and two features for the centroid distance from the robot to each primitive arguments.

For capturing spatial relation between two objects $a_{1,t}$ and $a_{2,t}$, we add one binary feature indicating whether primitive arguments $a_{1,t}, a_{2,t}$ are currently in collision with each other.

Arguments-environment-task (ϕ_{aet}): To capture relations between the objects of interest (task arguments) and objects of possible interest (primitive arguments), we build a binary vector of length 8. First four values represent the indicator values of whether the objects of interest are identical as the objects of possible interest, and the last four represent spatial relation of whether they overlap from top view.

It is important to realize the type of object that is below the objects of interests, and the desired property (e.g., bowl-like object or table-like object) may differ depending on the situation. We create two feature vectors, each of length l. If the robot is holding the object, we store its extracted attributes in the first vector. Otherwise, we store them in the second vector. If the primitive has two arguments, we use the first primitive argument since it often has higher level of interaction with the robot compared to the second argument.

Finally, to capture correlation between the high-level task and the types of object in primitive argument, we take a tensor product of two vectors: an attribute vector of length $2l$ for two objects and a binary occurrence vector of length $|\mathcal{T}|$. The matrix of size $2l \times |\mathcal{T}|$ is flattened to a vector.

Primitive-task (ϕ_{pt}): The set of primitives that are useful may differ depending on the type of the task. We create a $|\mathcal{T}| \times |\mathcal{P}|$ binary co-occurrence matrix between the task g and the primitive p_t that has a single non-zero entry in the current task's (g^{th}) row and current primitive's (${p_t}^{th}$) column.

Primitive-arguments-environment (ϕ_{pae}): Some primitives such as `hold_above` require one of the objects in arguments to be grasped or not to be grasped to execute correctly. We create a $|\mathcal{P}| \times 2$ matrix where the row for the current primitive (${p_t}^{th}$ row) contains two binary values indicating whether each primitive argument is in the manipulator.

Primitive-primitive(previous)-task (ϕ_{ppt}): The robot makes different transitions between primitives for different tasks. Thus, a binary co-occurrence matrix of size $|\mathcal{T}| \times |\mathcal{P}|^2$ represents transition occurrence between the primitives for each task. In this matrix, we encode two transitions for the current task g, from $t-2$ to t and from $t-1$ to t.

Primitive-arguments-arguments(previous)-environment (ϕ_{paae}): For a certain primitive in certain situations, the arguments may not change between time steps. For example, `pour(A,B)` would often be preceded by `hold_above (A,B)`. Thus, the matrix of size $|\mathcal{P}| \times 8$ is created, with the ${p_t}^{th}$ row containing 8 binary values representing whether the two primitive arguments at time t are the same as the two arguments at $t-1$ or the two arguments at $t-2$.

3.3.2 Attributes.

Every object in the environment including tables and the floor is represented using the following set of attributes: height h, max(width(w),length(l)), min(w, l), volume($w * l * h$), min(w, l, h)-over-max(w, l, h), median(w, l, h)-over-max(w, l, h), cylinder-shape, box-shape, liquid, container, handle, movable, large-horizontal-surface, and multiple-large-horizontal-surface. Attributes such as cylinder-shape, box-shape, container, handle, and large-horizontal-surface can be reliably extracted from RGB or RGBD images, and were shown to be useful in several different applications [32, 30, 81, 72]. We study the effects of attribute detection errors on our model in Section 3.4.

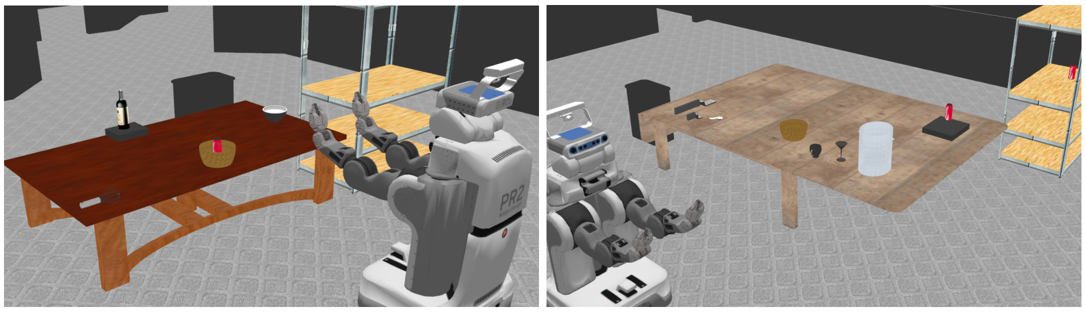


Figure 3.3: Figure showing two of our 13 environments in our evaluation dataset using 43 objects along with PR2 robot.

3.3.3 Learning

We use a max-margin approach to train a single model for all tasks. This maximum margin approach fits our formulation, since it assumes that the discriminant function is a linear function of a weight vector $\mathbf{w}$ and a joint feature map $\phi(g(g_{a1}, g_{a2}), e_t, p_t(a_{1,t}, a_{2,t}))$, and it has time complexity linear with the number of training examples when solved using the cutting plane method [62]. We formalize our problem as a "1-slack" structural SVM optimization problem:

$$
\begin{aligned}
&\min_{\mathbf{w},\xi\geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{l}\sum_{i=1}^{n}\sum_{t=1}^{l^i}\xi_t^i \\
s.t. \quad & \text{for } \ 1 \leq i \leq n, \text{for each time step } t: \\
& \forall \hat{p} \in \mathcal{P}, \forall \hat{a}_1, \hat{a}_2 \in \mathcal{E}: \\
& \quad \boldsymbol{w}^T[\phi(g^i(g^i_{a1}, g^i_{a2}), e^i_t, p^i_t(a^i_{1,t}, a^i_{2,t})) - \phi(g^i(g^i_{a1}, g^i_{a2}), e^i_t, \hat{p}(\hat{a}_1, \hat{a}_2))] \\
& \quad \geq \Delta(\{p^i_t, a^i_{1,t}, a^i_{2,t}\}, \{\hat{p}, \hat{a}_1, \hat{a}_2\}) - \xi^i_t
\end{aligned}
$$

where n is the number of example sequences, l^i is the length of the i^{th} sequence, and l is the total length combining all sequences. The loss function is defined as:

$$\Delta(\{p, a_1, a_2\}, \{\hat{p}, \hat{a}_1, \hat{a}_2\}) = \mathbb{1}(p \neq \hat{p}) + \mathbb{1}(a_1 \neq \hat{a}_1) + \mathbb{1}(a_2 \neq \hat{a}_2)$$

Table 3.1: Result of baselines, our model with variations of feature sets, and our full model on our dataset consisting of 127 sequences. The "prim" columns represent percentage of primitives correctly chosen regardless of arguments, and "args" columns represent percentage of a correct pair of primitive and arguments. The last column shows average percentage of sequences correct over the five programs evaluated.

	move_close		grasp		release		place_above		hold_above		traj_circle		traj_pour		**Average**		**Sequence**	
	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg
chance	14.3	1.1	14.3	1.1	14.3	1.1	14.3	0.1	14.3	0.1	14.3	1.1	14.3	0.1	14.3	0.7	0	0
multiclass	99.6	-	90.4	-	95.7	-	68.5	-	79.7	-	100.0	-	14.7	-	78.4	-	-	-
symb-plan-svm	99.6	82.5	94.2	72.4	67.4	63.0	60.9	43.5	76.6	73.4	96.7	76.7	97.1	91.2	84.6	71.8	58.4	49.6
symb-plan-manual	99.6	85.4	94.2	76.3	67.4	63.0	60.9	50.0	76.6	76.6	96.7	96.7	97.1	97.1	84.6	77.9	58.4	54.9
Only edge features	23.5	15.3	56.4	45.5	93.5	93.5	0.0	0.0	18.8	9.4	100.0	100.0	50.0	44.1	48.9	44.0	0	0
Only clique features	99.6	1.9	96.8	82.7	90.2	90.2	72.8	15.2	87.5	15.6	96.7	96.7	100.0	97.1	91.9	57.0	45.0	0
Ours - full	99.3	82.8	96.8	84.0	97.8	97.8	89.1	79.3	96.9	92.2	100.0	100.0	97.1	94.1	96.7	**90.0**	91.6	**69.7**

With a learned **w**, we choose the next action in sequence by selecting a pair of primitive and arguments that gives the largest discriminant value:

$$\operatorname*{argmax}_{p_t \in \mathcal{P}, a_{1,t}, a_{2,t} \in \mathcal{E}} \; w^T \phi(g(g_{a1}, g_{a2}), e_t, p_t(a_{1,t}, a_{2,t}))$$

3.4 Experiments

Dataset. We considered seven primitives (low-level controllers): `move_close (A)`,`grasp (A)`,`release (A)`,`place_above (A,B)`,`hold_above (A,B)`, `follow_traj_circle (A)` and `follow_traj_pour (A,B)`. Depending on the environment and the task, these primitives could be instantiated with different arguments. For example, consider an environment that contains a bottle (obj04) containing liquid (obj16) and an empty cup (obj02) placed on top of the shelf, among other objects. If, say from a recipe, our task is to pour the liquid, then our program should figure out the correct sequence of primitives with correct arguments (based on the objects' attributes, etc.):

```
{pour(obj16);env2} →
{move_close(obj02); grasp(obj02); move_close(obj04);
 place_above(obj02,obj26); release(obj02); grasp(obj04);
 hold_above(obj04,obj02); follow_traj_pour(obj04,obj02)}
```

Note that the actual sequence does not directly interact with the liquid (obj16)—the only object specified by the task—but rather with a container of liquid (obj04), an empty cup (obj02), and a table (obj26), while none of these objects are specified in the task arguments. As seen in this example, the input for our planning problem is under-specified.

For evaluation, we prepared a dataset where the goal was to produce correct sequences for the following tasks in different environments:

- `stir(A)`: Given a liquid A, the robot has to identify a stirrer of ideal size (from several) and stir with it. The liquid may be located on a tight shelf where it would be dangerous to stir the liquid, and the robot should always stir it on top of an open surface, like a table. The robot should always only interact with the container of the liquid, rather than the liquid itself, whenever liquid needs to be carried or poured. Our learning algorithm should learn such properties.
- `pick_and_place(A,B)`: The robot has to place A on top of B. If A is under some other object C, the object C must first be moved before interacting with object A.
- `pour(A)`: The robot has to identify a bowl-like object without object labels and pour liquid A into it. Note again that liquid A cannot be directly

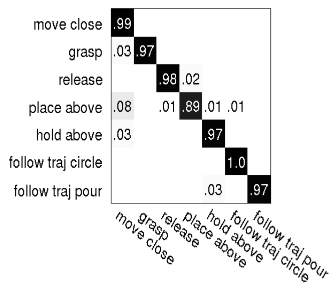
interacted with, and it should not be poured on top of a shelf.

- `pour_to(A,B)`: The liquid A has to be poured into the container B. (A variant of the previous task where the container B is specified but the model should be able to distinguish two different tasks.)
- `throw_away(A)`: The robot has to locate a garbage can in the environment and throw out object A.

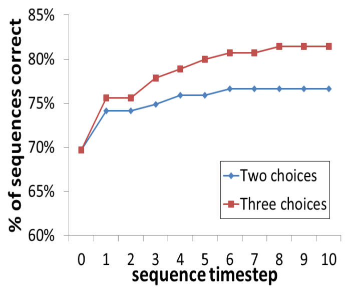
In order to learn these programs, we collected 127 sequences for 113 unique scenarios by presenting participants the environment in simulation and the task to be done. We considered a single-armed mobile manipulator robot for these tasks. In order to extract information about the environment at each time frame of every sequence, we implemented each primitive using OpenRAVE simulator [26]. Though most of the scenarios had a single optimal sequence, multiple sequences were introduced when there were other acceptable variations. The length of each sequence varies from 4 steps to 10 steps, providing a total of 736 instances of primitives. To ensure variety in sequences, sequences were generated based on the 13 different environments shown in Figure 3.3, using 43 objects each with unique attributes.

Baseline Algorithms. We compared our model against following baseline algorithms:

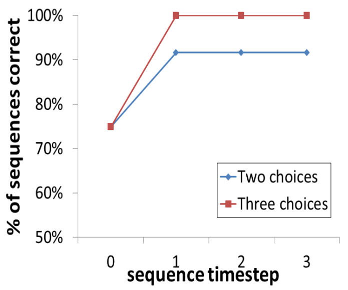
- *chance*: At each time step, a primitive and its arguments are selected at random.
- *multiclass*: A multiclass SVM [62] was trained to predict primitives without arguments, since the set of possible arguments changes depending on the environment.



(a) **Confusion matrix** for the seven primitives in our dataset. Our dataset consist of 736 instances of seven primitives in 127 sequences on five manipulation tasks.



(b) **Percentage of programs correct.** Without any feedback in completely autonomous mode, the accuracy is 69.7%. With feedback (number of feedbacks on x-axis), the performance increases. This is on full 127 sequence dataset.



(c) **Percentage of programs correct for 12 high-level tasks** such as making sweet tea. In completely autonomous mode, the accuracy is 75%. With feedback (number of feedbacks on x-axis), the performance increases.

Figure 3.4: **Results with cross-validation.** (a) On predicting the correct primitive individually. (b) On predicting programs, with and without user intervention. (c) On performing different tasks with the predicted sequences.

- *symbolic-plan-svm*: A PDDL-based symbolic planner [183, 190] requires a domain and a problem definition. Each scenario was translated to symbolic entities and relations. However, the pre-conditions and effects of each action in domain definition were hand-coded, and each object was labeled with attributes using predicates. Unlike our model that works on an under-specified problem, each symbolic planning problem requires an explicit goal state. In order to define these goal states, we have trained ranking SVMs [63] in order to detect a 'stirrer', an 'object to pour into' and a 'garbage can' for `stir`, `pour`, and `throw_away`, respectively. Each symbolic planning instance was then solved by reducing to a satisfiability problem [67, 128].
- *symbolic-plan-manual*: Based on the same method as *symbolic-plan-svm*, instead of training ranking SVMs, we provided ground-truth goal states.

Even after providing lots of hand-coded rules, it is still missing some rules due to the difficulty of representation using PDDL [183, 190], These missing rules include the fact that liquid needs to be handled through its container and that objects should not be manipulated on top of the shelf.

Evaluation and Results. We evaluated our algorithm through *6-fold cross-validation*, computing accuracies over primitives, over primitives with arguments, and over the full sequences. Figure 3.4a shows the confusion matrix for prediction of our seven primitives. We see that our model is quite robust for most primitives.

With our dataset, our model was able to correctly predict pairs of primitives and arguments 90.0% of the time and full sequences 69.7% of the time (Table 3.1). Considering only the primitives without arguments, it was able to predict primitive 96.7% of the time and full sequence 91.6% of the time. The last column of Table 3.1 shows the performance with respect to whether the complete sequence was correct or not. For example, for "pouring", our model has not only learned to bring a cup over to the table, but also learned to choose a cup when there are multiple other objects like a pot, a bowl, or a can that may have similar properties.

How do baselines perform for our under-specified planning problem? The results of various baseline algorithms are shown in Table 3.1. If the primitive and arguments pairs are predicted at random, none of the sequences would be correct because of the large search space of arguments. *Multiclass* baseline algorithm, which is only capable of predicting next primitives without filling in the arguments, was not able to even correctly identify the correct next primitive, especially for more complex primitives with two arguments.

The symbolic planner based approaches, *symbolic-plan-svm* and *symbolic-plan-manual*, suffered greatly from under-specified nature of the problem. The planners predicted correctly 49.6% and 54.9% of the times, respectively, compared to our model's performance of 69.7%. Even though both planners made use of heavily hand-coded domain definitions of the problem, due to the nature of the language used by symbolic planners, rules such as that liquid should not be handled on top of shelves were not able to be encoded. Even if the language were capable of encoding these rules, it would require a human expert in planning language to carefully encode every single rule the expert can come up with.

Also, by varying the set of features, it is evident that without very robust primitive-level accuracies, the models are unable to construct a single correct sequence.

How important is attribute representation of objects? For 113 unique scenarios in our dataset, we have randomly flipped binary attributes and observed the effects of detection errors on correctness for the full sequence (Figure 3.6). When there is no error in detecting attributes, our model performs at 69.7%. With 10% detection error, it performs at 55.8%, and with 40% detection errors, it performs at 38.1%. Since the attribute detection is more reliable than the object detection [32, 30, 81], our model will perform better than planners based on explicit object labels.

How can the robot utilize learned programs? These learned programs can form higher level tasks such as making a recipe found online. For example, serving sweet tea would require the following steps: pouring tea into a cup, pouring sugar into a cup, and stirring it (Figure 3.5). We have tested each of the four

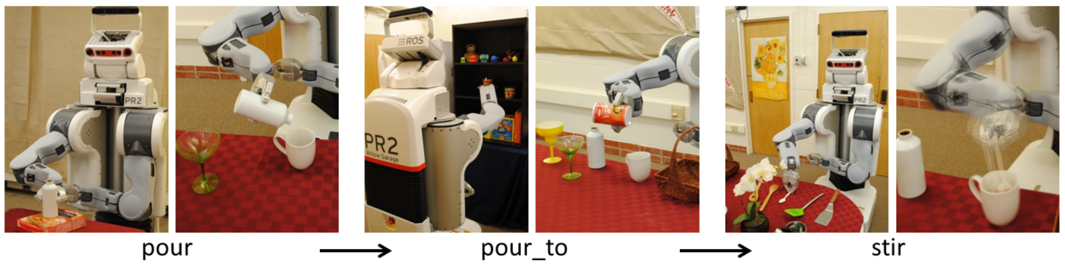


Figure 3.5: **Few snapshots of learned sequences** forming the higher level task of serving sweet tea, which takes the sequence of pouring tea into a cup, pouring sugar into a cup, and then stirring it.

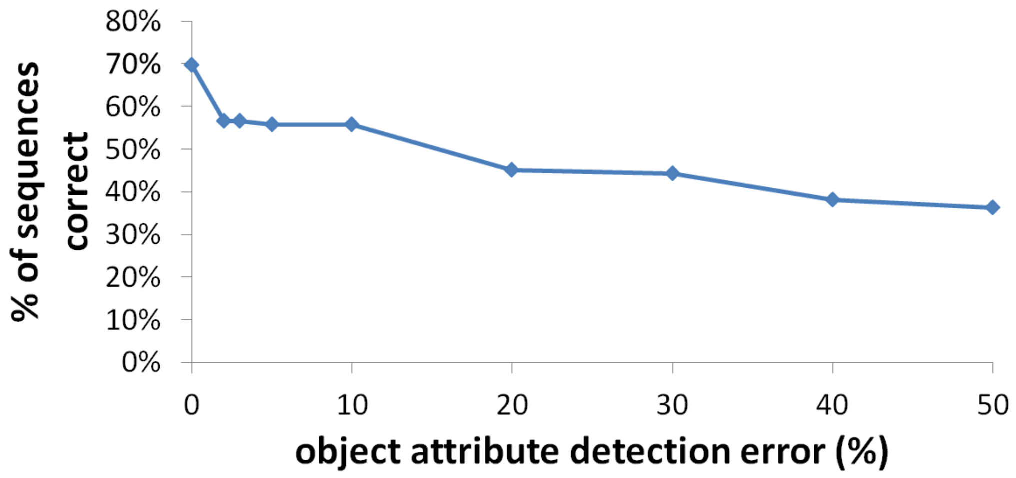


Figure 3.6: **Effect of attribute perception error.** Figure showing percentage of programs correct with attribute labeling errors for binary attributes. For 113 unique scenarios, binary attributes were randomly flipped.

tasks, *serve-sweet-tea*, *serve-coffee-with-milk*, *empty-container-and-throw-away*, and *serve-and-store*, in three environments. Each of the four tasks can be sequenced in following manner by programs respectively: `pour` → `pour_to` → `stir`, `pour_to` → `pour_to`, `pour` → `throw_away`, and `pour` → `pick_and_place`. Except few unsuccessful scenarios that were due to incorrect choice of arguments for part of the sequence, our model was able to successfully complete the full task for 9 scenarios out of total 12 scenarios.

Does the robot need a human observer? In an assistive robotics setting, a robot will often be accompanied by a human observer. With help from the human, performance can be greatly improved. Instead of choosing a primitive and argument pair that maximizes the discriminant function, the robot can present the top 2 or 3 primitive and argument pairs to the observer, who can simply give feedback on the best option among those choices. Such feedback not only gives more training data for robots to train on but also prevents robots from causing damage to the environment.

At the initial time step of the sequence, with only a single piece of feedback, given 2 or 3 choices, performance improves to 74.1% and 75.6% respectively from 69.7% (Figure 3.4b). If feedback was provided through whole sequence with the top 2 or 3 choices, it further improves to 76.7% and 81.4%. Furthermore, the four higher level tasks (recipes) considered earlier also shows that with a single feedback at the initial time step of each program, the results improve from 75% to 100% (Figure 3.4c).

Robotic Experiments. Finally, we demonstrate that our inferred programs can be successfully executed on our Kodiak PR2 robot for a given task in an environment. Using our implementation of the primitives discussed in Section 3.4, we show our robot performing the task of "serving sweet tea." It comprises executing three programs in series – `pour`, `pour_to` and `stir` – which in total required sequence of 20 primitives with correct arguments. Each of these programs (i.e., the sequence of primitives and arguments) is inferred for this environment. Figure 3.1 shows a few snapshots and the full video is available at: `http://pr.cs.cornell.edu/learningtasksequences`

3.5 Conclusion

When modeling tasks that involves objects in complex environments, context of various objects plays a important role. In this chapter, we considered the problem of learning sequences of controllers for robots in unstructured human environments. In an unstructured environment, even a simple task such as pouring can take variety of different sequences of controllers depending on the configuration of the environment. We took a dynamic planning approach, where we represent the current state of the environment using a set of attributes. To ensure that our dynamic plans are as general and flexible as possible, we designed a score function that captures relations between task, environment, primitives, and their arguments, and we trained a set of parameters weighting the various attributes from example sequences. By unrolling the program, we can obtain a Markov Random Field style representation, and use a maximum margin learning strategy. We demonstrated on a series of example sequences that our approach can effectively learn dynamic plans for various complex high-level tasks.

CHAPTER 4

DEEP MULTIMODAL EMBEDDING

In order to model tasks that involves objects in complex environments, a robot has to be able to reason with different modalities. In earlier chapters, we considered modeling tasks and activities based on sensor inputs and attributes of objects which may come from many different sources. In this chapter, we discuss a learning algorithm that builds a representation for interactions (object manipulation tasks) by mapping different modalities of visual, natural language, and complex motions into a shared representation.

Consider a robot manipulating a new appliance in a home kitchen, *e.g.* a toaster or a juicer. The robot must use the combination of its observations of the world and natural language instructions to infer how to manipulate objects. Such ability to fuse information from different input modalities and map them to actions is extremely useful to many applications of household robots [148], including assembling furniture, cooking recipes, and many more.

Even though similar concepts might appear very differently in different sensor modalities, humans are able to understand that they map to the same concept. For example, when asked to "turn the knob counter-clockwise" on a toaster, we are able to correlate the instruction language and the appearance of a knob on a toaster with the motion to do so. We also associate this concept more closely with a motion which would incorrectly rotate in the opposite direction than with, for example, the motion to press the toaster's handle downwards. There is strong evidence that we are able to correlate between different modalities through *common representations* [29].

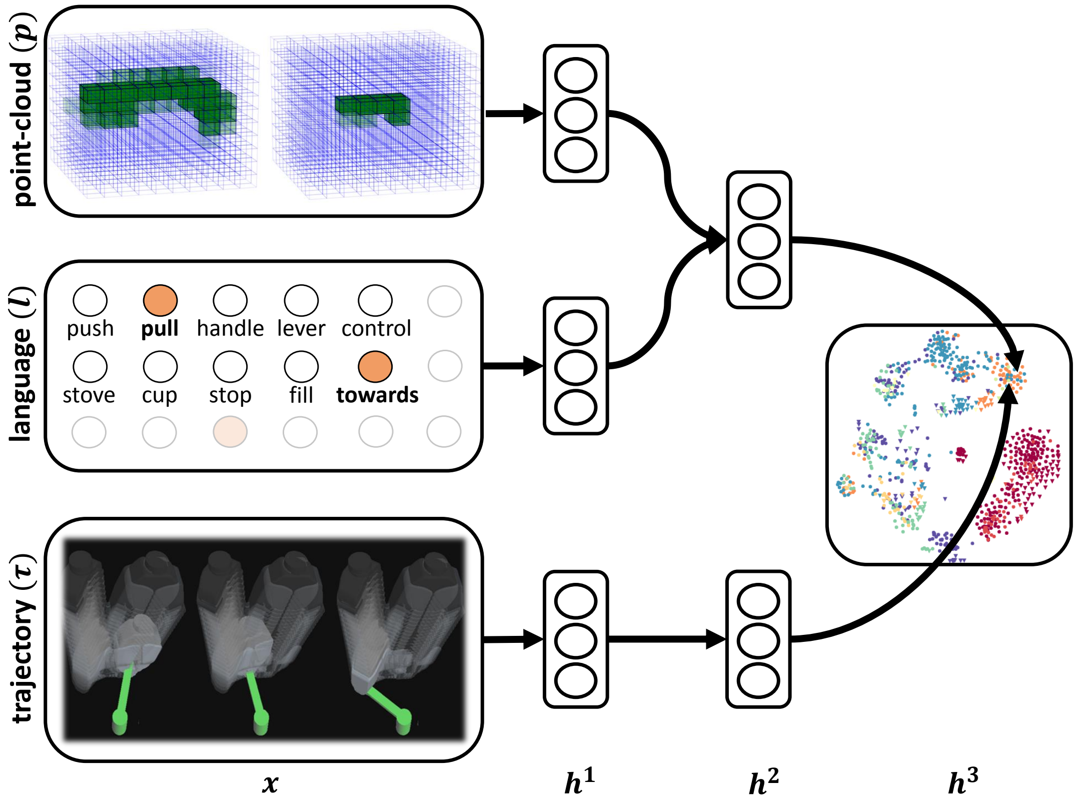


Figure 4.1: **Deep Multimodal Embedding:** Our deep neural network learns to embed both point-cloud/natural language instruction combinations and manipulation trajectories in the same semantically meaningful space, where distance represents the relevance of embedded data.

Obtaining a good common representation between different modalities is challenging for two main reasons. First, each modality might intrinsically have very different statistical properties – for example, here our trajectory representation is inherently dense, while our representation of language is naturally sparse. This makes it challenging to apply algorithms designed for unimodal data. Second, even with expert knowledge, it is extremely challenging to design joint features between such disparate modalities. Designing features which map different sensor inputs and actions to the same space, as required here, is particularly challenging.

In this work, we use a deep neural network to learn a shared embedding between the pairing of object parts in the environment with natural language instructions, and manipulation trajectories (Figure 4.1). This means that all three modalities are projected to the *same* feature space. We introduce an algorithm that learns to pull semantically similar environment/language pairs and their corresponding trajectories to the same regions, and push environment/language pairs away from irrelevant trajectories based on how irrelevant they are.

In the past, deep learning methods have shown impressive results for learning features for a wide variety of domains [76, 142, 47] and even learning cross-domain embeddings [144]. In contrast to these existing methods, here we present a new pre-training algorithm for initializing networks to be used for joint embedding of different modalities. Our algorithm trains each layer to map similar cases to similar areas of its feature space, as opposed to other methods which either perform variational learning [50] or train for reconstruction [44].

In Chapter 5, we validate our approach on a large manipulation dataset [148] and perform a series of robotic experiments with our learned embedding space. In summary, the key contributions of this chapter are:

- We present an algorithm which learns an semantically meaningful embedding space by enforcing a varying and loss-based margin.
- We present an algorithm for unsupervised pre-training of multi-modal features to be used for embedding which outperforms standard pre-training algorithms [44].

4.1 Related Work

Metric Embedding. Several works in machine learning make use of the power of shared embedding spaces. LMNN [174] learns a max-margin Mahalanobis distance for a unimodal input feature space. Weston et al. [176] learn linear mappings from image and language features to a common embedding space for automatic image annotation. Moore et al. [104] learn to map songs and natural language tags to a shared embedding space. However, these approaches learn only a shallow, linear mapping from input features, whereas here we learn a deep non-linear mapping which is less sensitive to input representations.

Deep Learning for Multimodal Data. In recent years, deep learning algorithms have enjoyed huge successes, particularly in the domains of computer vision and natural language processing (e.g. [76, 142]). In robotics, deep learning has previously been successfully used for detecting grasps for novel objects in multi-channel RGB-D images [84] and for classifying terrain from long-range vision [47].

Ngiam et al. [111] use deep learning to learn features incorporating both video and audio modalities. Sohn et al. [143] propose a new generative learning algorithm for multimodal data which improves robustness to missing modalities at inference time. In these works, a single network takes all modalities as inputs, whereas here we perform joint embedding of multiple modalities using multiple networks.

Deep Learning for Joint Embedding. Several previous works use deep networks for joint embedding between different feature spaces. Mikolov et al. [99] map different languages to a joint feature space for translation. Srivastava and Salakhutdinov [144] map images and natural language “tags” to the same space for automatic annotation and retrieval. While these works use conventional pre-training algorithms, here we present a new pre-training approach for learning embedding spaces and show that it outperforms these existing methods (Sec. 5.5.3.) Our algorithm trains each layer to map similar cases to similar areas of its feature space, as opposed to other methods which either perform variational learning [50] or train for reconstruction [44].

Hu et al. [53] also use a deep network for metric learning for the task of face verification. Similar to LMNN [174], Hu et al. [53] enforces a constant margin between distances among inter-class objects and among intra-class objects. In Sec. 5.5.3, we show that our approach, which uses a loss-dependent variable margin, produces better results for our problem. Our work builds on deep neural network to embed three different modalities of point-cloud, language, and trajectory into shared embedding space while handling lots of label-noise originating from crowd-sourcing.

4.2 Deep Multimodal Embedding: Our Approach

The main challenge of our work is to learn a model which maps three disparate modalities – point-clouds, natural language, and trajectories – to a single semantically meaningful space. In particular, we focus on point-clouds of object parts, natural language instructing manipulation of different objects, and trajectories that would manipulate these objects.

We introduce a method that learns a common point-cloud/language/trajectory embedding space in which the projection of a task (point-cloud/language combination) should higher similarity to projections of relevant trajectories than task-irrelevant trajectories. Among these irrelevant trajectories, some might be less relevant than others, and thus should be pushed further away.

For example, given a door knob that needs to be grasped normal to the door surface and an instruction to rotate it clockwise, a trajectory that correctly approaches the door knob but rotates counter-clockwise should have higher similarity to the task than one which approaches the knob from a completely incorrect angle and does not execute any rotation.

We learn non-linear embeddings using a deep learning approach, as shown in Fig. 4.1, which maps raw data from these three different modalities to a joint embedding space. Prior to learning a full joint embedding of all three modalities, we pre-train embeddings of subsets of the modalities to learn semantically meaningful embeddings for these modalities.

We show in Sec. 5.4 that learned joint embedding space can be efficiently used for finding an appropriate manipulation trajectory for object parts with natural language instruction.

4.2.1 Network Architecture

To solve this problem of learning to manipulate novel objects and appliance as defined in equation (5.1), we learn two different mapping functions that map to a common space—one from a point-cloud/language pair and the other from a trajectory. More formally, we want to learn $\Phi_{\mathcal{P},\mathcal{L}}(p, l)$ and $\Phi_{\mathcal{T}}(\tau)$ which map to a joint feature space $\mathbb{R}^M$:

$$\Phi_{\mathcal{P},\mathcal{L}}(p,l) : (\mathcal{P},\mathcal{L}) \to \mathbb{R}^M$$

$$\Phi_{\mathcal{T}}(\tau) : \mathcal{T} \to \mathbb{R}^M$$

Here, we represent these mappings with a deep neural network, as shown in Figure 4.1.

The first, $\Phi_{\mathcal{P},\mathcal{L}}$, which maps point-clouds and trajectories, is defined as a combination of two mappings. The first of these maps to a joint point-cloud/language space $\mathbb{R}^{N_{2,pl}}$ — $\Phi_{\mathcal{P}}(p) : \mathcal{P} \to \mathbb{R}^{N_{2,pl}}$ and $\Phi_{\mathcal{L}}(l) : \mathcal{L} \to \mathbb{R}^{N_{2,pl}}$. Once each is mapped to $\mathbb{R}^{N_{2,pl}}$, this space is then mapped to the joint space shared with trajectory information: $\Phi_{\mathcal{P},\mathcal{L}}(p,l) : ((\mathcal{P},\mathcal{L}) \to \mathbb{R}^{N_{2,pl}}) \to \mathbb{R}^M$.

We use two separate multi-layer deep neural networks, one for $\Phi_{\mathcal{P},\mathcal{L}}(p,l)$ and one for $\Phi_{\mathcal{T}}(\tau)$. Take N_p as the size of point-cloud input p, N_l as similar for natural language input l, $N_{1,p}$ and $N_{1,l}$ as the number of hidden units in the first hidden layers projected from point-cloud and natural language features, respectively, and $N_{2,pl}$ as the number of hidden units in the combined point-cloud/language layer. With W's as network weights, which are the learned parameters of our system, and $a(\cdot)$ as a rectified linear unit (ReLU) activation function [188], our model for projecting from point-cloud and language features to the shared embedding h^3 is as follows:

$$\begin{aligned}
h_i^{1,p} &= a\left(\sum_{j=0}^{N_p} W_{i,j}^{1,p} p_j\right)\\
h_i^{1,l} &= a\left(\sum_{j=0}^{N_l} W_{i,j}^{1,l} l_j\right)\\
h_i^{2,pl} &= a\left(\sum_{j=0}^{N_{1,p}} W_{i,j}^{2,p} h_j^{1,p} + \sum_{j=0}^{N_{1,l}} W_{i,j}^{2,l} h_j^{1,l}\right)\\
h_i^{3} &= a\left(\sum_{j=0}^{N_{2,pl}} W_{i,j}^{3,pl} h_j^{2,pl}\right)
\end{aligned}$$

The model for projecting from trajectory input τ is similar, except it takes input only from a single modality.

4.3 Learning Deep Multimodal Embedding of Point-cloud, Language and Trajectory

In our joint feature space, proximity between two mapped points should reflect how relevant two data-points are to each other, even if they are from completely different modalities. Such property should hold in both *inter-modal* distances between embeddings of different modalities and *intra-modal* distances between embeddings of same modalities,

We train our network to bring demonstrations that manipulate a given object according to some language instruction closer to the mapped point for that object/instruction pair, and to push away demonstrations that would not correctly manipulate that object. Trajectories which have no semantic relevance to the object are pushed much further away than trajectories that have some relevance, even if the latter would not manipulate the object according to the instruction.

For every training point-cloud/language pair (p_i, l_i), we have two sets of demonstrations: a set of trajectories $\mathcal{T}_{i,S}$ that are relevant (similar) to this task and a set of trajectories $\mathcal{T}_{i,D}$ that are irrelevant (dissimilar) as described in Sec. 5.2.4.

Inter-modal Constraints. Trajectories that are appropriate for the task (point-cloud/language) should be closer in embedding space than trajectories that are not appropriate for the task. For each pair of (p_i, l_i), we want all projections of $\tau_j \in \mathcal{T}_{i,S}$ to have higher similarity to the projection of (p_i, l_i) than $\tau_k \in \mathcal{T}_{i,D}$.

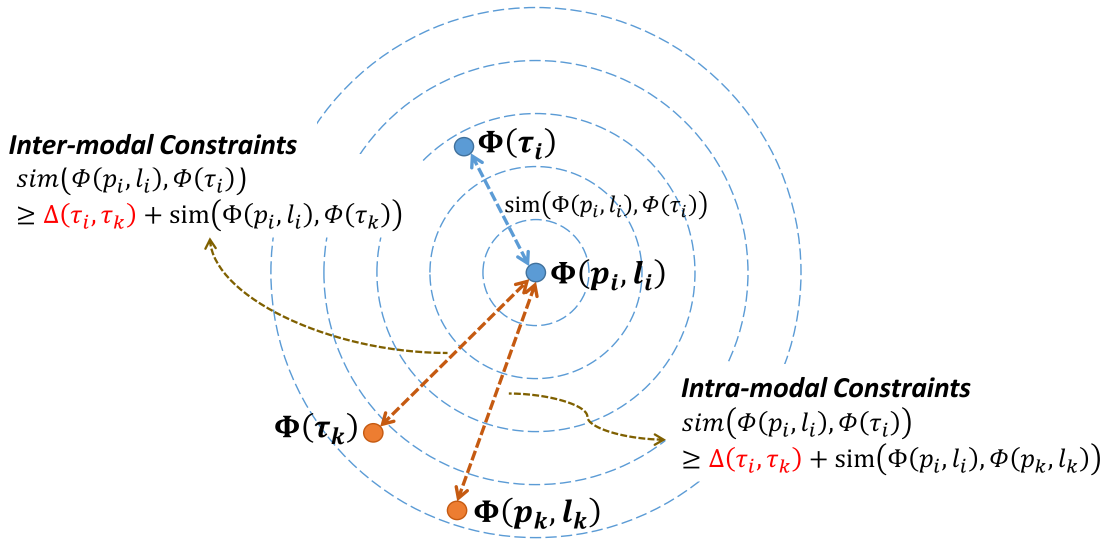

Inter-modal Constraints
$sim(\Phi(p_i, l_i), \Phi(\tau_i)) \geq \Delta(\tau_i, \tau_k) + \mathrm{sim}(\Phi(p_i, l_i), \Phi(\tau_k))$
$\Phi(\tau_i)$
$\mathrm{sim}(\Phi(p_i, l_i), \Phi(\tau_i))$
$\Phi(p_i, l_i)$
$\Phi(\tau_k)$
$\Phi(p_k, l_k)$
Intra-modal Constraints
$sim(\Phi(p_i, l_i), \Phi(\tau_i)) \geq \Delta(\tau_i, \tau_k) + \mathrm{sim}(\Phi(p_i, l_i), \Phi(p_k, l_k))$

Figure 4.2: The proximity between two mapped points should reflect how relevant two data-points are to each other, even if they are from completely different modalities. We train our network to bring demonstrations that manipulate a given object according to some language instruction closer to the mapped point for that object/instruction pair, and to push away demonstrations that would not correctly manipulate that object.

A simple approach would be to train the network to distinguish these two sets by enforcing a finite distance (safety margin) between the similarities of these two sets [174], which is written in the form of a constraint:

$$sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_j)) \geq 1 + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_k))$$

Rather than simply being able to distinguish two sets, we want to learn semantically meaningful embedding spaces from different modalities. Recalling our earlier example where one incorrect trajectory for manipulating a door knob was much closer to correct than another, it is clear that our learning algorithm should drive some of incorrect trajectories to be more dissimilar than others. The difference between the similarities of τ_j and τ_k to the projected point-cloud/language pair (p_i, l_i) should be at least the loss $\Delta(\tau_j, \tau_k)$. This can be written as a form of a constraint:

$$\forall \tau_j \in \mathcal{T}_{i,S}, \forall \tau_k \in \mathcal{T}_{i,D}$$
$$sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_j)) \geq \Delta(\tau_j, \tau_k) + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_k))$$

Intuitively, this forces trajectories with higher loss (Δ) from the ground truth to embed further than those with lower distance. Enforcing all combinations of these constraints could grow exponentially large. Instead, similar to the cutting plane method for structural support vector machines [163], we find the most violating trajectory $\tau' \in \mathcal{T}_{i,D}$ for each training pair of $(p_i, l_i, \tau_i \in \mathcal{T}_{i,S})$ at each iteration. The most violating trajectory has the highest similarity augmented with the loss scaled by a constant α:

$$\tau_i' = \underset{\tau \in \mathcal{T}_{i,D}}{\text{argmax}}(sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau)) + \alpha\Delta(\tau_i, \tau))$$

The cost of our deep embedding space h^3 is computed as the hinge loss of the most violating trajectory.

$$L_{h^3}(p_i, l_i, \tau_i) = |\Delta(\tau_i', \tau_i) + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_i')) - sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_i))|_+$$

The cost of inter-modal constraints for our embedding space h^3 is computed as the hinge loss of the most violating trajectory:

$$L_{h^3}^{inter} = |\Delta(\tau_i', \tau_i) + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_i')) - sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_i))|_+ \tag{4.1}$$

Intra-modal Constraints. Similarly, the proximity between embeddings of same modalities should also reflect semantic relevance between the points. Since we have most clear definition of semantic relevance between trajectories, we constrain the point-cloud/language pairs (p_k, l_k) associated with the irrelevant trajectories $(\tau_k \in \mathcal{T}_{i,D})$ should be at least be $\Delta(\tau_j, \tau_k)$. Thus, in the form of constraint:

$$\forall \tau_j \in \mathcal{T}_{i,S}, \forall \tau_k \in \mathcal{T}_{i,D}$$
$$sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_j)) \geq \Delta(\tau_j, \tau_k) + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{P},\mathcal{L}}(p_k, l_k))$$

Fig. 4.2 shows visualizations of inter-modal and intra-modal constraints.

Again, we find the most violated point-cloud/language pair (p'_i, l'_i) and its associated trajectory (τ'_i).

$$L_{h^3}^{intra} = |\Delta(\tau'_i, \tau_i) + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{P},\mathcal{L}}(p'_i, l'_i)) - sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_i))|_+ \quad (4.2)$$

Thus, the cost of our deep embedding space h^3 is computed as summation of inter-modal costs and intra-modal costs:

$$L_{h^3} = L_{h^3}^{inter} + L_{h^3}^{intra}$$

The average cost of each minibatch is back-propagated through all the layers of the deep neural network using the AdaDelta [187] algorithm.

4.3.1 Pre-training Joint Point-cloud/Language Model

One major advantage of deep learning methods is the use of unsupervised pre-training to initialize neural network parameters to a good starting point before the final supervised fine-tuning stage. Pre-training helps these high-dimensional networks to avoid overfitting to the training data.

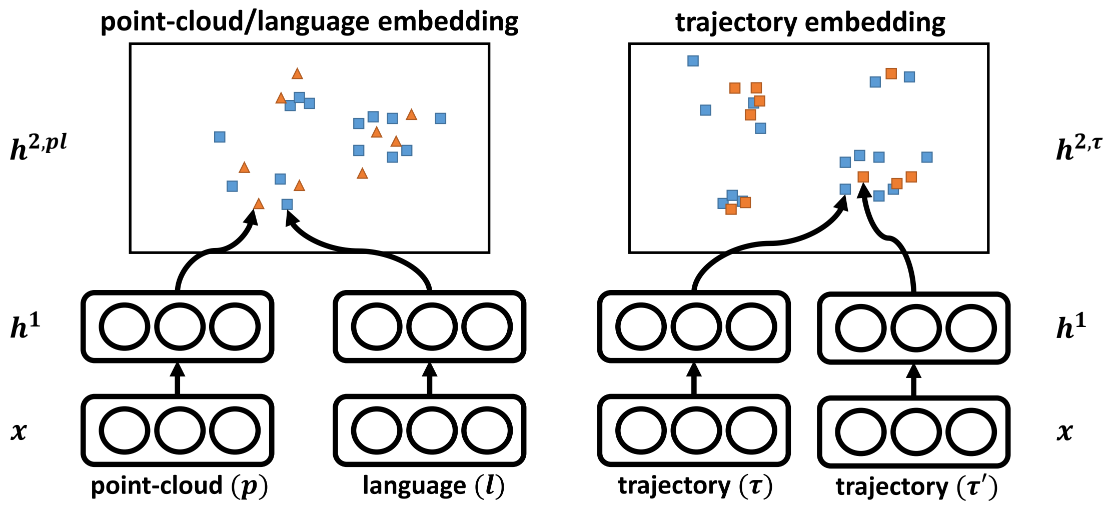


Figure 4.3: **Pre-training lower layers:** Visualization of our pre-training approaches for $h^{2,pl}$ and $h^{2,\tau}$. For $h^{2,pl}$, our algorithm pushes matching point-clouds and instructions to be more similar. For $h^{2,\tau}$, our algorithm pushes trajectories with higher DTW-MT similarity to be more similar.

Our lower layers $h^{2,pl}$ and $h^{2,\tau}$ represent features extracted exclusively from the combination of point-clouds and language, and from trajectories, respectively. Our pre-training method initializes $h^{2,pl}$ and $h^{2,\tau}$ as semantically meaningful embedding spaces similar to h^3, as shown later in Section 5.5.3.

First, we pre-train the layers leading up to these layers using spare denoising autoencoders [170, 188]. Then, our process for pre-training $h^{2,pl}$ is similar to our approach to fine-tuning a semantically meaningful embedding space for h^3 presented above, except now we find the most violating language l' while still relying on a loss over the associated optimal trajectory:

$$l' = \underset{l \in \mathcal{L}}{\operatorname{argmax}}(sim(\Phi_{\mathcal{P}}(p_i), \Phi_{\mathcal{L}}(l)) + \alpha \Delta(\tau, \tau_i^*))$$

$$L_{h^{2,pl}}(p_i, l_i, \tau_i) = |\Delta(\tau_i, \tau') + sim(\Phi_{\mathcal{P}}(p_i), \Phi_{\mathcal{L}}(l')) - sim(\Phi_{\mathcal{P}}(p_i), \Phi_{\mathcal{L}}(l_i))|_+$$

Notice that although we are training this embedding space to project from point-cloud/language data, we guide learning using trajectory information.

After the projections $\Phi_{\mathcal{P}}$ and $\Phi_{\mathcal{L}}$ are tuned, the output of these two projections are added to form the output of layer $h^{2,pl}$ in the final feed-forward network.

4.3.2 Pre-training Trajectory Model

For our task of inferring manipulation trajectories for novel objects, it is especially important that similar trajectories τ map to similar regions in the feature space defined by $h^{2,\tau}$, so that trajectory embedding $h^{2,\tau}$ itself is semantically meaningful and they can in turn be mapped to similar regions in h^3. Standard pretraining methods, such as sparse de-noising autoencoder [170, 188] would only pre-train $h^{2,\tau}$ to reconstruct individual trajectories. Instead, we employ pre-training similar to Sec. 4.3.1, except now we pre-train for only a single modality – trajectory data.

As shown on right hand side of Fig. 4.3, the layer that embeds to $h^{2,\tau}$ is duplicated. These duplicated embedding layers are treated as if they were two different modalities, but all their weights are shared and updated simultaneously. For every trajectory $\tau \in \mathcal{T}_{i,S}$, we can again find the most violating $\tau' \in \mathcal{T}_{i,D}$ and the minimize a similar cost function as we do for $h^{2,pl}$.

In the following chapter, we demonstrate how learned model enables robots to perform complex tasks of manipulating novel objects utilizing information from different modalities.

CHAPTER 5

LEARNING TO MANIPULATE NOVEL OBJECTS VIA DEEP MULTIMODAL EMBEDDING

The previous chapter has introduced a model that can map different modalities of information into a same representation. In this chapter, we focus on using such learned representation to model sequences of complex motions for the task of manipulating novel objects in complex environments given a visual and a natural language instruction of an object.

Consider the espresso machine in Figure 5.1—even without having seen this machine before, a person can prepare a cup of latte by reading an instruction manual and visually observing the machine. This is possible because humans have vast prior experience with manipulating differently-shaped objects. While robots can perform increasing more complex tasks (e.g. [88, 85]), it is un-scalable and infeasible for roboticists to program the exact sequence for a every variety of objects in human environments.

In this work, we focus on answering the following question: *can a robot manipulate novel objects by learning to transfer manipulation trajectories from completely different objects?* Our goal is to enable robots to generalize their manipulation ability to a large number of tasks on novel objects ranging from toaster, sink, water fountain, soda dispenser to toilets.

The key idea of our work is that objects designed for use by humans share many *similarly-operated object parts* such as handles, levers, nozzles, and buttons; thus, manipulation motions can be transferred even between completely different objects if we represent these motions with respect to these parts. For exam-

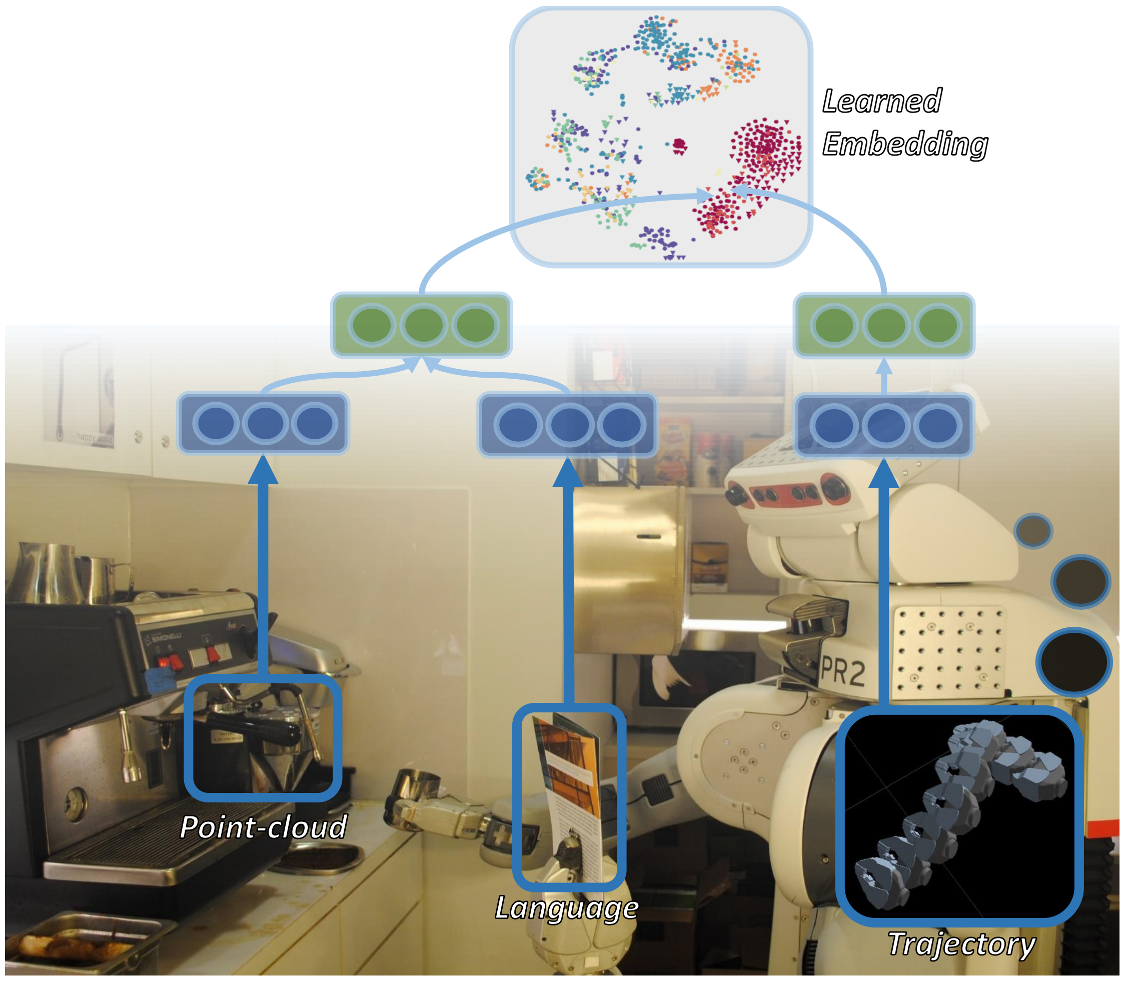


Figure 5.1: **First encounter of an espresso machine** by our PR2 robot. Without ever having seen the machine before, given language instructions and a point-cloud from Kinect sensor, our robot is capable of finding appropriate manipulation trajectories from prior experience using our deep multimodal embedding model.

ple, even if the robot has never seen an espresso machine before, it should be able to manipulate it if it has previously seen similarly-operated parts of other objects such as a urinal, soda dispenser, or restroom sink, as illustrated in Figure 5.2.

Classification of objects or object parts (e.g. 'handle') alone does not provide enough information for robots to actually manipulate them, since semantically-similar objects or its parts are often operated completely differently—consider, for example, manipulating the ‘handle’ of a urinal, as opposed to the ‘handle’ of a door. While object parts that are operated in similar fashion are often referred by different names, there exists similarity in their shapes that allows motions to be transferred between completely different objects. Thus, rather than relying on scene understanding techniques [12, 90, 41], we directly use point-clouds for manipulation planning using machine learning algorithms.

We use deep neural networks to learn a shared embedding space between the object parts point-cloud, natural language instructions, and manipulation trajectories (Figure 4.1), which in turn is used to identify appropriate manipulation trajectories from prior experience to transfer to novel objects. Our algorithm also allows for efficient inference because, given a new instruction and point-cloud, we only need to find the nearest trajectory to the projection of this pair in the learned embedding space using a fast nearest-neighbor algorithms [105].

To train our joint embedding, we need scalable methods of collecting thousands of demonstrations on more than a hundred different objects. To this end, we develop Robobarista platform, a crowd-sourcing platform that allows any person on the web to teach a robot by simply dragging the robot end-effector in 3D visualizer. With our noise handling algorithm, our model trained with crowd-sourced demonstrations outperforms the model trained with expert demonstrations, even with the significant amount of noise in crowd-sourced manipulation demonstrations.

We evaluate our approach on a large dataset of *116 objects* with *250 natural* *language instructions* for which there are *1225 crowd-sourced manipulation trajectories* from 71 non-expert users. Our results show that deep multimodal embedding algorithm outperforms expert-designed feature-based models and other deep learning approaches. We also perform 100 fully autonomous end-to-end experiments on PR2 robot, showing that our approach allows robot to actually manipulate appliances it has never seen before. Finally, we further test our hypothesis of object part-based transfer of manipulation trajectories to even prepare a cup of latte.

In summary, the key contributions of this chapter are:

- We present a novel approach of *part-based transfer* between different objects for manipulation planning of novel objects.
- We present an online *crowd-sourcing* platform which allows us to easily scale collection of manipulation demonstrations.
- We introduce a large-scale manipulation dataset and evaluate our approach on this dataset, showing significant improvements over other state-of-the-art methods.

5.1 Related Work

Improving robotic perception and teaching manipulation strategies to robots has been a major research area in recent years. In this section, we describe related work in various aspects of learning to manipulate novel objects.

Scene Understanding. In recent years, there has been significant research focus on semantic scene understanding [90, 72, 76, 180], human activity detection

[146, 54] as we have shown in Chapter 2, and features for RGB-D images and point-clouds [141, 80]. Similar to our idea of using part-based transfers, the deformable part models [41, 42] are effective in object detection. However, when robot encounters object it has not manipulated before, classification of objects or its parts alone does not provide enough information for a robot to reliably plan manipulation. Even a simple category such as 'kitchen sinks' or 'handle' has a huge amount of variation in how different instances are manipulated – for example, depending on the brand and the model of a kitchen sink, each 'handle' requires very different strategies such as pulling the handle upwards, pushing downwards, pushing sideways, and so on. Instead of classifying object, direct perception approaches [39, 77] perceive affordances based on the shape of the object. These works focus on detecting an object part given an object with known affordance for motions such as 'pour', while we focus on predicting the correct motion of novel objects with just an instruction manual.

Manipulation Strategy. Many works in robotic manipulation focus on task-specific manipulation of *known* objects—for example, baking cookies with known tools [14] and folding the laundry [100] – or focus on learning specific motions such as grasping [68] and opening doors [28]. Others [147, 101] focus on sequencing manipulation tasks assuming perfect manipulation primitives such as *grasp* and *pour* are available. Instead, here, we use learning to generalize to manipulating *novel* objects never seen before by the robot, without relying on preprogrammed motion primitives.

For the more general task of manipulating new instances of objects, previous approaches rely on finding articulation [145, 121] or using interaction [66], but they are limited by tracking performance of a vision algorithm. Many objects that humans operate daily have small parts such as 'knobs', which leads to significant occlusion as manipulation is demonstrated. Another approach using part-based transfer between objects has been shown to be successful for grasping [22, 25]. We extend this approach and introduce a deep learning model that enables part-based transfer of *trajectories* by automatically learning relevant features. Our focus is on the generalization of manipulation trajectory via part-based transfer using point-clouds without knowing objects a priori and without assuming any of the sub-steps ('approach', 'grasping', and 'manipulation').

A few recent works use deep learning approaches for robotic manipulation. Levine et al. [87] use a Gaussian mixture model to learn system dynamics, then use these to learn a manipulation policy using a deep network. Lenz et al. [85] use a deep network to learn system dynamics for real-time model-predictive control. Both these works focus on learning low-level controllers, whereas here we learn high-level manipulation trajectories.

Learning from Demonstration. Several successful approaches for teaching robots tasks, such as helicopter maneuvers [2] or table tennis [107], have been based on Learning from Demonstration (LfD) [5]. Although LfD allows end users to demonstrate a manipulation task by simply taking control of the robot's arms, it focuses on learning individual actions and separately relies on high level task composition [96, 23] or is often limited to previously seen objects [120, 118]. We believe that learning a single model for an action such as 'turning on' is impossible because human environments have so many variations. Although being able to explore and learn from its own practice is an important skill for robots, it is not feasible to pre-train robots for all possible variations in all possible variations of environments.

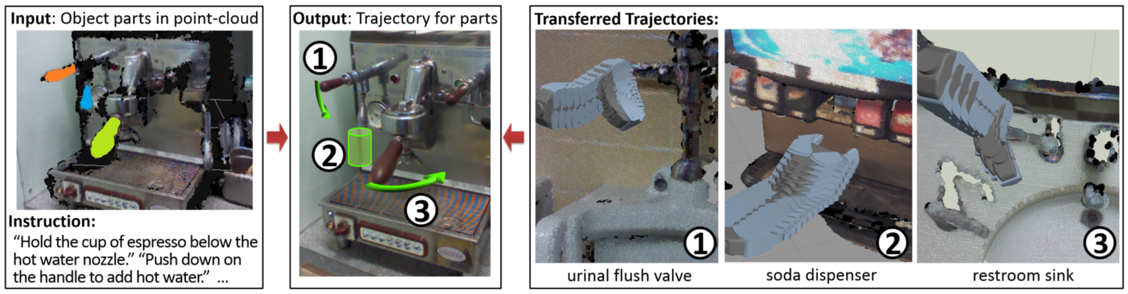

Input: Object parts in point-cloud
Instruction:
"Hold the cup of espresso below the hot water nozzle." "Push down on the handle to add hot water." ...
Output: Trajectory for parts
1
2
3
Transferred Trajectories:
1
2
3
urinal flush valve
soda dispenser
restroom sink

Figure 5.2: **Mapping object part and natural language instruction input to manipulation trajectory output.** Objects such as the espresso machine consist of distinct object parts, each of which requires a distinct manipulation trajectory for manipulation. For each part of the machine, we can re-use a manipulation trajectory that was used for some other object with similar parts. So, for an object part in a point-cloud (each object part colored on left), we can find a trajectory used to manipulate some other object (labeled on the right) that can be *transferred* (labeled in the center). With this approach, a robot can operate a new and previously unobserved object such as the 'espresso machine', by successfully transferring trajectories from other completely different but previously observed objects. Note that the input point-cloud is very noisy and incomplete (black represents missing points).

Unlike learning a model from demonstration, instance-based learning [3, 35] replicates one of the demonstrations. Similarly, we directly transfer one of the demonstrations, but focus on generalizing manipulation planning to completely new objects, enabling robots to manipulate objects they have never seen before.

Crowd-sourcing. Many approaches to teaching robots manipulation and other skills have relied on demonstrations by skilled experts [5, 2]. Among previous efforts to scale teaching to the crowd [20, 154, 58], Forbes et al. [35] employs a similar approach towards crowd-sourcing but collects multiple instances of similar table-top manipulation with same object. Others also build web-based platform for crowd-sourcing manipulation [158, 159]. However, these approaches either depend on the presence of an expert (due to required special software), or require a real robot at a remote location. Our Robobarista platform borrows some components of work from Alexander et al. [4], but works on any standard web browser with OpenGL support and incorporates real point-clouds of various scenes.

5.2 Our Approach

Our goal is to build an algorithm that allows a robot to infer a manipulation trajectory when it is introduced to a new object or appliance with its natural language instruction manual. To this end, we propose an idea that because many differently-shaped objects share similarly-operated object parts, the manipulation trajectory for a novel object can be transferred from a completely different object with similarly-operated parts.

For example, the motion required to operate the handle of the espresso machine in Figure 5.2 is almost identical to the motion required to flush the urinal with the handle. By identifying and transferring trajectories from prior experience with parts of other objects, robots can even manipulate objects they have never seen before.

We first formulate this problem as a structured prediction problem (Figure 5.2). Given a point-cloud for each part of an espresso machine and a natural language instruction such as 'Push down on the handle to add hot water' for each part of the object, our algorithm outputs a trajectory which performs the desired task using a pool of prior motion experience.

This is a challenging problem because the object is entirely new to the robot, and because it must jointly consider the point-cloud, natural language instruction, and each potential trajectory. Manually designing useful features from these three modalities is extremely challenging.

Popular supervised deep learning approaches (e.g. [76, 142, 47]) would directly try to output manipulation trajectory; however, in order for an algorithm to output in extremely large space (time-series sequences of 6-DoF end-effector poses), it requires a huge number of expert demonstrations on a large number of objects. Collecting such data is extremely time and resource extensive as it requires joint physical presence of a robot, an expert, and the object to be manipulated.

In order to address these challenges, we employ a *deep multimodal embedding* algorithm (Chapter 4) that learns a shared, semantically meaningful embedding space between these modalities, while dealing with a noise in crowd-sourced demonstration data. Then, we introduce our Robobarista crowd-sourcing platform, which allows us to easily scale the collection of manipulation demonstrations to non-experts on the web.

5.2.1 Problem Formulation

Our goal is to learn a function f that maps a given pair of point-cloud $p \in \mathcal{P}$ of an object part and a natural language instruction $l \in \mathcal{L}$ to a trajectory $\tau \in \mathcal{T}$ that can manipulate the object part as described by free-form natural language l:

$$f : \mathcal{P} \times \mathcal{L} \to \mathcal{T} \tag{5.1}$$

For instance, given the handle of the espresso machine in Figure 5.2 and an natural language instruction 'Push down on the handle to add hot water', the algorithm should output a manipulation trajectory that will correctly accomplish the task on the object part according to the instruction.

Point-cloud Representation. Each instance of a point-cloud $p \in \mathcal{P}$ is represented as a set of n points in three-dimensional Euclidean space where each point (x, y, z) is represented with its RGB color (r, g, b):

$$p = \{p^{(i)}\}_{i=1}^{n} = \{(x, y, z, r, g, b)^{(i)}\}_{i=1}^{n}$$

The size of this set varies for each instance. These points are often obtained by stitching together a sequence of sensor data from an RGBD sensor [56].

Trajectory Representation. Each trajectory $\tau \in \mathcal{T}$ is represented as a sequence of m *waypoints*, where each waypoint consists of gripper status g, translation (t_x, t_y, t_z), and rotation (r_x, r_y, r_z, r_w) with respect to the origin:

$$\tau = \{\tau^{(i)}\}_{i=1}^{m} = \{(g, t_x, t_y, t_z, r_x, r_y, r_z, r_w)^{(i)}\}_{i=1}^{m}$$

where $g \in \{\text{“open”}, \text{“closed”}, \text{“holding”}\}$. g depends on the type of the end-effector, which we have assumed to be a two-fingered parallel-plate gripper like that of PR2 or Baxter. The rotation is represented as quaternions (r_x, r_y, r_z, r_w) instead of the more compact Euler angles to prevent problems such as gimbal lock.

In order to acquire a smooth trajectory from a waypoint-based trajectory τ, we interpolate intermediate waypoints. Translation is linearly interpolated and the quaternion is interpolated using spherical linear interpolation (Slerp) [138].

Each modality is converted into a fixed-length vector for our machine learning algorithm as described in Sec 5.4.1.

5.2.2 Direct Manipulation Trajectory Transfer

Even if we have a trajectory to transfer, a conceptually transferable trajectory is not necessarily directly compatible if it is represented with respect to an inconsistent reference point.

To make a trajectory compatible with a new situation without modifying the trajectory, we need a representation method for trajectories, based on point-cloud information, that allows a *direct transfer of a trajectory without any modification*.

Challenges. Making a trajectory compatible when transferred to a different object or to a different instance of the same object without modification can be challenging depending on the representation of trajectories and the variations in the location of the object, given in point-clouds.

Many approaches which control high degree of freedom arms such as those of PR2 or Baxter use configuration-space trajectories, which store a time-parameterized series of joint angles [157]. While such approaches allow for direct control of joint angles during control, they require costly recomputation for even a small change in an object's position or orientation.

One approach that allows execution without modification is representing trajectories with respect to the object by aligning via point-cloud registration (e.g. [35]). However, a large object such as a stove might have many parts (e.g. knobs and handles) whose positions might vary between different stoves. Thus, object-aligned manipulation of these parts would not be robust to different stoves, and in general would impede transfer between different instances of the same object.

Lastly, it is even more challenging if two objects require similar trajectories, but have slightly different shapes. And this is made more difficult by limitations of the point-cloud data. As shown in left of Fig. 5.2, the point-cloud data, even when stitched from multiple angles, are very noisy compared to the RGB images.

Our Solution. Transferred trajectories become compatible across different objects when trajectories are represented *1)* in the task space rather than the configuration space, and *2)* relative to the object *part* in question (aligned based on its principal axis), rather than the object as a whole.

Trajectories are represented in the task space by recording only the position and orientation of the end-effector. By doing so, we can focus on the actual interaction between the robot and the environment rather than the movement of the arm. It is very rare that the arm configuration affects the completion of the task as long as there is no collision. With the trajectory represented as a sequence of gripper position and orientation, the robot can find its arm configuration that is collision free with the environment using inverse kinematics.

However, representing the trajectory in task space is not enough to make transfers compatible. The trajectory must also be represented in a common coordinate frame regardless of the object's orientation and shape.

Thus, we align the negative z-axis along gravity and align the x-axis along the principal axis of the object *part* using PCA [51]. With this representation, even when the object part's position and orientation changes, the trajectory does not need to change. The underlying assumption is that similarly operated object parts share similar shapes leading to a similar direction in their principal axes.

5.2.3 Inferring Manipulation Trajectory to Transfer

Once all mappings are learned using deep multimodal embedding algorithm, we solve the original problem from equation (5.1) by choosing, from a library of prior trajectories, the trajectory that gives the highest similarity (closest in distance) to the given point-cloud p and language l in our joint embedding space $\mathbb{R}^M$. As in previous work [176], similarity is defined as $sim(a, b) = a \cdot b$, and the trajectory that maximizes the magnitude of similarity is selected:

$$\operatorname*{argmax}_{\tau \in \mathcal{T}} sim(\Phi_{\mathcal{P},\mathcal{L}}(p, l), \Phi_{\mathcal{T}}(\tau))$$

The previous approach to this problem [148] required projecting the combination of the current point-cloud and natural language instruction with *every* trajectory in the training set through the network during inference. Here, we pre-compute the representations of all training trajectories in h^3, and need only project the new point-cloud/language pair to h^3 and find its nearest-neighbor trajectory in this embedding space. As shown in Section 5.5.3, this significantly improves both the runtime and accuracy of our approach and makes it much more scalable to larger training datasets like those collected with crowdsourcing platforms.

5.2.4 Label Noise

When our data contains a significant number of noisy trajectories τ, e.g. due to crowd-sourcing (Sec. 5.3), not all trajectories should be trusted as equally appropriate, as will be shown in Sec. 5.5.

For every pair of inputs (p_i, l_i), we have $\mathcal{T}_i = \{\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,n_i}\}$, a set of trajectories submitted by the crowd for (p_i, l_i). First, the best candidate label $\tau_i^* \in \mathcal{T}_i$ for (p_i, l_i) is selected as the one with the smallest average trajectory distance to the others:

$$\tau_i^* = \underset{\tau \in \mathcal{T}_i}{\operatorname{argmin}} \frac{1}{n_i} \sum_{j=1}^{n_i} \Delta(\tau, \tau_{i,j})$$

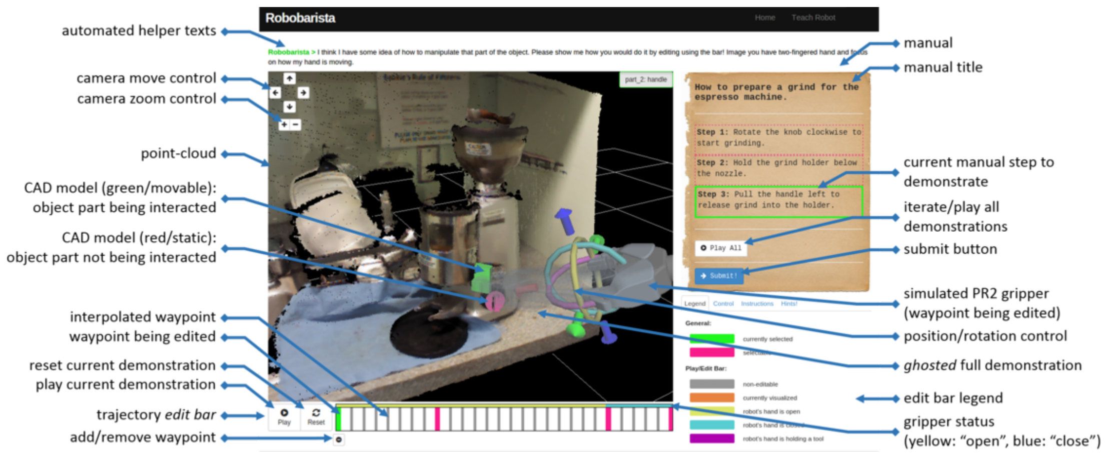
We assume that at least half of the crowd tried to give a reasonable demonstration. Thus a demonstration with the smallest average distance to all other demonstrations must be a good demonstration. We use the DTW-MT distance function (described later in Appendix A.1) for our loss function $\Delta(\tau, \bar{\tau})$, but it could be replaced by any function that computes the loss of predicting $\bar{\tau}$ when τ is the correct demonstration.

Using the optimal demonstration and a loss function $\Delta(\tau, \bar{\tau})$ for comparing demonstrations, we find a set of trajectories $\mathcal{T}_{i,S}$ that are relevant (similar) to this task and a set of trajectories $\mathcal{T}_{i,D}$ that are irrelevant (dissimilar.) We can use thresholds (t_S, t_D) determined by the expert to generate two sets from the pool of trajectories:

$$\mathcal{T}_{i,S} = \{\tau \in \mathcal{T} | \Delta(\tau_i^*, \tau) < t_S\}$$

$$\mathcal{T}_{i,D} = \{\tau \in \mathcal{T} | \Delta(\tau_i^*, \tau) > t_D\}$$

This method allows our model to be robust against noisy labels and also serves as a method of data augmentation by also considering demonstrations given for other tasks in both sets of $\mathcal{T}_{i,S}$ and $\mathcal{T}_{i,D}$.


automated helper texts
camera move control
camera zoom control
point-cloud
CAD model (green/movable): object part being interacted
CAD model (red/static): object part not being interacted
interpolated waypoint
waypoint being edited
reset current demonstration
play current demonstration
trajectory *edit bar*
add/remove waypoint
manual
manual title
current manual step to demonstrate
iterate/play all demonstrations
submit button
simulated PR2 gripper (waypoint being edited)
position/rotation control
ghosted full demonstration
edit bar legend
gripper status (yellow: "open", blue: "close")
Robobarista
Home
Teach Robot
part_2: handle
Play All
Submit!
Play
Reset
Legend
Control
Instructions
Hints!

Robobarista > I think I have some idea of how to manipulate that part of the object. Please show me how you would do it by editing using the bar! Image you have two-fingered hand and focus on how my hand is moving.

How to prepare a grind for the espresso machine.

Step 1: Rotate the knob clockwise to start grinding.

Step 2: Hold the grind holder below the nozzle.

Step 3: Pull the handle left to release grind into the holder.

General:

🟩	currently selected
🟥	selectable

Play/Edit Bar:

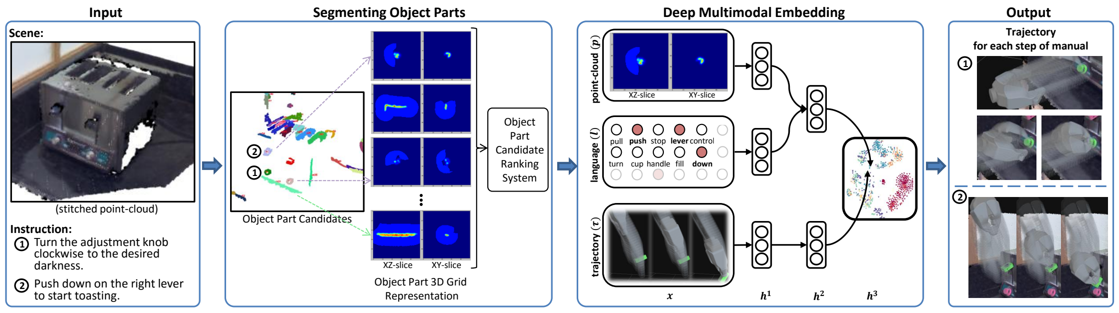
⬜	non-editable
🟧	currently visualized
🟨	robot's hand is open
🟦	robot's hand is closed
🟪	robot's hand is holding a tool

Figure 5.3: **Screen-shot of Robobarista,** the crowd-sourcing platform running on Chrome browser. We have built Robobarista platform for collecting a large number of crowd demonstrations for teaching the robot.

5.3 Robobarista: Crowd-sourcing Platform

To train our multimodal embedding neural network, it is important to have access to a large amount of training data, which is especially challenging for robot manipulation tasks with lots of different objects. In order to collect a large number of manipulation demonstrations without risking, we built a crowd-sourcing web platform that we call Robobarista (see Fig. 5.3). It provides a virtual environment where non-expert users can teach robots via a web browser, without expert guidance or physical presence with a robot and a target object.

The system simulates a situation where the user encounters a previously unseen target object and a natural language instruction manual for its manipulation. Within the web browser, users are shown a point-cloud in the 3-D viewer on the left and a *manual* on the right. A manual may involve several instructions, such as "Push down and pull the handle to open the door". The user's goal is



Input
Scene:
(stitched point-cloud)
Instruction:
① Turn the adjustment knob clockwise to the desired darkness.
② Push down on the right lever to start toasting.
Segmenting Object Parts
Object Part Candidates
XZ-slice
XY-slice
Object Part 3D Grid Representation
Object Part Candidate Ranking System
Deep Multimodal Embedding
point-cloud (p)
language (l)
trajectory (τ)
pull push stop lever control
turn cup handle fill down
x
h¹
h²
h³
Output
Trajectory for each step of manual

Figure 5.4: **System Overview:** Given a point-cloud and a language instruction, our goal is to output a trajectory that would manipulate the object according to the instruction. The given point-cloud scene is segmented into many parts and ranked for each step of the instruction manual. By embedding point-cloud, language, and trajectory modalities into a joint embedding space, our algorithm selects the best trajectory to transfer to the new object.

to demonstrate how to manipulate the object in the scene for each instruction.

The user starts by selecting one of the instructions on the right to demonstrate (Fig. 5.3). Once selected, the target object part is highlighted and the trajectory *edit bar* appears below the 3-D viewer. Using the *edit bar*, which works like a video editor, the user can playback and edit the demonstration. The trajectory representation, as a set of waypoints (Sec. 5.2.1), is directly shown on the *edit bar*. The bar shows not only the set of waypoints (red/green) but also the interpolated waypoints (gray). The user can click the 'play' button or hover the cursor over the edit bar to examine the current demonstration. The blurred trail of the current trajectory (*ghosted*) demonstration is also shown in the 3-D viewer to show its full expected path.

Generating a full trajectory from scratch is difficult for non-experts. Thus, similar to Forbes et al. [35], we provide a trajectory that the system has already

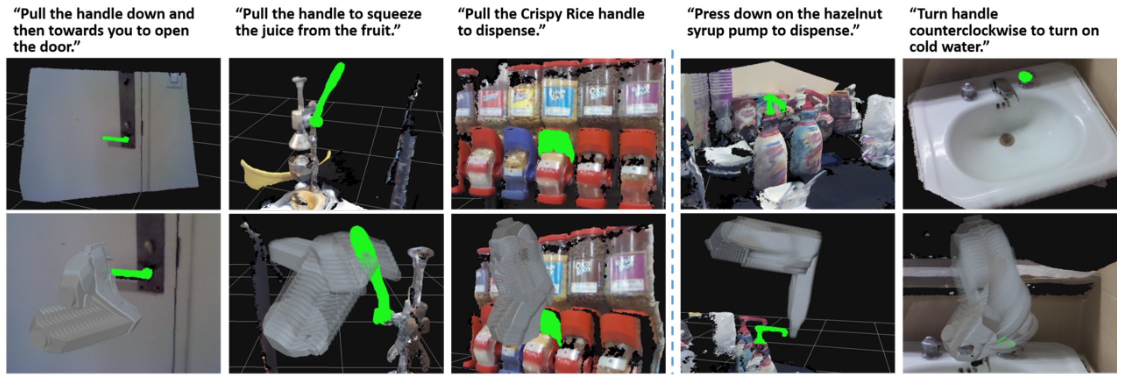

"Pull the handle down and then towards you to open the door."
"Pull the handle to squeeze the juice from the fruit."
"Pull the Crispy Rice handle to dispense."
"Press down on the hazelnut syrup pump to dispense."
"Turn handle counterclockwise to turn on cold water."

Figure 5.5: **Examples from our dataset,** each of which consists of a natural language instruction (top), an object part in point-cloud representation (highlighted), and a manipulation trajectory (below) collected via Robobarista. Objects range from kitchen appliances such as stove and rice cooker to urinals and sinks in restrooms. As our trajectories are collected from non-experts, they vary in quality from being likely to complete the manipulation task successfully (left of dashed line) to being unlikely to do so successfully (right of dashed line).

seen for another object as the initial starting trajectory to edit.[1]

In order to simulate a realistic experience of manipulation, instead of simply showing a static point-cloud, we have overlaid CAD models for parts such as 'handle' so that functional parts actually move as the user tries to manipulate the object.

A demonstration is edited by: 1) modifying the position/orientation of a waypoint, 2) adding/removing a waypoint, and 3) opening/closing the gripper. Once a waypoint is selected, the PR2 gripper is shown with six directional arrows and three rings, used to modify the gripper's position and orientation, respectively. To add extra waypoints, the user can hover the cursor over an interpolated (gray) waypoint on the *edit bar* and click the plus(+) button. To re-

[1]We have made sure that it does not initialize with trajectories from other folds to keep *5-fold cross-validation* in experiment section valid.

move an existing waypoint, the user can hover over it on the *edit bar* and click minus(-) to remove. As modification occurs, the edit bar and ghosted demonstration are updated with a new interpolation. Finally, for editing the status (open/close) of the gripper, the user can simply click on the gripper.

For broader accessibility, all functionality of Robobarista, including 3-D viewer, is built using Javascript and WebGL. We have made the platform available online (`http://robobarista.cs.cornell.edu`)

5.4 System Details

Our goal is to use our learned embedding space to allow the robot to infer a manipulation trajectory when it is introduced to a new appliance with its natural language instruction manual.

As shown in Fig. 5.4, for example, given a point-cloud of a scene with a toaster and an instruction such as 'Push down on the right lever to start toasting,' it should identify part of the scene the instruction is referring to and output a trajectory, representative of how the two-fingered end-effector should move, including how to approach, grasp, and push down on the lever.

First, in order to correctly identify a part p out of a scene s that an instruction asks to manipulate, a point-cloud of a scene s is segmented into many small potential candidates. All segments are ranked for each step of the manual instruction. Details of finding and ranking of object part candidate algorithm are explained in Appendix A.2.

Multiple variations of correct segmentations and lots of incorrect segmentation provided by segmentation algorithm make learning of our deep embedding representation even more robust as it is used as extra positive and negative pairs as empirically shown in Sec. 5.5.3.

Then, from a library of trajectories with prior experience, the trajectory that gives the highest similarity to the selected point-cloud p and language l in our embedding space $\mathbb{R}^M$:

$$\operatorname*{argmax}_{\tau \in \mathcal{T}} \; sim(\Phi_{\mathcal{P},\mathcal{L}}(p, l), \Phi_{\mathcal{T}}(\tau))$$

As in [176], similarity is defined as $sim(a, b) = a \cdot b$.

Our approach allows us to pre-embed all candidate trajectories into a shared embedding space. The correct trajectory can then be identified by embedding only a new point-cloud/language pair. As shown in Sec. 5.5.3, this significantly improves both the inference run-time and accuracy as it makes it more scalable to a larger number of trajectories.

5.4.1 Data Representation

All three data modalities – point-cloud, language, and trajectory data (p, l, τ) are variable-length and must be transformed into a fixed-length representation.

Each point-cloud segment is converted into a real-valued 3D occupancy grid where each cell's value is proportional to how many points fall into the cube it spans. We use a $100 \times 100 \times 100$ grid of cubic cells with sides of $0.25cm$. Unlike our previous work [148], each cell count is also distributed to the neighboring cells with an exponential distribution. This smooths out missing points and increases the amount of information represented. The grid then is normalized to be between $0 \sim 1$ by dividing by the maximal count.

While our approach focuses on the shape of the part in question, the shape of the nearby scene can also have a significant effect on how the part is manipulated. To account for this, we assign a value of 0.2 to any cell which contains only points which belong to the scene but not the specific part in question, but are within some distance from the nearest point for the given part. To fill hollow parts behind the background, such as tables and walls, we ray-trace between the starting location of the sensor and cells filled by background points and fill these similarly.

While our segment ranking algorithm uses the full-sized grid for each segment, our main embedding algorithm uses two compact grids generated by taking average of cells: $10 \times 10 \times 10$ grid with cells with sides of $2.5cm$ and of $1cm$.

Each language instruction is represented as a fixed-size bag-of-words representation with stop words removed. Finally, for each trajectory $\tau \in \mathcal{T}$, we first compute its smooth interpolated trajectory $\tau_s \in \mathcal{T}_s$ (Sec. 5.2.1), and then normalize all trajectories $\mathcal{T}_s$ to the same length while preserving the sequence of gripper states such as 'opening', 'closing', and 'holding'.

5.4.2 Robotic Platform

We tested our algorithms on a PR2, an omni-directional base, and many sensors including a Microsoft Kinect, stereo cameras, and a tilting laser scanner (Fig. 5.13). For our these experiments, a point-cloud is acquired from the head mounted Kinect sensor and each motion is executed on the specified arm using a Cartesian end-effector stiffness controller [14] in ROS [123]. Our embedding algorithm are written with Theano [7], and most of our computations are done on a remote computer utilizing a GPU for our embedding model.

5.4.3 Model Parameters

Through validation, we found an optimal embedding space size M of 25 and intermediate-layer sizes $N_{1,p}$, $N_{1,l}$, $N_{1,\tau}$, $N_{2,pl}$, and $N_{2,\tau}$ of 250, 150, 100, 125, and 100 with the loss scaled by $\alpha = 0.2$. These relatively small layer sizes also had the advantage of fast inference, as shown in Sec. 5.5.3.

5.5 Experiments

We perform a series of experiments on our approach to manipulating novel objects focusing on following set of questions:

1. Does our approach of directly transferring manipulation trajectories apply to a large number of novel appliances and objects?
2. Is it possible to crowd-source manipulation trajectories from non-experts and use it to learn to manipulate novel appliances and objects?
3. How does our deep multimodal embedding algorithm compare to other machine learning approaches in the task of transferring trajectories? Does it learn a semantically meaningful space for the three distinct modalities of point-cloud, language and manipulation trajectories?

Table 5.1: **Results on our dataset** with *5-fold cross-validation*. Rows list models we tested including our model and baselines. Columns show different metrics used to evaluate the models.

	per manual	per instruction	
Models	**DTW-MT**	**DTW-MT**	**Accuracy (%)**
Chance	28.0 (±0.8)	27.8 (±0.6)	11.2 (±1.0)
Object Part Classifier	-	22.9 (±2.2)	23.3 (±5.1)
Structured SVM	21.0 (±1.6)	21.4 (±1.6)	26.9 (±2.6)
Latent SSVM + Kinematic [145]	17.4 (±0.9)	17.5 (±1.6)	40.8 (±2.5)
Task similarity + Random	14.4 (±1.5)	13.5 (±1.4)	49.4 (±3.9)
Task Similarity + Weights [35]	13.3 (±1.2)	12.5 (±1.2)	53.7 (±5.8)
Deep Network with Noise-handling without Embedding	13.7 (±1.6)	13.3 (±1.6)	51.9 (±7.9)
Deep Multimodal Network without Embedding	14.0 (±2.3)	13.7 (±2.1)	49.7 (±10.0)
Deep Multimodal Network with Noise-handling without Embedding [148]	13.0 (±1.3)	12.2 (±1.1)	60.0 (±5.1)
LMNN-like Cost Function [174]	15.4 (±1.8)	14.7 (±1.6)	55.5 (±5.3)
Our Model without Any Pretraining	13.2 (±1.4)	12.4 (±1.0)	54.2 (±6.0)
Our Model with SDA	11.5 (±0.6)	11.1 (±0.6)	62.6 (±5.8)
Our Model without Noise Handling	12.6 (±1.3)	12.1 (±1.1)	53.8 (±8.0)
Our Model without Multiple Segmentations	11.0 (±0.8)	10.5 (±0.7)	65.1 (±4.9)
Our Model with Experts	12.3 (±0.5)	11.8 (±0.9)	56.5 (±4.5)
Our Model - Deep Multimodal Embedding	**10.3** (±0.8)	**9.9** (±0.5)	**68.4** (±5.2)

4. Does our approach allow real robots to successfully complete the manipulation task when encountered with novel objects?

To answer question 1, we first collect a large dataset consisting of a wide variety of objects for evaluation. We utilize our Robobarista crowd-sourcing platform (question 2) to collect manipulation trajectories for these objects. On this large dataset, we design different machine learning approaches to compare against our deep multimodal embedding algorithm and evaluate the learned embedding space of different modalities (question 3). We then test our full end-to-end system on PR2 robot manipulating different objects (question 4). Finally, to fully explore our idea of transferring manipulation trajectories (question 1) on a physical robot (question 4), we ask our robot to prepare a cup of latte with a grinder and an espresso machine.

5.5.1 Robobarista Dataset

In order to test our model, we have collected a dataset of 116 point-clouds of objects with 249 object parts (examples shown in Figure 5.5). Objects range from kitchen appliances such as stoves and rice cookers to bathroom hardware such as sinks and toilets. Figure 5.6 shows a sample of 70 such objects. There are also a total of 250 natural language instructions (in 155 manuals).[2] Using the crowd-sourcing platform Robobarista, we collected 1225 trajectories for these objects from 71 non-expert users on the Amazon Mechanical Turk. After a user is shown a 20-second instructional video, the user first completes a 2-minute tutorial task. At each session, the user was asked to complete 10 assignments where each consists of an object and a manual to be followed.

For each object, we took raw RGB-D images with the Microsoft Kinect sensor and stitched them using Kinect Fusion [56] to form a denser point-cloud in order to incorporate different viewpoints of objects. Objects range from kitchen appliances such as 'stove', 'toaster', and 'rice cooker' to 'urinal', 'soap dispenser', and 'sink' in restrooms. The dataset is made available at `http://robobarista.cs.cornell.edu`

[2]Although not necessary for training our model, we also collected trajectories from the expert for evaluation purposes.

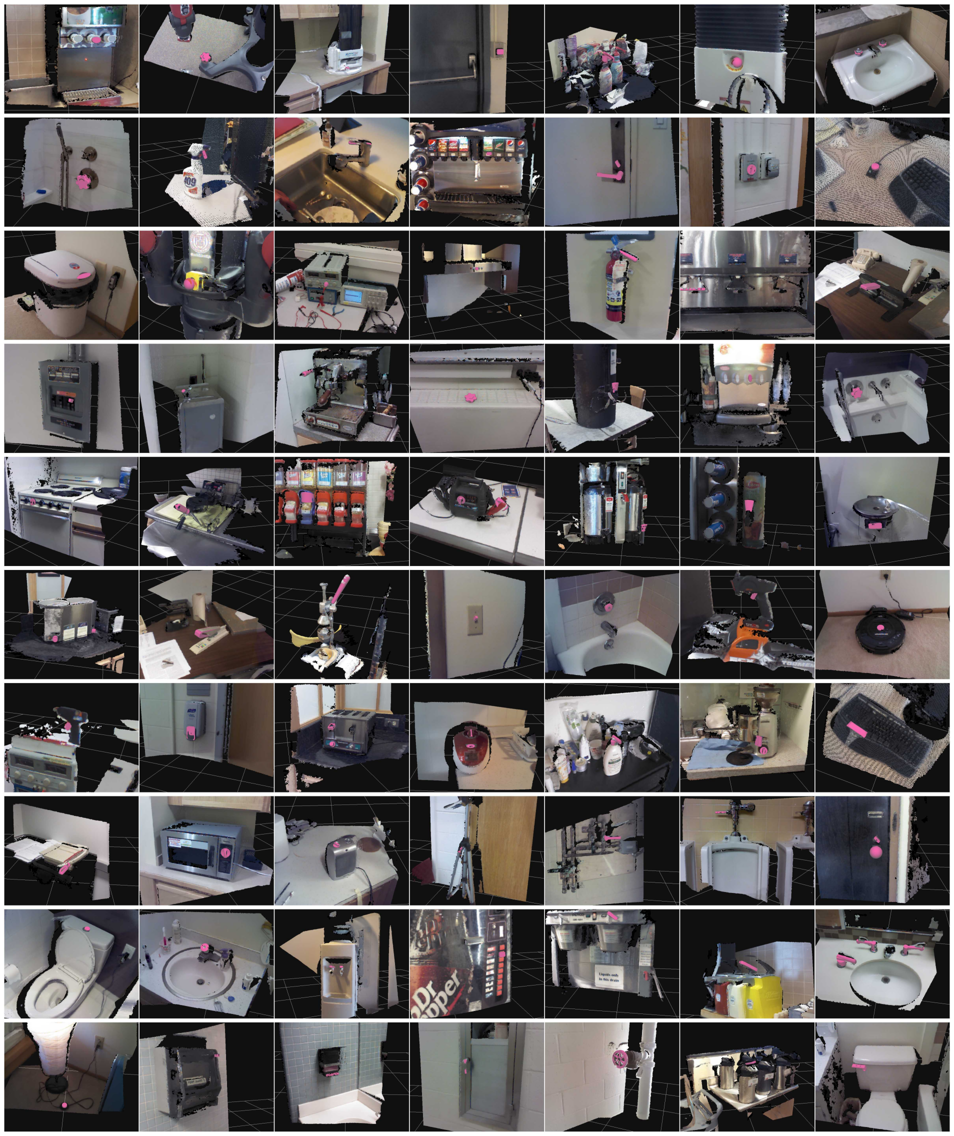


Figure 5.6: **Examples of objects from our dataset.** Each image shows the point cloud representation of an object. We overlaid some of its parts by CAD models for online Robobarista crowd-sourcing platform. Note that the actual underlying point-cloud of object parts contains much more noise and is not clearly segmented, and none of the models have access to overlaid model for inferring manipulation trajectory.

5.5.2 Baselines

To compare different aspects of our model, we designed a large number of models. *Object part classifier* classifies each object part into a part label and transfer

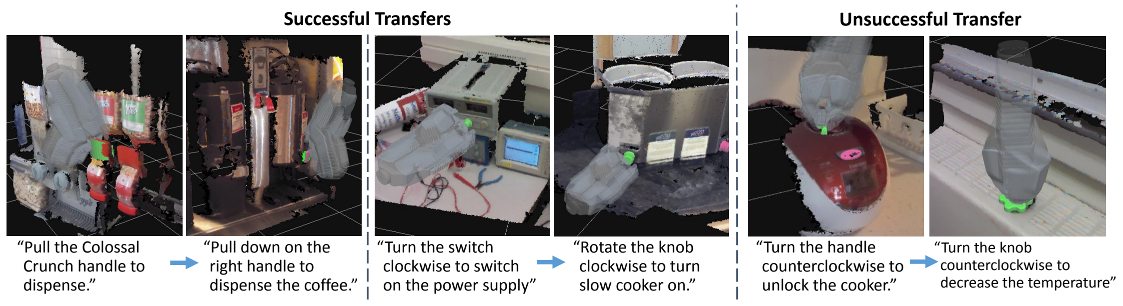

Successful Transfers
Unsuccessful Transfer
“Pull the Colossal Crunch handle to dispense.”
“Pull down on the right handle to dispense the coffee.”
“Turn the switch clockwise to switch on the power supply”
“Rotate the knob clockwise to turn slow cooker on.”
“Turn the handle counterclockwise to unlock the cooker.”
“Turn the knob counterclockwise to decrease the temperature”

Figure 5.7: **Examples of successful and unsuccessful transfers** of manipulation trajectory from left to right using our model. In first two examples, though the robot has never seen the ‘coffee dispenser’ and ‘slow cooker’ before, the robot has correctly identified that the trajectories of ‘cereal dispenser’ and ‘DC power supply’, respectively, can be used to manipulate them.

accordingly. *Structured SVM*-based models and *task similarity*-based models rely on expert-designed state-of-the-art techniques to reason about three modalities of point-cloud, language and trajectory. We also designed *deep neural network*-based models [148] that outputs a score for the given pairs of point-cloud, language and trajectory rather than finding a semantically meaningful embedding. Variations of our *deep multimodal embedding* are also tested with different pre-training algorithm, without noise-handling, with LMNN-like [174] cost function, and so forth. Please refer to Appendix A.3 for the detailed descriptions of each baseline model.

5.5.3 Evaluations on Robobarista Dataset

We evaluated all models on our dataset using *5-fold cross-validation* and the results are in Table 5.1. All models which required hyper-parameter tuning used 10% of the training data as the validation set.

Rows list the models we tested including our model and baselines. Each column shows one of three evaluation metrics. The first two use dynamic time warping for manipulation trajectory (DTW-MT) from Appendix A.1. The first column shows averaged DTW-MT for each instruction manual consisting of one or more language instructions. The second column shows averaged DTW-MT for every test pair (p, l).

As DTW-MT values are not intuitive, we also include a measure of "accuracy," which shows the percentage of transferred trajectories with DTW-MT value less than 10. Through expert surveys, we found that when DTW-MT of manipulation trajectory is less than 10, the robot came up with a reasonable trajectory and will very likely be able to accomplish the given task. Additionally, Fig. 5.9 shows accuracies obtained by varying the threshold on the DTW-MT measure.

Can manipulation trajectories be transferred from completely different objects? Our full model gave 65.1% accuracy (Table 5.1), outperforming every other baseline approach tested.

Fig. 5.7 shows two examples of successful transfers and one unsuccessful transfer by our model. In the first example, the trajectory for pulling down on a cereal dispenser is transferred to a coffee dispenser. Because our approach to trajectory representation is based on the principal axis (Sec. 5.2.2), even though the cereal and coffee dispenser handles are located and oriented differently, the transfer is a success. The second example shows a successful transfer from a DC power supply to a slow cooker, which have "knobs" of similar shape. The transfer was successful despite the difference in instructions ("Turn the switch.." and "Rotate the knob..") and object type. This highlights the advantages of our end-to-end approach over relying on semantic classes for parts and actions.

The last example in Fig. 5.7 shows a potentially unsuccessful transfer. Despite the similarity in two instructions and similarity in required counterclockwise motions, the transferred motion might not be successful. While the knob on radiator must be grasped in the middle, the rice cooker has a handle that extends sideways, requiring it to be grasped off-center. For clarity of visualization in figures, we have overlaid CAD models over some noisy point-clouds. Many of the object parts were too small and/or too glossy for the Kinect sensor. We believe that a better 3-D sensor would allow for more accurate transfers. On the other hand, it is interesting to note that the transfer in opposite direction from the radiator knob to the rice cooker handle may have yielded a correct manipulation.

Can we crowd-source the teaching of manipulation trajectories? When we trained our full model with expert demonstrations, which were collected for evaluation purposes, it performed at 56.5% compared to 65.1% by our model trained with crowd-sourced data. Even though non-expert demonstrations can carry significant noise, as shown in last two examples of Fig. 5.5, our noise-handling approach allowed our model to take advantage of the larger, less accurate crowd-sourced dataset. Note that all of our crowd users are true non-expert users from Amazon Mechanical Turk.

Is intermediate object part labeling necessary? A multiclass SVM trained on object part labels was able to obtain over 70% recognition accuracy in classifying five major classes of object parts ('button', 'knob', 'handle', 'nozzle', 'lever'.) However, the *Object Part Classifier* baseline, based on this classification, performed at only 23.3% accuracy for actual trajectory transfer, outperforming

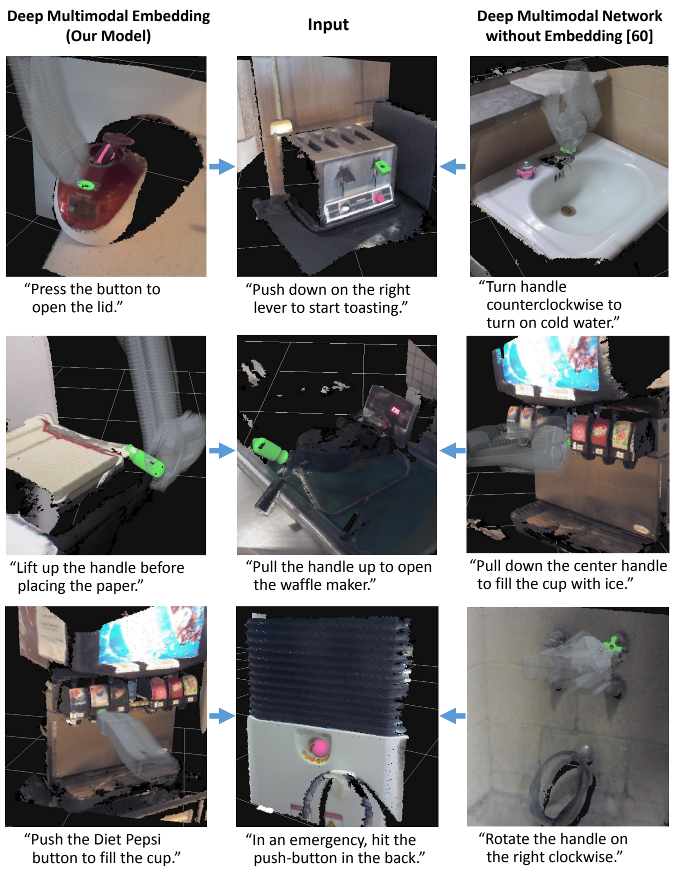

Deep Multimodal Embedding (Our Model)
Input
Deep Multimodal Network without Embedding [60]
"Press the button to open the lid."
"Push down on the right lever to start toasting."
"Turn handle counterclockwise to turn on cold water."
"Lift up the handle before placing the paper."
"Pull the handle up to open the waffle maker."
"Pull down the center handle to fill the cup with ice."
"Push the Diet Pepsi button to fill the cup."
"In an emergency, hit the push-button in the back."
"Rotate the handle on the right clockwise."

Figure 5.8: **Comparisons of transfers** between our model and the baseline (deep multimodal network without embedding [148]). In these three examples, our model successfully finds correct manipulation trajectory from these objects while the other one does not. Given the lever of the toaster, our algorithm finds similarly slanted part from the rice cooker while the other model finds completely irrelevant trajectory. For the opening action of waffle maker, trajectory for paper cutter is correctly identified while the other model transfers from a handle that has incompatible motion.

chance by merely 12.1%, and significantly underperforming our model's result of 65.1%. This shows that object part labels alone are not sufficient to enable manipulation motion transfer, while our model, which makes use of richer information, does a much better job.

Can features be hand-coded? What does our learned deep embedding space

represent? Even though we carefully designed state-of-the-art task-specific features for the SSVM and LSSVM models, these models only gave at most 40.8% accuracy. The *task similarity* method gave a better result of 53.7%, but it requires access to *all* of the raw training data (point-clouds, language, and trajectories) at test time, which leads to heavy computation at test time and requires a large amount of storage as the size of training data increases. Our approach, by contrast, requires only the trajectory data, and a low-dimensional representation of the point-cloud and language data, which is much less expensive to store than the raw data.

This shows that it is extremely difficult to find a good set of features which properly combines these three modalities. Our multimodal embedding model does not require hand-designing such features, instead learning a joint embedding space as shown by our visualization of the top layer h^3 in Figure 5.11. This visualization is created by projecting all training data (point-cloud/language pairs and trajectories) of one of the cross-validation folds to h^3, then embedding them to 2-dimensional space using t-SNE [166]. Although previous work [148] was able to visualize several nodes in the top layer, most were difficult to interpret. With our model, we can embed all our data and visualize all the layers (see Figs. 5.11 and 5.14).

One interesting result is that our system was able to naturally learn that “nozzle” and “spout” are effectively synonyms for purposes of manipulation. It clustered these together in the lower-right of Fig. 5.11 based solely on the fact that both are associated with similar point-cloud shapes and manipulation trajectories. At the same time, it also identified one exception, a small cluster of “nozzles” in the center of Fig. 5.11 which require different manipulation mo-

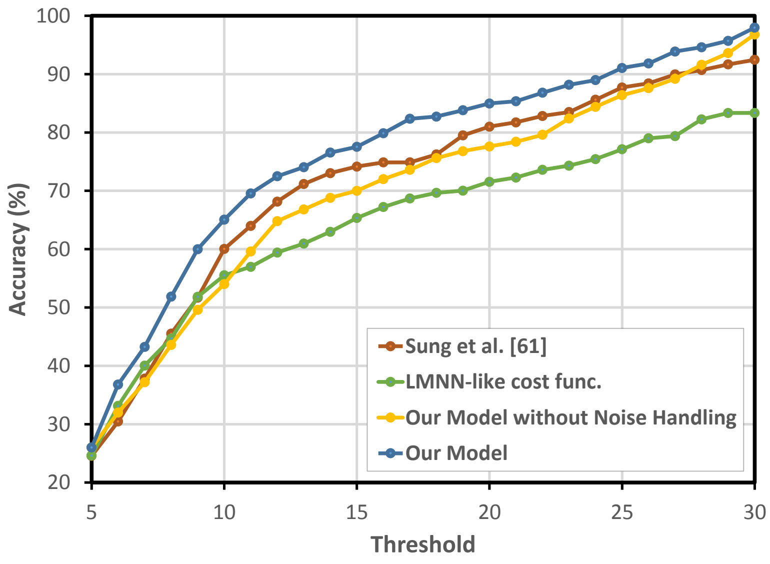


Figure 5.9: **Thresholding Accuracy:** Accuracy-threshold graph showing results of varying thresholds on DTW-MT scores. Our algorithm consistently outperforms the previous approach [148] and an LMNN-like cost function [174].

tions.

In addition to the aforementioned cluster in the bottom-right of Fig. 5.11, we see several other logical clusters. Importantly, we can see that our embedding maps vertical and horizontal rotation operations to very different regions of the space – roughly 12 o'clock and 8 o'clock in Fig. 5.11, respectively. Even though these have nearly identical language instructions, our algorithm learns to map them differently based on their point-clouds, mapping nearby the appropriate manipulation trajectories.

Should cost function be loss-augmented? When we changed the cost function for pre-training h^2 and fine-tuning h^3 to use a constant margin of 1 between relevant $\mathcal{T}_{i,S}$ and irrelevant $\mathcal{T}_{i,D}$ demonstrations [174], performance drops to 55.5%. This loss-augmentation is also visible in our embedding space. Notice the purple cluster around the 6 o'clock region of Fig. 5.11, and the lower part of the

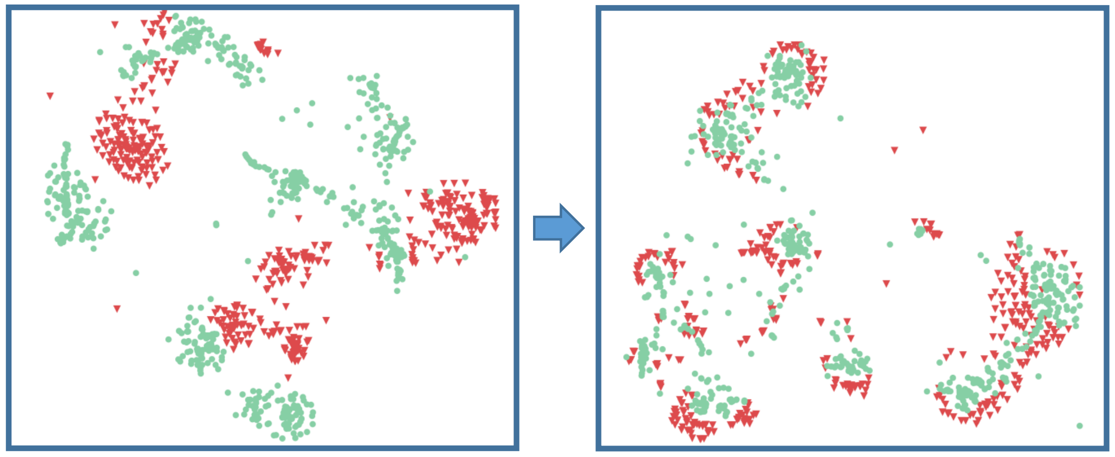


Figure 5.10: **Effect of Constraints:** The use of *inter-modal* and *intra-modal constraints* on deep embedding space h^3. The left one shows embedding space trained only with inter-modal constraints and the right one shows that of utilizing both constraints (our full model). The red points represents embeddings of point-cloud/language pairs and the green points represents embedding of trajectories. While both embeddings shows similar accuracy on our dataset, our full model provides much visible clusters, allowing us to more easily visualize and analyze what model has learned.

cluster in the 5 o'clock region. The purple cluster represents tasks and demonstrations related to pushing a bar (often found on soda fountains), and the lower part of the red cluster represents the task of holding a cup below the nozzle. Although the motion required for one task would not be replaceable by the other, the motions and shapes are very similar, especially compared to most other motions e.g. turning a horizontal knob.

Is pre-embedding important? As seen in Table 5.1, without any pre-training our model gives an accuracy of only 54.2%. Pre-training the lower layers with the conventional stacked de-noising auto-encoder (SDA) algorithm [170, 188] increases performance to 62.6%, still significantly underperforming our pre-training algorithm, which gives 65.1%. This shows that our metric embedding

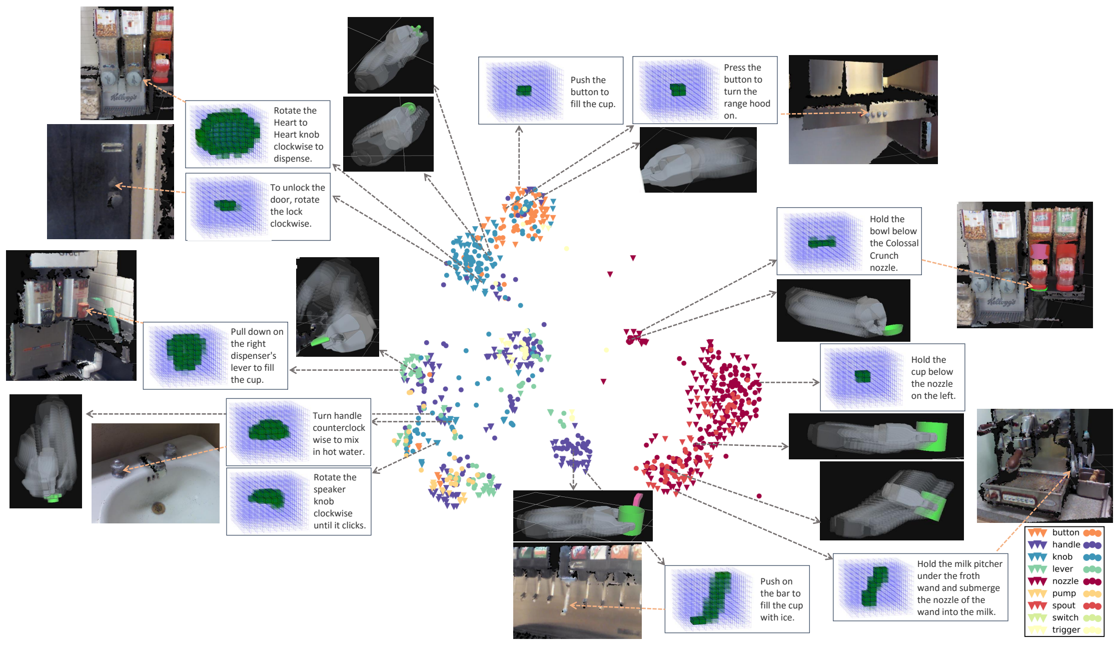


Figure 5.11: **Learned Deep Point-cloud/Language/Trajectory Embedding Space:** Joint embedding space h^3 after the network is fully fine-tuned, visualized in 2d using t-SNE [166] . Inverted triangles represent projected point-cloud/language pairs, circles represent projected trajectories. The occupancy grid representation of object part point-clouds is shown in green in blue grids. Among the two occupancy grids (Sec. 5.4.1), we selected the one that is more visually parsable for each object. The legend at the bottom right shows classifications of object parts by an expert, collected for the purpose of building a baseline. As shown by result of this baseline (object part classifier in Table 5.1), these labels do not necessarily correlate well with the actual manipulation motion. Thus, full separation according to the labels defined in the legend is not optimal and will not occur in this figure or Fig. 5.14. These figures are best viewed in color.

pre-training approach provides a better initialization for an embedding space than SDA.

Fig. 5.14 shows the joint point-cloud/language embedding $h^{2,pl}$ after the network is initialized using our pre-training algorithm and then fine-tuned using our cost function for h^3. While this space is not as clearly clustered as h^3 shown in Fig. 5.11, we note that point-clouds tend to appear in the more general center of the space, while natural language instructions appear around the more-specific edges. This makes sense because one point-cloud might afford many possible actions, while language instructions are much more specific.

Can automatically segmented object parts be manipulated? From Table 5.2, we see that our segmentation approach was able to find a good segmentation for the object parts in question in 50 of 60 robotic trials (Sec. 5.5.4), or 83.3% of the time. Most failures occurred for the beverage dispenser, which had a small lever that was difficult to segment.

When our full DME model utilizes two variations of same part and uses all candidates as a training data for the auto-encoder, our model performs at 68.4% compared to 65.1% which only used expert segmentations.

Does embedding improve efficiency? The previous model [148] had $749,638$ parameters to be learned, while our model has only $418,975$ (and still gives better performance.)

The previous model had to compute joint point-cloud/language/trajectory features for all combinations of the current point-cloud/language pair with *each* candidate trajectory (i.e. all trajectories in the training set) to infer an optimal trajectory. This is inefficient and does not scale well with the number of training datapoints. However, our model pre-computes the projection of all trajectories into h^3. Inference in our model then requires only projecting the new point-cloud/language combination to h^3 once and finding the trajectory with maximal similarity in this embedding.

In practice, this results in a significant improvement in efficiency, decreasing

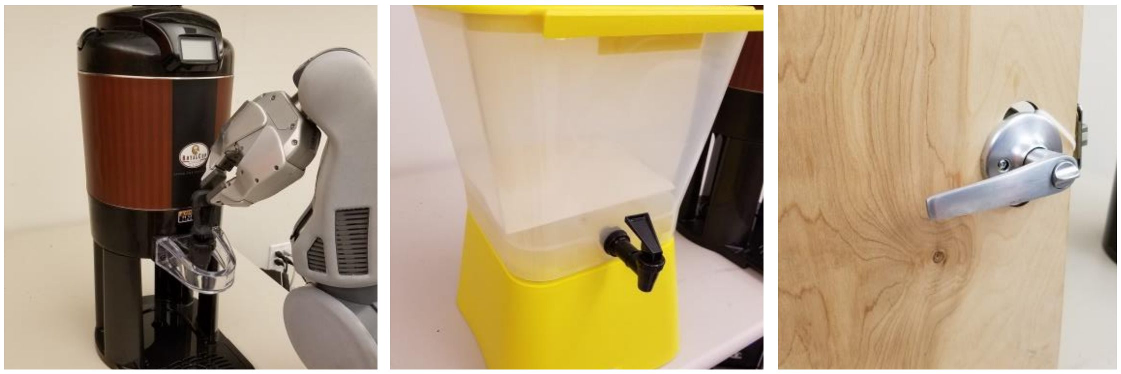


Figure 5.12: **Robotic Experiments:** We test our algorithm on a PR2 robot with three different novel objects – coffee dispenser handle, beverage dispenser lever, and door handle.

the average time to infer a trajectory from 2.3206ms to 0.0135ms, a speed-up of about 171x. Time was measured on the same hardware, with a GPU (GeForce GTX Titan X), using the Theano library [7]. We measured inference times 10000 times for first test fold, which has a pool of 962 trajectories. Time to preprocess the data and time to load into GPU memory was not included in this measurement. We note that the only part of our algorithm's runtime which scales up with the amount of training data is the nearest-neighbor computation, for which there exist many efficient algorithms [105]. Thus, our algorithm could be scaled to much larger datasets, allowing it to handle a wider variety of tasks, environments, and objects.

5.5.4 End-to-end Robot Experiments

In this section, we evaluate our complete end-to-end system on a physical robot. We perform a total of 100 fully autonomous experiments on a PR2 robot with three objects of coffee dispenser handle, beverage lever, and door handle as

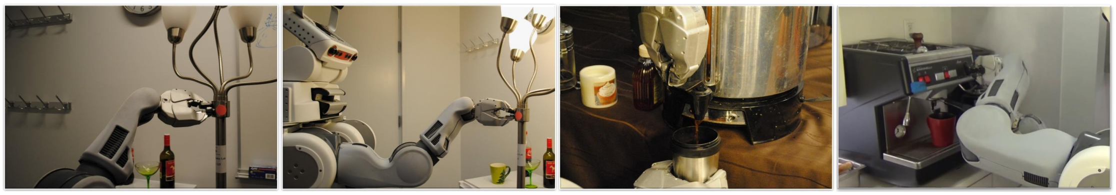


Figure 5.13: **Examples of transferred trajectories** being executed on PR2. On the left, PR2 is able to rotate the 'knob' to turn the lamp on. In the third snapshot, using two transferred trajectories, PR2 is able to hold the cup below the 'nozzle' and press the 'lever' of 'coffee dispenser'. In the last example, PR2 is frothing milk by pulling down on the lever, and is able to prepare a cup of latte with many transferred trajectories.

shown in Fig. 5.12.

We first performed 60 experiments with three objects individually presented to the robot. We presented the robot with the object placed within reach from different starting locations along with a language instruction. For 20 experiments per each object, locations of the robot as well as the object were changed. Each object was also presented with two different natural language instruction. Given a raw point-cloud data, the robot has to 1) segment and identify appropriate object part, 2) infer manipulation trajectory using our deep multimodal embedding algorithm, and 3) execute inferred trajectory.

Table 5.2 shows the results of the experiment on three objects, presenting success rate of each algorithm of end-to-end system. The segmentation ranking algorithm performed well at 83.3% on average but it was not as reliable for the beverage dispenser which has a small lever. Note that our main contribution of this work, the idea of transferring manipulation trajectory using deep multimodal embedding algorithm, has a success rate of 94.4%, 100.0% and 78.9%. The door handle had slightly lower success rate because the sensor (Kinect) failed

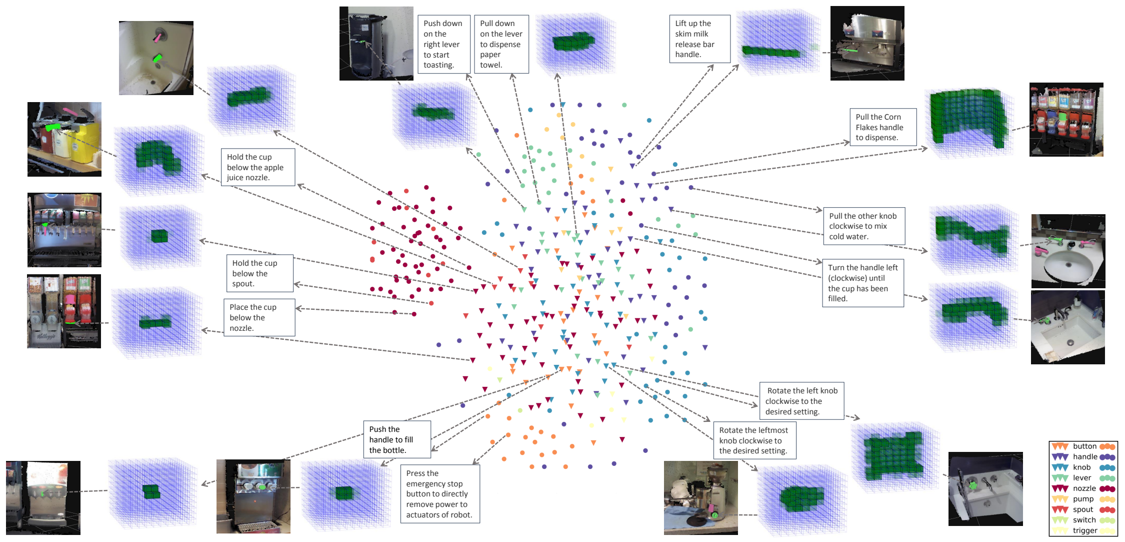


Figure 5.14: **Learned Point-cloud/Language Space:** Visualization of the point-cloud/language layer $h^{2,lp}$ in 2d using t-SNE [166] after the network is fully fine-tuned. Inverted triangles represent projected point-clouds and circles represent projected instructions. A subset of the embedded points are randomly selected for visualization. Since 3D point-clouds of object parts are hard to visualize, we also include a snapshot of a point-cloud showing the whole object. Notice correlations in the motion required to manipulate the object or follow the instruction among nearby point-clouds and natural language.

to pick up enough structure that makes it look like a door handle, especially missing the majority of cylindrical part that connects to the door. Our robot was then able to correctly follow these trajectories with a few occasional slips due to the relatively large size of its gripper compared to the object parts.

Visual Distraction Experiments. We now introduce our robot with visual distractions so that there are also other potential target objects. We present robot with two manipulable objects, coffee dispenser with a handle and beverage dispenser with a lever.

The result of 40 experiments are shown in Table 5.3. These two objects were both new to the robot and our training data did not contain any similar scene

Table 5.2: **Results** of 60 experiments on a PR2 robot running end-to-end experiments autonomously on three different objects.

Success Rate of Each Step	**Dispenser Handle**	**Beverage Lever**	**Door Handle**	**Avg.**
1) Segmentation	90.0%	65.0%	95.0%	83.3%
2) DME Traj. Inference	94.4%	100.0%	78.9%	91.1%
3) Execution of Traj.	82.4%	76.9%	100.0%	86.4%

that had these two types of object parts together. However, the reliability of segmentation ranking algorithm only dropped slightly from 83.3% to 75.0%. Our main contribution of deep multimodal embedding algorithm and its execution on an arm performed well at 87.1% and 88.5%.

All videos of robotic experiments are available at following website: `http://robobarista.cs.cornell.edu/videos`

5.5.5 Transfer Experiments on Robot

We further evaluate our fundamental hypothesis that many novel objects can be manipulated by transferring manipulation trajectories for each part from completely different objects without any modification. To validate the idea, we tested with three complex objects the algorithm had never seen before—a coffee grinder, a lamp, and an espresso machine (Fig. 5.13).

For this experiment, for each object, the robot is presented with a pre-segmented point-cloud along with a natural language text manual, with each step in the manual associated with a segmented part in the point-cloud. Once our algorithm outputs a trajectory (transferred from a completely different object), we find the manipulation frame for the part's point-cloud by using its principal axis (Sec. 5.4.1). Then, the transferred trajectory is executed relative to the part using this coordinate frame, without any modification to the trajectory.

The chosen manipulation trajectory, defined as a set of waypoints, is converted to a smooth and densely interpolated trajectory (Sec. 5.2.1.) The robot first computes and execute a collision-free motion to the starting point of the manipulation trajectory. Then, starting from this first waypoint, the interpolated trajectory is executed. For these experiments, we placed the robot in reach of the object, but one could also find a location using a motion planner that would make all waypoints of the manipulation trajectory reachable.

Some of the examples of successful execution on a PR2 robot are shown in Fig. 5.13. For example, a manipulation trajectory from the task of "turning on a light switch" is transferred to the task of "flipping on a switch to start extracting espresso", and a trajectory for turning on DC power supply (by rotating clockwise) is transferred to turning on the floor lamp.

Our robot was able to even prepare a cup of latte by following 5 instruction steps with one hard-coded motion of putting portafilter into the espresso machine. These demonstrations shows that part-based transfer of manipulation trajectories is feasible without any modification to the source trajectories by carefully choosing their representation and coordinate frames (Sec. 5.2.2). Although some of these motions can definitely be more refined, it is surprising that a robot can even use an espresso machine, which is difficult even for a novice human user.

Table 5.3: **Results** of 40 visual distraction experiments on a PR2 robot running end-to-end experiments autonomously. The robot is presented with two different novel objects that can be manipulated. The robot has to identify the object part as specified from the natural language instruction without any visual hints.

Success Rate of Each Step	**Dispenser Handle**	**Beverage Lever**	**Avg.**
1) Segmentation	80.0%	70.0%	75.0%
2) DME Traj. Inference	81.3%	92.9%	87.1%
3) Execution of Traj.	92.3%	84.6%	88.5%

The video of making a cup of latte is available at the project website: `http://robobarista.cs.cornell.edu/videos`

5.6 Conclusion

In order for a robot to reason about manipulation tasks for various objects in human environments, the robot has to reason about various modalities of information. In this chapter, we propose a novel approach to predicting manipulation trajectories via object-part based transfer using deep multimodal embedding. Our algorithm even allows robots to successfully manipulate objects they have never seen before. We formulate this as a structured-output problem and approach the problem of inferring manipulation trajectories for novel objects by jointly embedding point-cloud, natural language, and trajectory data into a common space using deep neural networks. We introduce a method for learning a semantically meaningful common representation of multimodal data using a loss-augmented cost function. We also introduce a method for pre-training the network's lower layers, learning embeddings for subsets of modalities, and show that it outperforms standard pre-training algorithms. Learning such an embedding space allows efficient inference by comparing the embedding of a new point-cloud/language pair against pre-embedded demonstrations. To overcome the challenge of collecting large-scale manipulation dataset, we introduce our crowd-sourcing platform, Robobarista, which allows non-expert users to easily give manipulation demonstrations over the web. This enables us to collect a large-scale dataset of 249 object parts with 1225 crowd-sourced demonstrations, on which our algorithm outperforms other methods. Finally, we also verify on our robot that even manipulation trajectories transferred from completely different objects can be used to successfully manipulate novel objects the robot has never seen before.

CHAPTER 6

LEARNING TO REPRESENT HAPTIC FEEDBACK FOR PARTIALLY-OBSERVABLE TASKS

When a robot performs a task on a complex object, a visual or a language instruction may not be sufficient, and it may require incorporating haptic feedback. In previous chapters, we discussed a method for building a representation of different modalities that can assist in planning a complex motion. In this chapter, we focus on building a representation of haptic feedback for the purpose of modifying plans while manipulating objects.

Many tasks in human environments that we do without much effort require more than just visual observation. Very often they require incorporating the sense of touch to complete the task. For example, consider the task of turning a knob that needs to be rotated until it clicks, like the one in Figure 6.1. The robot could observe the consequence of its action if any visible changes occur, but such clicks can often only be directly observed through the fingers. Many of the objects that surround us are explicitly designed with feedback – one of the key interaction design principles – otherwise "one is always wondering whether anything has happened" [116].

Recently, there has been a lot of progress in making robots understand and act based on images [86, 173, 103] and point-clouds [148]. A robot can definitely gain a lot of information from visual sensors, including a nominal trajectory plan for a task [148]. However, when the robot is manipulating a small object or once the robot starts interacting with small parts of appliances, self-occlusion by its own arms and its end-effectors limits the use of the visual information.

However, building an algorithm that can examine haptic properties and incorporate such information to influence a motion is very challenging for multiple reasons. First, haptic feedback is a dynamic response that is dependent on the action the robot has taken on the object as well as internal states and properties of the object. Second, every haptic sensor produces a vastly different raw sensor signal.

Moreover, compared to the rich information that can be extracted about a current state of the task from few images (e.g. position and velocity information of an end-effector and an object [103, 86]), a short window of haptic sensor signal is merely a partial consequence of the interaction and of the changes in an unobservable internal mechanism. It also suffers from perceptual aliasing – *i.e.* many segments of a haptic signal at different points of interaction can produce a very similar signal. These challenges make it difficult to design an algorithm that can incorporate information from haptic modalities (in our case, tactile sensors).

In this work, we introduce a framework that can learn to represent haptic feedback for tasks requiring incorporation of a haptic signal. Since a haptic signal only provides a partial observation, we model the task using a partially observable Markov decision process (POMDP). However, since we do not know of definition of states for a POMDP, we first learn an appropriate representation from a haptic signal to be used as continuous states for a POMDP. To overcome the intractability in computing the posterior, we employ a variational Bayesian method, with a deep recurrent neural network, that maximizes lower bound of likelihood of the training data.

Using a learned representation of the interaction with feedback, we build on

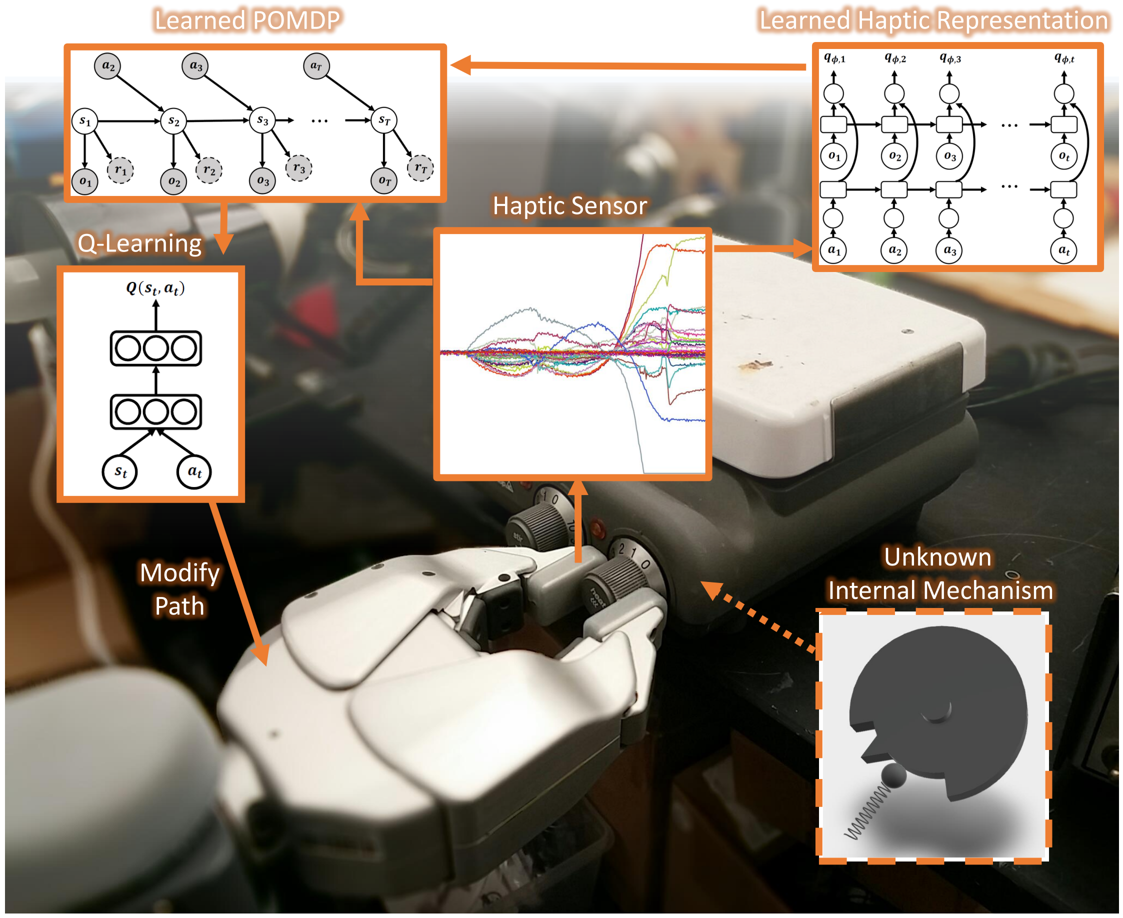


Figure 6.1: **Haptic feedback** from a tactile sensor being used to modify a nominal plan of manipulation. Our framework learns an appropriate representation (embedding space) which in turn is used to learn to find optimal control.

deep Q-learning [103] to identify an appropriate phase of the action from a provided nominal plan. Unlike most other applications of successful reinforcement learning [103, 139], the biggest challenge is a lack of a robotics simulation software that can generate realistic haptic signals for a robot to safely simulate and explore various combinations of states with different actions.

To validate our approach, we collect a large number of sequences of haptic feedback along with their executed motion for the task of 'turning a knob until it clicks' on objects of various shapes. We empirically show on a PR2 robot that we can modify a nominal plan and successfully accomplish the task using the learned models, incorporating tactile sensor feedback on the fingertips of the robot. In summary, the key contributions of this chapter are:

- We introduce an algorithm which learns task relevant representation of haptic feedback.
- We present a framework for modifying a nominal manipulation plan for interactions that involves haptic feedback.
- We present an algorithm for learning optimal actions with limited data without simulator.

6.1 Related Work

Haptics. Haptic sensors mounted on robots enable many different interesting applications. Using force and tactile input, a food item can be classified with characteristics which map to appropriate class of motions [38]. Haptic adjectives such as 'sticky' and 'bumpy' can be learned with biomimetic tactile sensors [18]. Whole-arm tactile sensing allows fast reaching in dense clutter. We focus on tasks with a nominal plan (e.g. [148]) but requires incorporating haptic (tactile) sensors to modify execution length of each phase of actions.

For closed-loop control of robot, there is a long history of using different feedback mechanisms to correct the behavior [10]. One of the common approaches that involves contact relies on stiffness control, which uses the pose of an end-effector as the error to adjust applied force [133, 6]. The robot can even self-tune its parameters for its controllers [160]. A robot also uses the error in predicted pose for force trajectories [85] and use vision for visual servoing [17].

Haptic sensors have also been used to provide feedback. A human operator with a haptic interface device can teleoperate a robot remotely [117]. Features extracted from tactile sensors can serve as feedback to planners to slide and roll objects [91]. [119] uses tactile sensor to detect success and failure of manipulation task to improve its policy.

Partial Observability. A POMDP is a framework for a robot to plan its actions under uncertainty given that the states are often only obtained through noisy sensors [157]. The framework has been successfully used for many tasks including navigation and grasping [52, 78]. Using wrist force/torque sensors, hierarchical POMDPs help a robot localize certain points on a table [168]. While for some problems [52], states can be defined as continuous robot configuration space, it is unclear what the ideal state space representation is for many complex manipulation tasks.

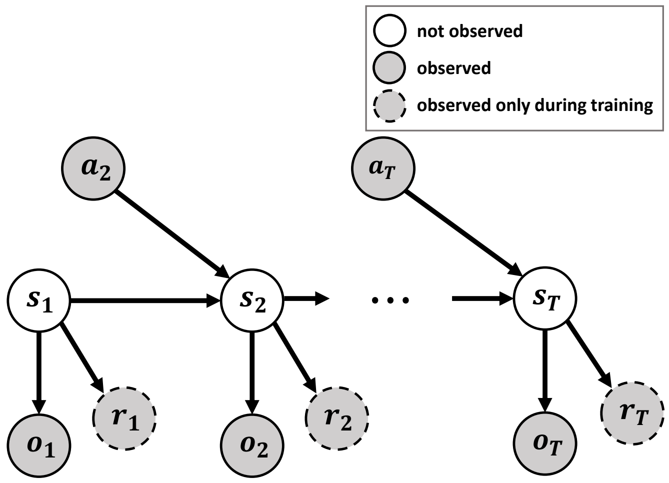
When the knowledge about the environment or states is not sufficient, [134] use a fully connected DBN for learning factored representation online, while [19] employ a two step method of first learning optimal decoder then learning to encode. While many of these work have access to a good environment model, or is able to simulate environment where it can learn online, we cannot explore or simulate to learn online. Also, the reward function is not available. For training purposes, we perform privileged learning [167] by providing an expert reward label only during the training phase.

Representation Learning. Deep learning has recently vastly improved the performance of many related fields such as compute vision (e.g. [76]) and speech recognition (e.g. [48]). In robotics, it has helped robots to better classify haptic adjectives by combining images with haptic signals [37], predict traversability from long-range vision [47], and classify terrains based on acoustics [164].

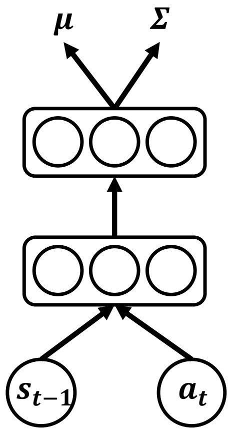
For controlling robots online, a deep auto-encoder can learn lower-dimensional embedding from images and model-predictive-control (MPC) is used for optimal control [171]. DeepMPC [85] predicts its future end-effector position with a recurrent network and computes an appropriate amount of force. Convolutional neural network can be trained to directly map images to motor torques [86, 34]. As mentioned earlier, we only take input of haptic signals, which suffers from perceptual aliasing, and contains a lot less information in a single timestep compared to RGB images.

Recently developed variational Bayesian approach [70, 127], combined with a neural network, introduces a recognition model to approximate intractable true posterior. Embed-to-Control [173] learns embedding from images and transition between latent states representing unknown dynamical system. Deep Kalman Filter [75] learns very similar temporal model based on Kalman Filter but is used for counterfactual inference on electronic health records.

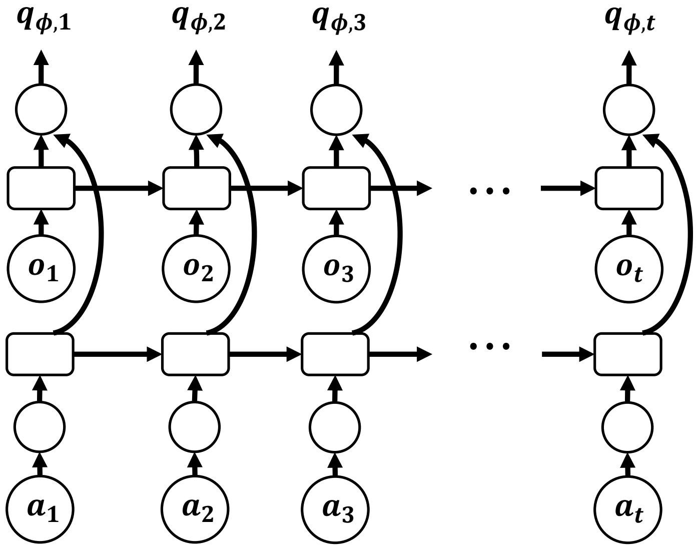
Reinforcement learning (RL), also combined with a neural network, has recently learned to play computer games by looking at pixels [103, 49]. Applying standard RL to a robotic manipulation task, however, is challenging due to lack of suitable state space representation [34]. Also, most RL techniques rely on trial and error [152] with the ability to try different actions from different states and



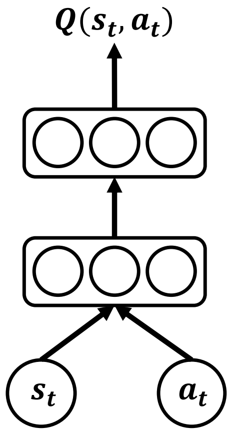
(a) **Graphical Model Rep. of POMDP Model**



(b) **Tran. Network**



(c) **Deep Recurrent Recognition Network**



(d) **Deep Q Network**

Figure 6.2: **Framework Overview.** We model the task that requires incorporation of tactile feedback in a partially observable MDP (a) which its transition and emission functions are parametrized by neural networks (b). To find an appropriate representation of states for the POMDP, we approximate the posterior with a Deep Recurrent Recognition Network (c), consisting of two LSTM (square blocks) recurrent networks. Deep Q-Network (d), consisting of two fully connected layers, utilizes a learned representation from (c) and a learned transition model from (a) to train Deep Q-Network (d).

observe reward and state transition. However, for many of the robotic manipulation tasks that involve physical contact with the environment, it is too risky to let an algorithm try different actions, and reward is not trivial without instrumentation of the environment for many tasks. In this work, the robot learns to represent haptic feedback and find optimal control from limited amount of haptic sequences despite lack of good robotic simulator for haptic signal.

6.2 Our Approach

Our goal is to build a framework that allows robots to represent and reason about haptic signals generated by its interaction with an environment.

Imagine you were asked to turn off the hot plate in Figure 6.1 by rotating the knob until it clicks. In order to do so, you would start by rotating the knob clockwise or counterclockwise until it clicks. If it doesn't click and if you feel the wall, you would start to rotate it in the opposite direction. And, in order to confirm that you have successfully completed the task or hit the wall, you would use your sense of touch on your finger to feel a click. There could also be a sound of a click as well as other observable consequences, but you would not feel very confident about the click in the absence of haptic feedback.

However, such haptic signal itself does not contain sufficient information for a robot to directly act on. It is unclear what is the best representation for a state of the task, whether it should only be dependent on states of internal mechanisms of the object (which are unknown) or it should incorporate information about the interaction as well. The haptic signal is merely a noisy partial observation of latent states of the environment, influenced by many factors such as a type of interaction that is involved and a type of grasp by the robot.

To learn an appropriate representation of the state, we first define our manipulation task as a POMDP model. However, posterior inference on such latent state from haptic feedback is intractable. In order to approximate the posterior, we employ variational Bayes methods to jointly learn model parameters for both a POMDP and an approximate posterior model, each parametrized by a deep recurrent neural network.

Another big challenge is the limited opportunity to explore with different policies to fine-tune the model, unlike many other applications that employs POMDP or reinforcement learning. Real physical interactions involving contact are too risky for both the robot and the environment without lots of extra safety measures. Another common solution is to explore in a simulated environment; however, none of the available robot simulators, as far as we are aware, are capable of generating realistic feedback for objects of our interest.

Instead, we learn offline from previous experiments by utilizing a learned haptic representation along with its transition model to explore offline and learn Q-function.

6.2.1 Problem Formulation

Given a sequence of haptic signals ($\vec{o} = o_1, ..., o_t$) up to current time frame t along with a sequence of actions taken ($\vec{a} = a_1, ..., a_t$), our goal is to output a sequence of appropriate state representations ($\vec{s} = s_1, ..., s_t$) such that we can take an optimal next action a_{t+1} inferred from the current state s_t.

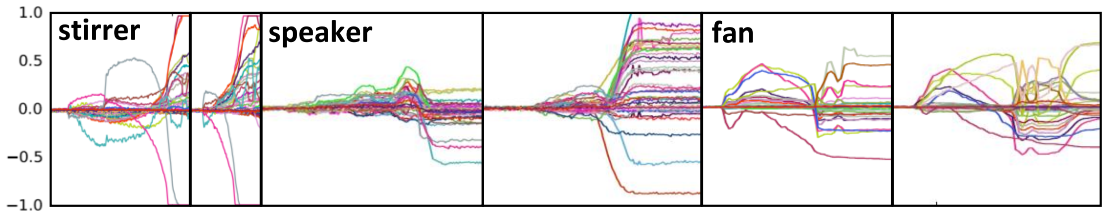


Figure 6.3: **Samples of haptic signals** from three different objects with a PR2 fingertip tactile sensor. Each graph shows a normalized temporal sequence of signals from both tips of the finger. Notice a large variation in feedback produced by what humans identify as a 'click'.

6.2.2 Generative Model

We formulate the task that requires haptic feedback as a POMDP model, defined as (S, A, T, R, O). S represents a set of states, A represents a set of actions, T represents a state transition function, R represents a reward function, and O represents an observation probability function. Fig. 6.2a represents a graphical model representation of a POMDP model and all notations are summarized in Table 6.1.

Among the required definitions of a POMDP model, most importantly, state S and its representation are unknown. Thus, all functions T, R, O that rely on states S are also not available.

We assume that all transition and emission probabilities are distributed as Gaussian distributions; however, they can take any appropriate distribution for the application. Mean and variance of each distribution are defined as a function

Table 6.1: Summary of Notations.

Notations	Descriptions
S	continuous state space (a learned representation)
O	observation probability ($S \rightarrow O$) of haptic signal
T	conditional probability between states ($S \times A \rightarrow S$)
A	a set of possible actions to be taken at each time step
R	a reward function ($S \rightarrow \mathbb{R}$)
p_θ	a generative model for O and R
θ	parameters of generative model
q_ϕ	an approximate posterior distribution
	(a recognition network for representing haptic signal)
ϕ	parameters of recognition network (recurrent neural network)
$Q(s, a)$	an approximate action-value function ($S \times A \rightarrow \mathbb{R}$)
γ	a discount factor

with input as parent nodes in the graphical model (Fig. 6.2a):

$$\begin{aligned}
s_1 &\sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\
s_t &\sim \mathcal{N}(f_{s_\mu}(s_{t-1}, a_t), f_{s_\Sigma}(s_{t-1}, a_t)^2\mathbf{I}) \\
o_t &\sim \mathcal{N}(f_{o_\mu}(s_t), f_{o_\Sigma}(s_t)^2\mathbf{I}) \\
r_t &\sim \mathcal{N}(f_{r_\mu}(s_t), f_{r_\Sigma}(s_t)^2\mathbf{I})
\end{aligned}$$

We parametrize each of these functions as a neural network. Fig. 6.2b shows a two layer network for parametrization of the transition function, and emission networks take a similar structure. The parameters of these networks form the parameters of the generative model $\theta = \{s_\mu, s_\Sigma, o_\mu, o_\Sigma, r_\mu, r_\Sigma\}$.

6.2.3 Deep Recurrent Recognition Network

Due to non-linearity of multi-layer neural network, computing the posterior distribution $p(\vec{s}|\vec{o}, \vec{r}, \vec{a})$ becomes intractable [75]. The variational Bayes method [70, 127] allows us to approximate the real posterior distribution with a recognition network (encoder) $q_\phi(\vec{s}|\vec{o}, \vec{r}, \vec{a})$.

Although it is possible to build a recognition network $q_\phi(\vec{s}|\vec{o}, \vec{r}, \vec{a})$ that takes the reward $\vec{r}$ as a part of the input, such recognition network would not be useful during a test time when the reward $\vec{r}$ is not available. Since a reward is not readily available for many of the interaction tasks, we assume that the sequence of rewards $\vec{r}$ is available only during a training phase given by a expert. Thus, we build an encoder $q_\phi(\vec{s}|\vec{o}, \vec{a})$ without a reward vector while our goal will be to reconstruct a reward $\vec{r}$ as well (Sec. 6.2.4).

Among many forms and structures q_ϕ could take, through validation with our dataset, we chose to define $q_{\phi,t}(s_t|o_1, ..., o_t, a_1, ..., o_t)$ as a deep recurrent network with two long short-term memory (LSTM) layers as shown in Fig. 6.2c.

6.2.4 Maximizing Variational Lower-bound

To jointly learn parameters for the generative θ and the recognition network ϕ, our objective is to maximize likelihood of the data:

$$max_\theta[log\ p_\theta(\vec{o}, \vec{r}|\vec{a})]$$

Using a variational method, a lower bound on conditional log-likelihood is defined as:

$$\begin{aligned} log\ p_\theta(\vec{o},\vec{r}|\vec{a}) &= D_{KL}(q_\phi(\vec{s}|\vec{o},\vec{r},\vec{a})||p_\theta(\vec{s}|\vec{o})) + \mathcal{L}(\theta,\phi) \\ &\geq \mathcal{L}(\theta,\phi) \end{aligned}$$

Thus, to maximize $max_\theta[log\ \ p_\theta(\vec{o},\vec{r}|\vec{a})]$, the lower bound $\mathcal{L}(\theta,\phi)$ can instead be maximized.

$$\begin{aligned} \mathcal{L}(\theta,\phi) = &-D_{KL}(q_\phi(\vec{s}|\vec{o},\vec{r},\vec{a})||p_\theta(\vec{s}|\vec{a})) \\ &+ \mathbb{E}_{q_\phi(\vec{s}|\vec{o},\vec{r},\vec{a})}[log\ p_\theta(\vec{o},\vec{r}|\vec{s},\vec{a})] \end{aligned} \tag{6.1}$$

Using a reparameterization trick [70] twice, we arrive at following lower bound (refer to Appendix for full derivation):

$$\begin{aligned} \mathcal{L}(\theta,\phi) \approx &-D_{KL}(q_\phi(s_1|\vec{o},\vec{r},\vec{a})||p(s_1)) \\ &-\frac{1}{L}\sum_{t=2}^{T}\sum_{l=1}^{L}[D_{KL}(q_\phi(s_t|s_{t-1},\vec{o},\vec{r},\vec{a})||p(s_t|s_{t-1}^{(l)},u_{t-1}))] \\ &+\frac{1}{L}\sum_{l=1}^{L}[log\ p_\theta(\vec{o}|\vec{s}^{(l)}) + log\ p_\theta(\vec{r}|\vec{s}^{(l)})] \\ &\text{where } \vec{s}^{(l)} = g_\phi(\epsilon^{(l)},\vec{o},\vec{r},\vec{a}) \text{ and } \epsilon^{(l)} \sim p(\epsilon) \end{aligned} \tag{6.2}$$

We jointly back-propagate on neural networks for both sets of encoder ϕ and decoder θ parameters with mini-batches to maximize the lower bound using AdaDelta [187].

6.2.5 Optimal Control in Learned Latent State Space

After learning a generative model for the POMDP and a recognition network using a variational Bayes method, we need an algorithm for making an optimal

Algorithm 4 Deep Q-Learning in Learned Latent State Space

```
$D_{gt} = \{\}$                                   ▷ "ground-truth" transitions by $q_\phi$
for all timestep $t$ of $(\vec{o}, \vec{a})$ in training data $(i)$ do
    $s_t, s_{t+1} \leftarrow q_{\phi,\mu} + \epsilon\, q_{\phi,\Sigma}$ where $\epsilon \sim p(\epsilon)$
    $D_{gt} \leftarrow D_{gt} \cup \langle s_t^{(i)}, a_{t+1}^{(i)}, r_{t+1}^{(i)}, s_{t+1}^{(i)} \rangle$
end for
loop
    $D_{explore} = \{\}$                          ▷ explore with learned transition
    for all $s_t^{(i)}$ in training data that succeeded do
        $a_{t+1} = \begin{cases} rand(a \in A) & \text{with prob. } \epsilon \\ \operatorname{argmax}_{a \in A} Q(s_t^{(i)}, a) & \text{otherwise} \end{cases}$
        $r_{t+1} = \begin{cases} r_t^{(i)} & \text{if } a_{t+1} == a_{t+1}^{(i)} \\ -1 & \text{otherwise} \end{cases}$
        $s_{t+1} \leftarrow T(s_t^{(i)}, a_t)$
        $D_{explore} \leftarrow D_{explore} \cup \langle s_t^{(i)}, a_{t+1}, r_{t+1}, s_{t+1} \rangle$
    end for
    $D \leftarrow D_{gt} \cup D_{explore}$                ▷ update deep Q-network
    for all minibatch from $D$ do
        $y_t \leftarrow r_t + \gamma max_{a'} Q(s_{t+1}, a')$
        Take gradient with loss $[y_t - Q(s_t, a_{t+1})]^2$
    end for
end loop
```

decision in learned representation of haptic feedback and action. We employ a reinforcement learning method, Q-Learning, which learns to approximate an optimal action-value function [152]. The algorithm computes a score for each state action pair:

$$Q : S \times A \to \mathbb{R}$$

The Q function is approximated by a two layer neural network as shown in Fig. 6.2d.

In a standard reinforcement learning setting, in each state s_t, an agent learns by exploring the selected action $\text{argmax}_{a \in A} Q(s_t, a)$ with a current Q function. However, doing so requires an ability to actually take or simulate the chosen action from s_t and observe r_{t+1} and s_{t+1}. However, there does not exist a good robotics simulation software that can simulate complex interactions between a robot and an object and generate different haptic signals. Thus, we cannot freely explore any states.

Instead, we first take all state transitions and rewards $\langle s_t^{(i)}, a_{t+1}^{(i)}, r_{t+1}^{(i)}, s_{t+1}^{(i)} \rangle$ from the i-th training data sequence and store in D_{gt}. Both $s_t^{(i)}$ and $s_{t+1}^{(i)}$ are computed by the recognition network q_ϕ with a reparameterization technique (similar to Sec. 6.2.4).

At each iteration, we first have an exploration stage. For explorations, we start from states $s_t^{(i)}$ of training sequences that resulted in successful completion of the task and choose an action a_{t+1} with ϵ-greedy. With the learned transition function T (Sec. 6.2.2), the selected action a_{t+1} is executed from $s_t^{(i)}$. However, since we are using a learned transition function, any deviation from the distribution of training data could result in unexpected state, unlike explorations in a real or a simulated environment.

Thus, if the optimal action a_{t+1} using a current Q-function deviates from the ground-truth action $a_{t+1}^{(i)}$, the action is penalized with a negative reward to pre-

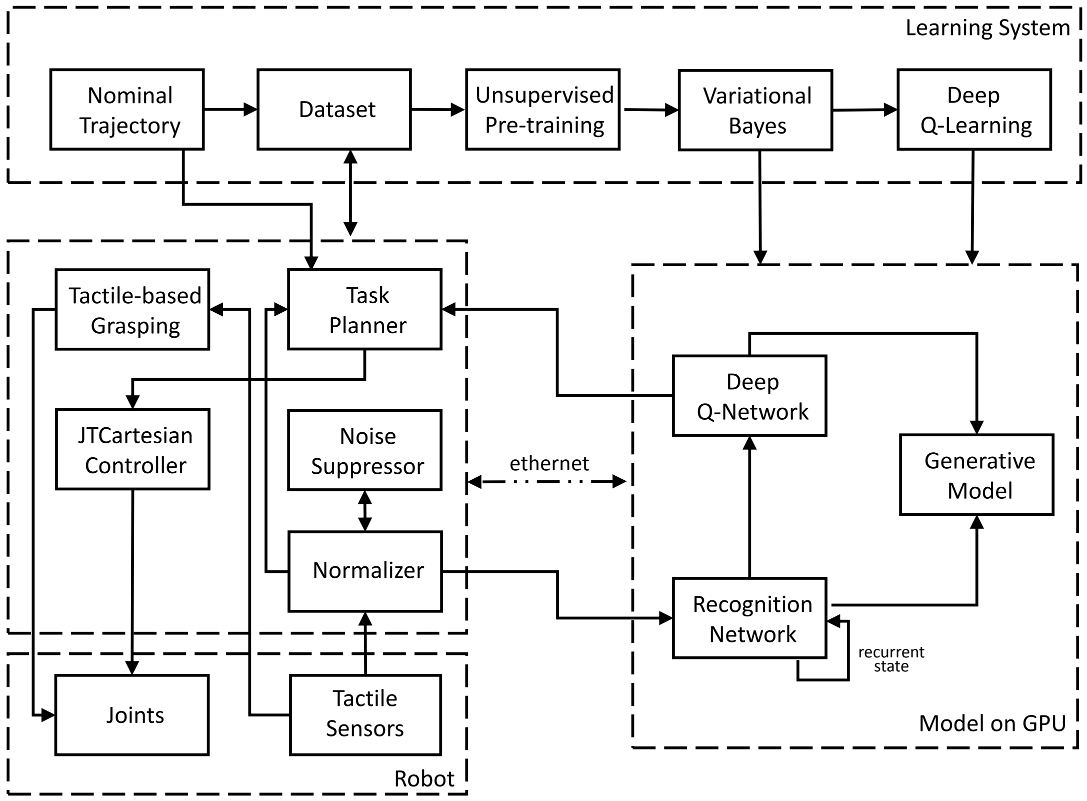


Figure 6.4: **System Details** of our system for learning and robotic experiments.

vent deviations into unexplored states. If the optimal action is same as the ground-truth, the same reward as the original is given. For such cases, the only difference from the ground-truth would be in s_{t+1}, which is inferred by the learned transition function. All exploration steps are recorded in $D_{explore}$.

After the exploration step in each iteration, we take minibatches from $D = D_{gt} \cup D_{explore}$ and backpropagate on the deep Q-network with the loss function:

$$[r_t + \gamma max_{a'} Q(s_{t+1}, a')] - Q(s_t, a_t)]^2$$

The algorithm is summarized in Algorithm 4.

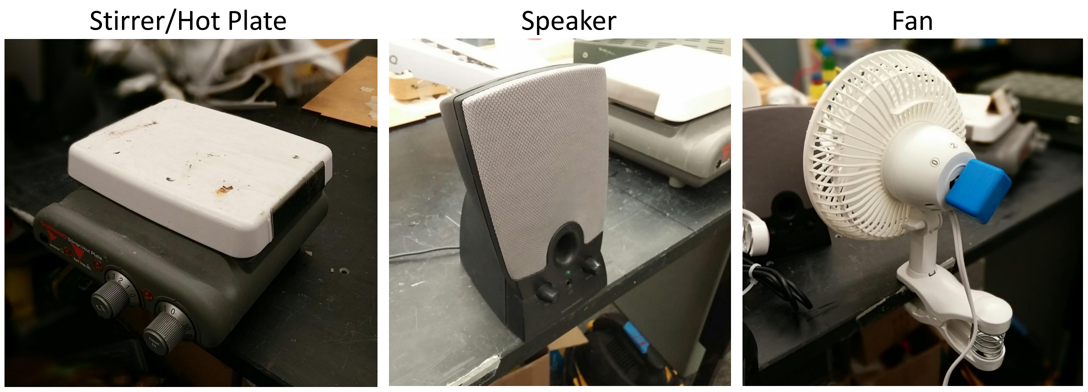


Figure 6.5: A set of objects used for experiment. All three objects have different surface area and shape, which results in vastly different types of *'clicks'* when observed via a tactile sensor.

6.3 System Details

Robotic Platform: All experiments were performed on a PR2 robot, a mobile robot with two 7 degree-of-freedom arms. Each two-fingered end-effector has an array of tactile sensors located at its tips. We used a Jacobian-transpose based JTCartesian controller [43] for controlling its arm during experiments.

For stable grasping, we take advantage of the tactile sensors to grasp an object. The gripper is slowly closed until certain thresholds are reached on both sides of the sensors, allowing the robot to easily adapt to objects of different sizes and shapes. To avoid saturating the tactile sensors, the robot does not grasp the object with maximal force.

Tactile Sensor: Each side of the fingertip of a PR2 robot is equipped with RoboTouch tactile sensor, an array of 22 tactile sensors covered by protective silicone rubber cover. The sensors are designed to detect range of 0 – 30 psi (0 – 205 kPa) with sensitivity of 0.1 psi (0.7 kPa) at the rate of 35 Hz.

Table 6.2: **Result of haptic signal prediction and robotic experiment.** The prediction experiment reports the average L2-norm from the haptic signal (44 signals in newtons) and the robotic experiment reports the success rate. It shows the results of more than 200 robotic experiments.

	Haptics Prediction			**Robotic Experiment**		
	0.05$secs$	0.25$secs$	0.50$secs$	*Stirrer*	*Speaker*	*Desk Fan*
Chance	6.68 (±0.18)	6.68 (±0.17)	6.69 (±0.18)	31.6%	38.1%	28.5%
Non-recurrent Recognition Network [173]	1.39 (±2.51)	5.03e5 (±5.27e7)	3.23e7 (±1.07e10)	52.9%	57.9%	62.5%
Recurrent-network as Representation [85]	**0.33** (±0.01)	1.01 (±0.09)	1.76 (±0.03)	63.2%	68.4%	70.0%
Our Model without Exploration	-	-	-	35.0%	33.3%	52.6%
Our Model	0.72 (±0.08)	**0.79** (±0.09)	**0.78** (±0.10)	**80.0%**	**73.3%**	**86.7%**

We observed that each of the 44 sensors has a significant variation and noise in raw sensor readings with drifts over time. To handle such noise, values are first offset by starting values when interaction between an object and the robot started (*i.e.* when a grasp occurred). Given the relative signals, we find a normalization value for each of 44 sensors such that none of the values goes above 0.05 when stationary and all data is clipped to the range of -1 and 1. Normalization takes place by recording few seconds of sensor readings after grasping.

Learning Systems: For fast computation and executions, we offload all of our models onto a remote workstation with a GPU connected over a direct ethernet connection. Our models run on a graphics card using Theano [7], and our high level task planner sends a new goal location at the rate of 20 Hz. The overall system detail is shown in Figure 6.4.

6.4 Experiments & Results

In order to validate our approach, we perform a series of experiments on our dataset and on a PR2 robot.

6.4.1 Dataset

In order to test our algorithm that learns to represent haptic feedback, we collected a dataset of three different objects – a stirrer, a speaker, and a desk fan (Fig. 6.5) – each of which have a knob with a detent structure (an example CAD model shown in Fig. 6.1). Although these objects internally have some type of a detent structure that produce a feedback that humans would identify as a 'click', each 'click' from each object is very distinguishable to humans. Different shapes of objects and the flat surface of the two fingers result in vastly differently tactile sensor readings as shown in Fig. 6.3.

In our model, for the haptic signals $\vec{o}$, we use a vector of 44 tactile sensor array as described in Sec. 6.3. The reward $\vec{r}$ is given as one of three classes at each time step, representing a positive, a negative and a neutral reward. For every object, actions $\vec{a}$ are numbers representing phases in its nominal plan, with each phase as a binary variable.

In more detail, the **stirrer** (hot plate) has a knob with a diameter of 22.7*mm* with a depth of 18.7*mm*, and the haptic feedback requires and lasts about 30° rotations to turn it on or turn it off. We start from both left (off state) and right side (on state) of the knob. The **speaker** has a small cylindrical knob that decreases in its diameter from 13.1*mm* to 9.1*mm* with height of 12.8*mm* and requires 30° degree rotation. However, given that PR2 fingertips with silicon cover measures 23*mm* and are parallel plates, grasping a tiny 9.1*mm* knob results in drastically different sensor readings at every execution of the task. The **desk fan** has a square knob with a two-step detent control that has a width of 25.1*mm* and a large surface area. The click lasts 45° degree rotation and has a narrow stoppable window of about ±20° degrees. To test whether our model has learned the concept of clicking once, if it goes beyond the first stopping point, resulting in two clicks, it is considered as a failure.

The stirrer and the speaker can be rotated both clockwise and counterclockwise with each having a wall at both ends. The desk fan has three stoppable points (near 0°, 45°, and 90°) to adjust fan speed and could get stuck in-between if the rotation is not enough or exceeds the stopping point.

Each object has an associated multiple phases of nominal path which is defined as a set of smoothly interpolated waypoints consisting of end-effector position and orientation along with gripper actions of grasping similar to [148]. For each of the objects, we collected at least 25 successes and 25 total failures. The failures include slips, excessive rotations beyond acceptable range, rotation even after hitting a wall, and near breaking of the knob. Also included were trajectories that resulted in failure in less dramatic manner such as insufficient rotations. Each data sequence consists of a sequence of trajectory phases as well as tactile sensor signals after each execution of the waypoint.

To label the reward for each sequence afterwards, an external camera with a microphone was placed nearby the object. The expert labeled the timeframe for each sequence that succeeded or failed by reviewing the audio and visually inspecting haptic signal. These recordings were only used for labeling the data, and such camera or microphone is not made available to robot during our experiments. For sequences that turned the knob past the successful stage but did not stop rotating the knob, only negative rewards were given.

Of multiple phases in a nominal plan, which includes pre-grasping and post-interaction trajectories, we focus on two or three phases (rotation and stopping phases). These phases occur after grasping and success is determined by ability to detect when to shift to the last phase (a shift from interaction to post-interaction phase).

6.4.2 Baselines

We compare our model against several baseline methods on our dataset and for robotic experiment. Since most of the related works are applied to problems in different domains, we take key ideas (or key structural differences) from relevant works and fit them to our problem.

1) *Chance:* It follows a nominal plan and makes a transition between phases by randomly selecting the amount of degree to rotate a knob without incorporating haptic feedback.

2) *Non-recurrent Recognition Network:* Similar to [173], we take non-recurrent deep neural network of only observations without actions. However, it has access to a short history in a sliding window of haptic signal at every frame. For control, we apply the same Q-learning method as our full model.

3) *Recurrent Network as Representation:* Similar to [85], we directly train a recurrent network to predict future haptic signals. At each time step t, the LSTM

network takes concatenated observation o_t and previous action a_t as input, and the output of LSTM is concatenated with a_{t+1} to predict o_{t+1}. However, while [85] relies on hand-coded MPC cost function to choose an action, we apply same Q-learning that was applied to our full model. For haptic prediction experiment, transitions happen by taking output of the next time step as input to the next observation.

4) *Our Model without Exploration:* During the final deep Q-Learning (Sec. 6.2.5) stage, it skips the exploration step that uses a learned transition model and only uses sequences of representation from the recognition network.

6.4.3 Results and Discussion

To evaluate all models, we perform two types of experiments – haptic signal prediction and robotic experiment.

Haptic Signal Prediction: We first compare our model against baselines on a task of predicting future haptic signal. For all sequences that either eventually succeeded or failed, we take every timestep t, and predict timestep $t + 1$ ($0.05 secs$), $t + 5$ ($0.25 secs$) and $t + 10$ ($0.5 secs$). The prediction is made by encoding (recognition network) a sequence up to time t and then transiting encoded states with a learned transition model to the future frames of interest. We take the L2-norm of the prediction of 44 sensor values (which are in newtons) and take the average of that result. The result is shown in the middle column of Table 6.2.

Robotic Experiment: We also test on a PR2 robot the task of turning the knob until it clicks on three different objects – stirrer, speaker, and desk fan (Fig. 6.5). The right hand side of Table 6.2 shows results of over 200 executions. Each algorithm was tested on each object at least 15 times.

Can it predict future haptic signals? When it predicts randomly (*chance*), regardless of the timestep, it has an average of 6.7. When the primary goal is to be able to perform the next haptic signal prediction, for one step prediction, recurrent-network as representation baseline performs best of 0.330 among all models, while ours performed 0.718. On the other hand, our model does not diverge and performs consistently well. After $0.5 secs$, when other models started to diverge to an error of 1.757 or much larger, our model still had prediction error of 0.782.

What does learned representation represent? We visualize our learned embedding space of haptic feedback using t-SNE [165] in Fig. 6.6. Initially, both successful (blue paths) and unsuccessful (red paths) all starts from similar states but they quickly diverge into different clusters of paths much before they eventually arrive at states that were given positive or negative rewards shown as blue and red dots.

Does good representation lead to successful execution? Our model allows robot to successfully execute on the three objects 80.0%, 73.3%, and 86.7% respectively, performing the highest compared to any other models. The next best model which uses recurrent network as representation performed at 63.2%, 68.4%, and 70.0%. However, note that this baseline still take advantage of our Q-learning method. Our model that did not take advantage of simulated exploration performed much poorly (35.0%, 33.3%, and 52.6%), showing that good representation combined with our Q-learning method leads to successful

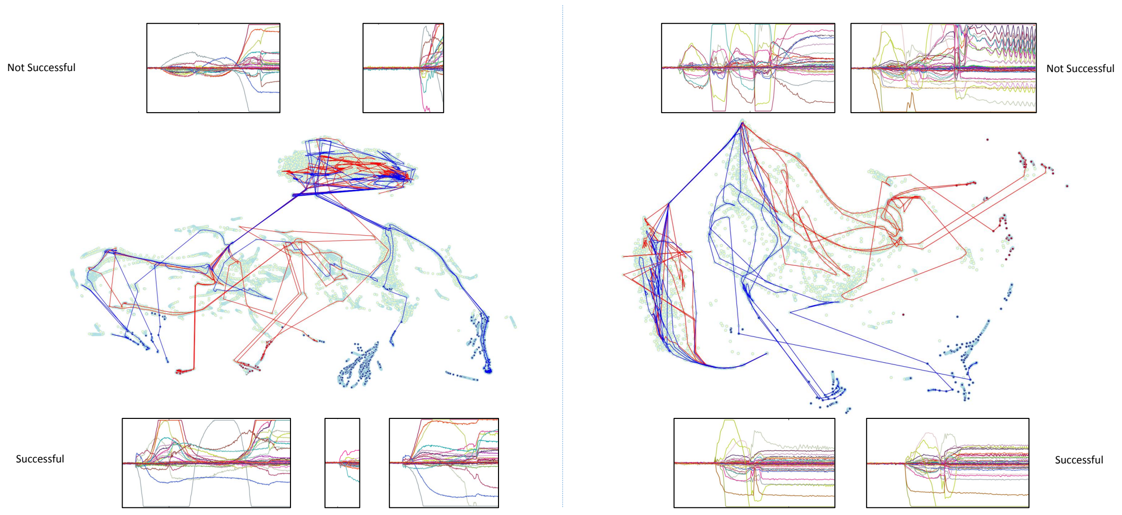


Figure 6.6: **Projection of learned representation** of haptic feedback using t-SNE [165] for 'stirrer' and 'fan'. Each dot represents an inferred state at each time frame, and blue and red dots represents positive and negative reward at those time frame. Here we show some of successful (blue) and unsuccessful (red) sequences. For both objects, notice both classes initially starts from similar state and then diverges, forming clusters. Several successful and unsuccessful haptic signals are shown as well.

execution of the tasks.

Is recurrent network necessary for haptic signals? *Non-recurrent recognition network* quickly diverged to extremely large number of $3.2e7$ even though it successfully predicted 1.389 for a single step prediction. Note that it takes windowed haptic sequence of last 5 frames as input. Unlike images, short window of data does not hold enough information about haptic sequence which lasts much longer timeframe. For robotic experiment, non-recurrent network performed 52.9%, 57.9%, and 62.5% even with our Q learning method.

How accurately does it perform the task? When our full model was being tested on three objects, we also had one of the author observe (visually and audibly) very closely and press a button as soon as the click occurs. On successful execution of the task, we measure the time difference between the time the robot

Table 6.3: **Time difference** between the time the robot stopped and the time the expert indicated it 'clicked'.

Stirrer	*Speaker*	*Desk Fan*
0.180 secs (±0.616)	0.539 secs (±1.473)	−0.405 secs (±0.343)

stops turning and the time the expert presses the key, and the results are shown in Table 6.3.

The positive number represents that the model was delayed than the expert and the negative number represents that the model transitioned earlier. Our model only differed from human with an average of 0.37 seconds. All executions of tasks were performed at same translational and rotational velocity as the data collection process. Since the robot was performing these tasks slowly, the 0.37 seconds delay results in very small amount of rotation.

Note that just like a robot has a reaction time to act on perceived feedback, an expert has a reaction time to press the key. Also, since the robot was relying on haptic feedback while the observer was using every possible human senses available including observation of the consequences without touch, some differences are expected. Especially, the fan had a delay in visible consequences compared to the haptic feedback because the robot was rotating these knobs slower than normal humans would turn in daily life; thus, the robot was able to react 0.4 seconds faster.

Video of robotic experiments are available at this website:
`https://sites.google.com/site/icra17haptics/`

6.5 Conclusion

For a robot to perform various manipulation tasks, a robot has to be able to model interplay of an object with the task being performed, including haptic feedback. In this chapter, we present a novel framework for learning to represent haptic feedback of an object that requires sense of touch. We model such tasks as partially observable model with its generative model parametrized by neural networks. To overcome intractability of computing posterior, variational Bayes method allows us to approximate posterior with a deep recurrent recognition network consisting of two LSTM layers. Using a learned representation of haptic feedback, we also introduce a Q-learning method that is able to learn optimal control without access to simulator in learned latent state space utilizing only prior experiences and learned generative model for transition. We evaluate our model on a task of rotating knobs until they click. With more than 200 robotic experiments on the PR2 robot, we show that our model is able to successfully manipulate knobs that click while predicting future haptic signals.

CHAPTER 7

CONCLUSION

As robots move into our environments at homes and offices, it is inevitable that robots will encounter objects it has never seen before. It is not feasible to train robots on all variations of objects in human environments. As we have shown, in order to even manipulate objects it has never encountered before, a robot has to reason about many sensor and information modalities including vision, natural language, manipulation motion, and haptic feedback.

In this dissertation, we introduced the new learning algorithms that address challenges of reasoning about multimodal data. Our deep multimodal embedding algorithm learns a joint-embedding of point-cloud, natural language, and manipulation trajectory. The algorithm trains deep neural networks to learn a semantically meaningful common representation of multimodal data using a loss-augmented cost function. We also introduced an algorithm for learning a representation of haptic feedback for partially observable task.

Furthermore, we proposed a novel approach to predicting manipulation trajectories by transferring motions from similarly operated object parts. We demonstrate that this idea allows robots to successfully manipulate novel objects, even preparing a cup of latte with an espresso machine and a coffee grinder that a robot has never seen before. We utilize learned embedding network to identify most appropriate manipulation trajectory for the point-cloud of the object and the natural language instruction of the task. To overcome the challenge of collecting large-scale manipulation dataset, we introduced our crowd-sourcing platform, Robobarista, which allows non-expert users to easily give manipulation demonstrations over the web.

7.1 Future Work

While my dissertation has shown the potential of identifying appropriate manipulation strategies for novel objects, there are many opportunities of broadening the scope of my work.

- **Closed-loop execution with various feedback:** For some of the objects, open-loop execution of a transferred pose trajectory sometimes may not be enough to correctly manipulate objects. In order to correctly execute a transferred manipulation trajectory on such objects, robots may have to incorporating diverse set of feedbacks including visual, auditory and force feedback [177, 169]. For such tasks, our algorithm can still provide an initialization of a required motion while the robot would incorporate different feedback to adapt its trajectory online.
- **Crowd-sourcing all sensor modalities:** In order for robots to manipulate novel objects in real environment, it would have to achieve nearly perfect accuracy due to dangers involved with having robots manipulate real world objects. Our work introduced one of the largest manipulation dataset using our crowd-sourcing platform. With recent advancement in computer vision and smart phones, it is now possible to also crowd-source collection of point-cloud and natural language instruction manual. As it was shown recently with large datasets in the field of computer vision and natural language processing, once we can scale our dataset to even larger size, our model would be able to achieve higher accuracy.
- **Connecting to reinforcement learning:** Even humans are bit rough when they are doing certain tasks for the first time, and they only get better after

several practices. Our algorithm can provide a fairly accurate transferred trajectory as the initial trajectory for reinforcement learning methods. Using the initial trajectory, a reinforcement learning algorithm can further polish the transferred trajectory in order for a robot to accomplish the task with more precisions. Such initialization would allow those algorithms to reach mastery of the task in a significantly lower number of steps.

APPENDIX A

APPENDIX OF LEARNING TO MANIPULATE NOVEL OBJECTS

A.1 Loss Function for Manipulation Trajectory

For both learning (Sec. 4.3) and evaluation (Sec. 5.5.3), we need a function which accurately represents distance between two trajectories. Prior metrics for trajectories consider only their translations (e.g. [73]) and not their rotations *and* gripper status. We propose a new measure, which uses dynamic time warping, for evaluating manipulation trajectories. This measure non-linearly warps two trajectories of arbitrary lengths to produce a matching, then computes cumulative distance as the sum of cost of all matched waypoints. The strength of this measure is that weak ordering is maintained among matched waypoints and that every waypoint contributes to the cumulative distance.

For two trajectories of arbitrary lengths, $\tau_A = \{\tau_A^{(i)}\}_{i=1}^{m_A}$ and $\tau_B = \{\tau_B^{(i)}\}_{i=1}^{m_B}$, we define a matrix $D \in \mathbb{R}^{m_A \times m_B}$, where $D(i, j)$ is the cumulative distance of an optimally-warped matching between trajectories up to index i and j, respectively, of each trajectory. The first column and the first row of D is initialized as:

$$D(i,1) = \sum_{k=1}^{i} c(\tau_A^{(k)}, \tau_B^{(1)}) \quad \forall i \in [1, m_A]$$
$$D(1,j) = \sum_{k=1}^{j} c(\tau_A^{(1)}, \tau_B^{(k)}) \quad \forall j \in [1, m_B]$$

where c is a local cost function between two waypoints (discussed later). The rest of D is completed using dynamic programming:

$$D(i,j) = c(\tau_A^{(i)}, \tau_B^{(j)}) + \min\{D(i-1, j-1), D(i-1, j), D(i, j-1)\}$$

Given the constraint that $\tau_A^{(1)}$ is matched to $\tau_B^{(1)}$, the formulation ensures that every waypoint contributes to the final cumulative distance $D(m_A, m_B)$. Also, given a matched pair $(\tau_A^{(i)}, \tau_B^{(j)})$, no waypoint preceding $\tau_A^{(i)}$ is matched to a waypoint succeeding $\tau_B^{(j)}$, encoding weak ordering.

The pairwise cost function c between matched waypoints $\tau_A^{(i)}$ and $\tau_B^{(j)}$ is defined:

$$\begin{aligned}
c(\tau_A^{(i)}, \tau_B^{(j)}; \alpha_T, \alpha_R, \beta, \gamma) &= w(\tau_A^{(i)}; \gamma) w(\tau_B^{(j)}; \gamma) \\
&\qquad \Big(\frac{d_T(\tau_A^{(i)}, \tau_B^{(j)})}{\alpha_T} + \frac{d_R(\tau_A^{(i)}, \tau_B^{(j)})}{\alpha_R}\Big)\Big(1 + \beta d_G(\tau_A^{(i)}, \tau_B^{(j)})\Big) \\
\text{where} \quad d_T(\tau_A^{(i)}, \tau_B^{(j)}) &= \|(t_x, t_y, t_z)_A^{(i)} - (t_x, t_y, t_z)_B^{(j)}\|_2 \\
d_R(\tau_A^{(i)}, \tau_B^{(j)}) &= \text{angle difference between } \tau_A^{(i)} \text{ and } \tau_B^{(j)} \\
d_G(\tau_A^{(i)}, \tau_B^{(j)}) &= \mathbb{1}(g_A^{(i)} = g_B^{(j)}) \\
w(\tau^{(i)}; \gamma) &= exp(-\gamma \cdot \|\tau^{(i)}\|_2)
\end{aligned}$$

The parameters α, β are for scaling translation and rotation errors, and gripper status errors, respectively. γ weighs the importance of a waypoint based on its distance to the object part. [1] Finally, as trajectories vary in length, we normalize $D(m_A, m_B)$ by the number of waypoint pairs that contribute to the cumulative sum, $|D(m_A, m_B)|_{path^*}$ (i.e. the length of the optimal warping path), giving the final form:

$$distance(\tau_A, \tau_B) = \frac{D(m_A, m_B)}{|D(m_A, m_B)|_{path^*}}$$

This distance function is used for noise-handling in our model and as the final evaluation metric.

[1]We assign $\alpha_T, \alpha_R, \beta, \gamma$ values of 0.0075 meters, 3.75°, 1 and 4 respectively.

A.2 Segmenting Object Parts from Point-clouds

Our algorithm, presented above (Sec. 4.2), assumes that object parts p corresponding to each natural language instruction l have already been segmented from the point-cloud scene s. While our focus here is on learning to manipulate these segmented parts, we also introduce a segmentation approach which both allows use to build a real-world end-to-end system and allows us to augment our training data for better unsupervised learning, as shown in Sec. 5.5.3.

A.2.1 Generating Object Part Candidates

We employ a series of geometric feature based techniques to segment a scene s into small overlapping segments $\{p_1, p_2, ..., p_n\}$.

We first extract Euclidean clusters of points while limiting the difference of normals between a local and larger region [55, 132]. We then filter out segments which are too big for human hands. To handle a wide variety of object parts of different scales, we generate two sets of candidates by executing our segmentation pipeline with two different sets of parameters, which are combined for evaluation.

A.2.2 Part Candidate Ranking Algorithm

Given a set of segmented parts p, we must now use our training data $\mathcal{D}$ to select for each instruction l_j the best-matching part p_j^*. We do so by optimizing the score $\psi(p_j, l_j)$ of each segment p_i for a step l_j in a manual, evaluated in three parts:

$$\psi(p_i, l_j; \mathcal{D}) = \psi_{feat}(p_i, l_j)(\psi_{pc}(p_i, l_j) + \psi_{lang}(p_i, l_j))$$

The score ψ_{pc} is based on the k_p-most identical segments from the training data $\mathcal{D}$, based on cosine similarity using our grid representation (Sec. 5.4.1). The score is summation of similarity against these segments and their associated language (scaled by α): $\psi_{pc}(p_i, l_j) = \sum_{k=1}^{n}(\text{sim}(p_i, p_k) + \alpha \ \text{sim}(l_i, l_k))$. If the associated language does not exist (*i.e.* p_k is not a manipulable part), it is given a set similarity value.

Similarly, the score ψ_{lang} is based on the k_l-most identical language instructions in the training data $\mathcal{D}$. It is a summation of similarity against identical language and associated expert segmentations of the point-cloud.

The feature score ψ_{feat} is computed by weighted features $w^T \phi_{feat}(p_i, l_j)$ as described in following section. Each score of the segmented parts is then adjusted by multiplying by ratio of its score against the marginalized score in the manual: $\hat{\psi}(p_i, l_j) = \frac{\psi(p_i, l_j)}{\sum_{l_k \in m_{new}} \psi(p_i, l_k)} \psi(p_i, l_j)$. For each $l_j \in m_{new}$, an optimal segment of the scene chosen as the segment with the maximum score: $max_{p_i \in s_{new}} \hat{\psi}(p_i, l_j)$.

A.2.3 Part Candidate Features

We compute 3 features for each segmentation in the context of the original scene. First, we infer where a person would stand by detecting the 'front' of the object based on plane segmentation, constrained to have a normal axis less than 45° from the line between the object's centroid and the original sensor location, thus assuming that the robot is introduced to the object within 45° from the 'front'.

We also compute a 'reach' distance from an imaginary person 1.7m tall. The reach distance is defined as the distance from the top of the person to each segment subtracted by the distance of the closest one.

Stitched point-clouds have significant noise near their edges. Thus, we compute the distance from the imaginary view ray, a line defined by the centroid of the scene to the head of the person. Lastly, some objects with many identical parts such as a soda fountain or a sauce dispenser, a description (e.g. 'Coke' or 'Sprite', 'ketchup' or 'mustard') might be difficult to disambiguate. Thus, for such cases, we also provided a 3D point as input, as if human is pointing at the label of the desired selection. Note that this does not point at the actual part but rather at its label or vicinity. A distance from this point is also used as a feature.

A.3 Details of Baseline Models

In this section, we explain each baseline model (Sec. 5.5.2) that we evaluate our full model against:

1) *Random Transfers (chance)*: Trajectories are selected at random from the set of trajectories in the training set.

2) *Object Part Classifier*: To test our hypothesis that classifying object parts as an intermediate step does not guarantee successful transfers, we trained an object part classifier using multiclass SVM [162] on point-cloud features $\phi(p)$ including local shape features [72], histogram of curvatures [131], and distribution of points. Using this classifier, we first classify the target object part p into an object part category (e.g. 'handle', 'knob'), then use the same feature space

to find its nearest neighbor p' of the same class from the training set. Then the trajectory τ' of p' is transferred to p.

3) *Structured support vector machine (SSVM)*: We used SSVM to learn a discriminant scoring function $\mathcal{F} : \mathcal{P} \times \mathcal{L} \times \mathcal{T} \rightarrow \mathbb{R}$. At test time, for target point-cloud/language pair (p, l), we output the trajectory τ from the training set that maximizes $\mathcal{F}$. To train SSVM, we use a joint feature mapping $\phi(p, l, \tau) = [\phi(\tau); \phi(p, \tau); \phi(l, \tau)]$. $\phi(\tau)$ applies Isomap [155] to interpolated τ for non-linear dimensionality reduction. $\phi(p, \tau)$ captures the overall shape when trajectory τ is overlaid over point-cloud p by jointly representing them in a voxel-based cube similar to Sec. 5.4.1, with each voxel holding count of occupancy by p or τ. Isomap is applied to this representation to get the final $\phi(p, \tau)$. Finally, $\phi(l, \tau)$ is the tensor product of the language features and trajectory features: $\phi(l, \tau) = \phi(l) \otimes \phi(\tau)$. We used our loss function (Appendix A.1) to train SSVM and used the cutting plane method to solve the SSVM optimization problem [62].

4) *Latent Structured SVM (LSSVM) + kinematic structure*: The way in which an object is manipulated largely depends on its internal structure – whether it has a 'revolute', 'prismatic', or 'fixed' joint. Instead of explicitly trying to learn this structure, we encoded this internal structure as latent variable $z \in \mathcal{Z}$, composed of joint type, center of joint, and axis of joint [145]. We used Latent SSVM [186] to train with z, learning the discriminant function $\mathcal{F} : \mathcal{P} \times \mathcal{L} \times \mathcal{T} \times \mathcal{Z} \rightarrow \mathbb{R}$. The model was trained with feature mapping $\phi(p, l, \tau, z) = [\phi(\tau); \phi(p, \tau); \phi(l, \tau); \phi(l, z); \phi(p, \tau, z)]$, which includes additional features that involve z. $\phi(l, z)$ captures the relation between l, a bag-of-words representation of language, and bag-of-joint-types encoded by z (vector of length 3 in-

dicating existence of each joint type) by computing the tensor product $\phi(l) \otimes \phi(z)$, then reshaping the product into a vector. $\phi(p, \tau, z)$ captures how well the portion of τ that actually interacts with p abides by the internal structure h. $\phi(p, \tau, z)$ is a concatenation of three types of features, one for each joint type. For 'revolute' type joints, it includes deviation of trajectory from plane of rotation defined by z, the maximum angular rotation while maintaining pre-defined proximity to the plane of rotation, and the average cosine similarity between rotation axis of τ and axis defined by z. For 'prismatic' joints, it includes the average cosine similarity between the extension axis and the displacement vector between waypoints. Finally, for 'fixed' joints, it includes whether the *uninteracting* part of τ has collision with the background p since it is important to approach the object from correct angle.

5) *Task-Similarity Transfers + random*: We compute the pairwise similarities between the test case (p_{test}, l_{test}) and each training example (p_{train}, l_{train}), then transfer a trajectory τ associated with the training example of highest similarity. Pairwise similarity is defined as a convex combination of the cosine similarity in bag-of-words representations of language and the average mutual point-wise distance of two point-clouds after a fixed number of iterations of the ICP [11] algorithm. If there are multiple trajectories associated with (p_{train}, l_{train}) of highest similarity, the trajectory for transfer is selected randomly.

6) *Task-similarity Transfers + weighting*: The previous method is problematic when non-expert demonstrations for a single task (p_{train}, l_{train}) vary in quality. Forbes et al. [35] introduces a score function for weighting demonstrations based on weighted distance to the "seed" (expert) demonstration. Adapting to our scenario of not having any expert demonstrations, we select τ that has the

lowest average distance from all other demonstrations for the same task, with each distance measured with our loss function (Appendix A.1.) This is similar to our noise handling approach in Sec. 5.2.4.

7) *Deep Network without Embedding*: We train a deep neural network to learn a similar scoring function $\mathcal{F} : \mathcal{P}\times\mathcal{L}\times\mathcal{T} \rightarrow \mathbb{R}$ to that learned for SSVM above. This model discriminatively projects the combination of point-cloud, language, and trajectory features to a score which represents how well the trajectory matches that point-cloud/language combination. Note that this is much less efficient than our joint embedding approach, as it must consider all combinations of a new point-cloud/language pair and every training trajectory to perform inference, as opposed to our model which need only project this new pair to our joint embedding space. This deep learning model concatenates all the input of three modalities and learns three hidden layers before the final layer.

8) *Deep Multimodal Network without Embedding*: The same approach as 'Deep Network without Embedding' with layers per each modality before concatenating as shown in Figure A.1. More details about the model can be found in [148].

9) *LMNN [174]-like cost function:* For all top layer fine-tuning and lower layer pre-training, we define the cost function without loss augmentation. Similar to LMNN [174], we give a finite margin between similarities. For example, as cost function for h^3:

$$\begin{aligned} L_{h^3}(p_i, l_i, \tau_i) = |1 + & sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau')) \\ & - sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_i))|_+ \end{aligned}$$

10) *Our Model without Pretraining:* Our full model finetuned without any pre-training of lower layers – all parameters are randomly initialized.

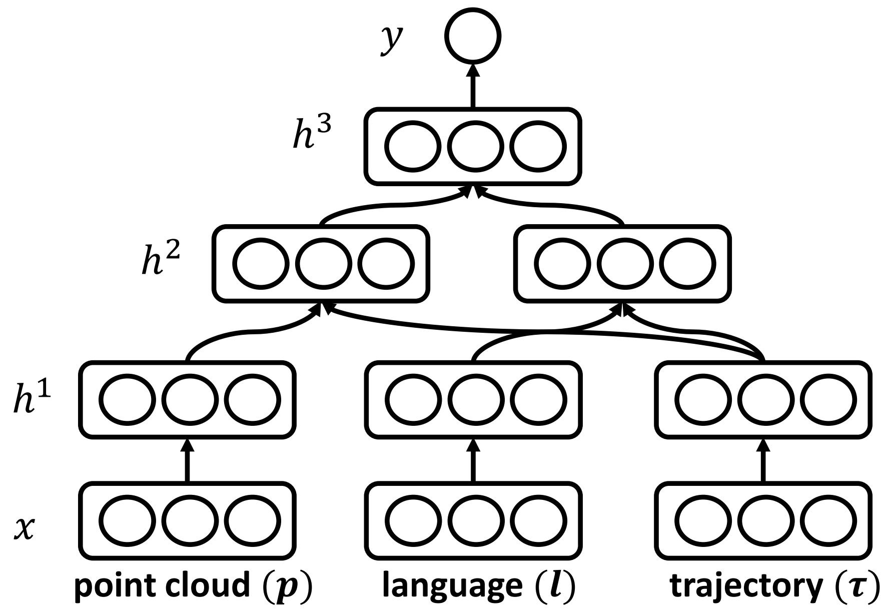

y
h^3
h^2
h^1
x
point cloud ($\boldsymbol{p}$)
language ($\boldsymbol{l}$)
trajectory ($\boldsymbol{\tau}$)

Figure A.1: **Deep Multimodal Network without Embedding**, a baseline model takes the input x of three different modalities (point-cloud, language, and trajectory) and outputs y, whether it is a good match or bad match. It first learns features separately (h^1) for each modality and then learns the relation (h^2) between input and output of the original structured problem. Finally, last hidden layer h^3 learns relations of all these modalities.

11) *Our Model with SDA:* Our full model without pre-training $h^{2,pl}$ and $h^{2,\tau}$ as defined in Sections 4.3.1 and 4.3.2. Instead, we pre-train each layer as stacked de-noising autoencoders [170, 188].

12) *Our Model without Noise Handling:* Our model is trained without noise handling as presented in Section 5.2.4. All of the trajectories collected from the crowd are trusted as a ground-truth labels.

13) *Our Model without Multiple Segmentations.* Our model trained only with expert segmentations, without taking utilizing all candidate segmentations in auto-encoders and multiple correct segmentations of the same part during train-

ing.

14) *Our Model with Experts:* Our model is trained only using trajectory demonstrations from an expert which were collected for evaluation purposes.

15) *Our Full Model - Deep Multimodal Embedding:* Our full model as described in this paper with network size of $h^{1,p}$, $h^{1,l}$, $h^{1,\tau}$, $h^{2,pl}$, $h^{2,\tau}$, and h^3 respectively having a layer with 150, 175, 100, 100, 75, and 50 nodes.

APPENDIX B

APPENDIX OF LEARNING TO REPRESENT HAPTIC FEEDBACK

B.1 Lowerbound Derivation

To continue our derivation of the lower bound on conditional log-likelihood from Sec. 6.2.4. The second term of equation 6.1:

$$\begin{aligned}
\mathbb{E}_{q_\phi(\vec{s}|\vec{o},\vec{r},\vec{a})}[log\ p_\theta(\vec{o},\vec{r}|\vec{s},\vec{a})] &= \mathbb{E}_{q_\phi(\vec{s}|\vec{o},\vec{r},\vec{a})}[log\ p_\theta(\vec{o}|\vec{s}) + log\ p_\theta(\vec{r}|\vec{s})] \\
&\approx \frac{1}{L}\sum_{l=1}^{L}[log\ p_\theta(\vec{o}|\vec{s}^{(l)}) + log\ p_\theta(\vec{r}|\vec{s}^{(l)})] \\
&= \frac{1}{L}\sum_{l=1}^{L}\sum_{t=1}^{T}[log\ p_\theta(o_t|s_t^{(l)}) + log\ p_\theta(r_t|s_t^{(l)})] \\
&\quad \text{where } \vec{s}^{(l)} = q_{\phi,\mu} + \epsilon^{(l)} q_{\phi,\Sigma} \text{ and } \epsilon^{(l)} \sim p(\epsilon)
\end{aligned}$$

Reparametrization trick ([70, 127]) at last step samples from the inferred distribution by a recognition network q_ϕ.

And, for the first term from equation 6.1:

$$
\begin{aligned}
D_{KL}(q_\phi(\vec{s}|\vec{o},\vec{r},\vec{a})||p_\theta(\vec{s}|\vec{a})) &= \int_{s_1}\cdots\int_{s_T} q_\phi(\vec{s}|\vec{o},\vec{r},\vec{a})\Big[log\frac{q_\phi(\vec{s}|\vec{o},\vec{r},\vec{a})}{p_\theta(\vec{s}|\vec{a})}\Big] \\
&= D_{KL}(q_\phi(s_1|\vec{o},\vec{r},\vec{a})||p(s_1)) \\
&\quad + \sum_{t=2}^{T}\mathbb{E}_{s_{t-1}\sim q_\phi(s_{t-1}|\vec{o},\vec{r},\vec{a})}[\\
&\qquad\qquad D_{KL}(q_\phi(s_t|s_{t-1},\vec{o},\vec{r},\vec{a})||p(s_t|s_{t-1},a_{t-1}))]
\end{aligned}
$$

using reparameterazation trick again,

$$
\begin{aligned}
&= D_{KL}(q_\phi(s_1|\vec{o},\vec{r},\vec{a})||p(s_1)) \\
&\quad + \sum_{t=2}^{T}\frac{1}{L}\sum_{l=1}^{L}[D_{KL}(q_\phi(s_t|s_{t-1},\vec{o},\vec{r},\vec{a})||p(s_t|s_{t-1}^{(l)},a_{t-1}))]
\end{aligned}
$$

where $s_{t-1}^{(l)} = q_{\phi,t-1,\mu} + \epsilon^{(l)}q_{\phi,t-1,\Sigma}$ and $\epsilon^{(l)} \sim p(\epsilon)$

Combining these two terms, we arrive at equation 6.2.

We do not explain each step of the derivation at length since similar ideas behind the derivation can be found at [75] although exact definition and formulation are different.

BIBLIOGRAPHY

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, 2004.

[2] Pieter Abbeel, Adam Coates, and Andrew Ng. Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*, 2010.

[3] David W Aha, Dennis Kibler, and Marc K Albert. Instance-based learning algorithms. *Machine learning*, 1991.

[4] Brandon Alexander, Kaijen Hsiao, Chad Jenkins, Bener Suay, and Russell Toris. Robot web tools [ros topics]. *Robotics & Automation Magazine, IEEE*, 19(4):20–23, 2012.

[5] Brenna Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *RAS*, 2009.

[6] Jennifer Barry, Kaijen Hsiao, Leslie Kaelbling, and Tomás Lozano-Pérez. Manipulation with multiple action types. In *ISER*, 2012.

[7] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Neural Information Processing Systems DLUFL Workshop, 2012.

[8] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mosenlechner, D. Pangercic, T. Ruhr, and M. Tenorth. Robotic roommates making pancakes. In *Humanoids*, 2011.

[9] Calin Belta, Antonio Bicchi, Magnus Egerstedt, Emilio Frazzoli, Eric Klavins, and George J Pappas. Symbolic planning and control of robot motion. *Robotics & Automation Magazine*, 2007.

[10] Stuart Bennett. A brief history of automatic control. *IEEE Control Systems Magazine*, 16(3):17–25, 1996.

[11] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.

[12] Matthew Blaschko and Christoph Lampert. Learning to localize objects with structured output regression. In *European Conference on Computer Vision*, 2008.

[13] Oren Boiman and Michal Irani. Detecting irregularities in images and in video. *IJCV*, 74(1):17–31, 2005.

[14] M. Bollini, J. Barry, and D. Rus. Bakebot: Baking cookies with the pr2. In *International Conference on Intelligent Robots and Systems PR2 Workshop*, 2011.

[15] M. Brand, N. Oliver, and A. Pentland. Coupled hidden makov models for complex action recognition. In *Computer Vision and Pattern Recognition*, 1997.

[16] H. Bui, D. Phung, and S. Venkatesh. Hierarchical hidden markov models with general state hierarchy. In *AAAI*, 2004.

[17] François Chaumette and Seth Hutchinson. Visual servo control. i. basic approaches. *Robotics & Automation Magazine, IEEE*, 13(4):82–90, 2006.

[18] Virginia Chu, Ian McMahon, Lorenzo Riano, Craig G McDonald, Qin He, Jorge Martinez Perez-Tejada, Michael Arrigo, Naomi Fitter, John C Nappo, Trevor Darrell, et al. Using robotic exploratory procedures to learn the meaning of haptic adjectives. In *International Conference on Robotics and Automation*, 2013.

[19] Gabriella Contardo, Ludovic Denoyer, Thierry Artieres, and Patrick Gallinari. Learning states representations in pomdp. In *ICLR*, 2014.

[20] Christopher Crick, Sarah Osentoski, Graylin Jay, and Odest Chadwicke Jenkins. Human and robot perception in large-scale learning from demonstration. In *Human-Robot Interaction*. ACM, 2011.

[21] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition*, 2005.

[22] Hao Dang and Peter K Allen. Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task. In *International Conference on Intelligent Robots and Systems*, 2012.

[23] C. Daniel, G. Neumann, and J. Peters. Learning concurrent motor skills in versatile solution spaces. In *International Conference on Intelligent Robots and Systems*. IEEE, 2012.

[24] Yiannis Demiris and Andrew Meltzoff. The robot in the crib: a developmental analysis of imitation skills in infants and robots. *Infant and Child Development*, 17(1):43–53, 2008.

[25] Renaud Detry, Carl Henrik Ek, Marianna Madry, and Danica Kragic. Learning a dictionary of prototypical grasp-predicting parts from grasping experience. In *International Conference on Robotics and Automation*, 2013.

[26] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, 2010.

[27] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *Int'l Wrksp Visual Surv Perf. Eval. Tracking Surv.*, 2005.

[28] Felix Endres, Jeff Trinkle, and Wolfram Burgard. Learning the dynamics of doors for robotic manipulation. In *International Conference on Intelligent Robots and Systems*, 2013.

[29] Goker Erdogan, Ilker Yildirim, and Robert A Jacobs. Transfer of object shape knowledge across visual and haptic modalities. In *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, 2014.

[30] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth. Describing objects by their attributes. In *Computer Vision and Pattern Recognition*, 2009.

[31] David Feil-Seifer and Maja J Matarié. Defining socially assistive robots. In *ICORR*, 2005.

[32] Vittorio Ferrari and Andrew Zisserman. Learning visual attributes. In *Neural Information Processing Systems*, 2007.

[33] Shai Fine, Yoram Singer, and Naftali Tishby. Parsing human motion with stretchable models. *Machine Learning*, 1998.

[34] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *International Conference on Robotics and Automation*, 2016.

[35] Maxwell Forbes, Michael Jae-Yoon Chung, Maya Cakmak, and Rajesh PN Rao. Robot programming by demonstration with crowdsourced action fixes. In *Second AAAI Conference on Human Computation and Crowdsourcing*, 2014.

[36] L. Frommberger. *Qualitative Spatial Abstraction in Reinforcement Learning*. Springer, 2010.

[37] Yang Gao, Lisa Anne Hendricks, Katherine J Kuchenbecker, and Trevor Darrell. Deep learning for tactile understanding from visual and haptic data. *arXiv preprint arXiv:1511.06065*, 2015.

[38] Mevlana C Gemici and Ankur Saxena. Learning haptic representation for manipulating deformable food objects. In *International Conference on Intelligent Robots and Systems*. IEEE, 2014.

[39] James Jerome Gibson. *The ecological approach to visual perception*. Psychology Press, 1986.

[40] Martin Giese and Tomaso Poggio. Neural mechanisms for the recognition of biological movement. *Nature Rev Neurosc.*, 4:179–192, 2003.

[41] Ross Girshick, Pedro Felzenszwalb, and David McAllester. Object detection with grammar models. In *Neural Information Processing Systems*, 2011.

[42] Ross Girshick, Forrest Iandola, Trevor Darrell, and Jitendra Malik. Deformable part models are convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[43] Stuart Glaser. Jt cartesian controller. `http://wiki.ros.org/robot_mechanism_controllers/JTCartesian%20Controller`. [Online; accessed 02.26.2016].

[44] Ian Goodfellow, Quoc Le, Andrew Saxe, Honglak Lee, and Andrew Y Ng. Measuring invariances in deep networks. In *Neural Information Processing Systems*, 2009.

[45] Cordell Green. Application of theorem proving to problem solving. Technical report, DTIC Document, 1969.

[46] Abhinav Gupta, Praveen Srinivasan, Jianbo Shi, and Larry S. Davis. Understanding videos, constructing plots learning a visually grounded storyline model from annotated videos. In *Computer Vision and Pattern Recognition*, 2009.

[47] Raia Hadsell, Ayse Erkan, Pierre Sermanet, Marco Scoffier, Urs Muller, and Yann LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *International Conference on Intelligent Robots and Systems*, 2008.

[48] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[49] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

[50] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[51] K. Hsiao, S. Chitta, M. Ciocarlie, and E. Jones. Contact-reactive grasping of objects with partial shape information. In *International Conference on Intelligent Robots and Systems*, 2010.

[52] Kaijen Hsiao, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Grasping pomdps. In *International Conference on Robotics and Automation*, 2007.

[53] Junlin Hu, Jiwen Lu, and Yap-Peng Tan. Discriminative deep metric learning for face verification in the wild. In *Computer Vision and Pattern Recognition*, 2014.

[54] Ninghang Hu, Zhongyu Lou, Gwenn Englebienne, and Ben Krse. Learning to recognize human activities from soft labeled data. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[55] Yani Ioannou, Babak Taati, Robin Harrap, et al. Difference of normals as a multi-scale operator in unorganized point clouds. In *3DIMPVT*, 2012.

[56] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *ACM Symposium on UIST*, 2011.

[57] Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *Neural Information Processing Systems*, 2013.

[58] Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning preferences for manipulation tasks from online coactive feedback. In *International Journal of Robotics Research*, 2015.

[59] H. Jhuang, T. Serre, L. Wolf, and T. Poggio. A biologically inspired system for action recognition. In *International Conference on Computer Vision*, 2007.

[60] Yun Jiang, Marcus Lim, Changxi Zheng, and Ashutosh Saxena. Learning to place new objects in a scene. *International Journal of Robotics Research*, 2012.

[61] Yun Jiang, Hema Koppula, and Ashutosh Saxena. Hallucinated humans as the hidden context for labeling 3d scenes. In *Computer Vision and Pattern Recognition*, 2013.

[62] T. Joachims, T. Finley, and C-N. J. Yu. Cutting-plane training of structural svms. *Machine Learning*, 2009.

[63] Thorsten Joachims. Training linear svms in linear time. In *KDD*, 2006.

[64] B. Johnson and H. Kress-Gazit. Probabilistic analysis of correctness of high-level robot behavior with sensor error. In *Robotics: Science and Systems*, 2011.

[65] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *International Conference on Robotics and Automation*, 2011.

[66] Dov Katz, Moslem Kazemi, J Andrew Bagnell, and Anthony Stentz. Interactive segmentation, tracking, and kinematic modeling of unknown 3d articulated objects. In *International Conference on Robotics and Automation*, pages 5003–5010. IEEE, 2013.

[67] Henry Kautz and Bart Selman. Planning as satisfiability. In *European conference on Artificial intelligence*, 1992.

[68] Ben Kehoe, Akihiro Matsukawa, Sal Candido, James Kuffner, and Ken Goldberg. Cloud-based robot grasping with the google object recognition engine. In *International Conference on Robotics and Automation*, 2013.

[69] Charles Kemp, Noah D Goodman, and Joshua B Tenenbaum. Learning to learn causal models. *Cognitive Science*, 34(7), 2010.

[70] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[71] Sven Koenig. Agent-centered search. *AI Magazine*, 2001.

[72] H. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic labeling of 3d point clouds for indoor scenes. *Neural Information Processing Systems*, 2011.

[73] Hema Koppula and Ashutosh Saxena. Anticipating human activities using object affordances for reactive robotic response. In *Robotics: Science and Systems*, 2013.

[74] H.S. Koppula, A. Anand, T. Joachims, and Ashutosh Saxena. Semantic labeling of 3d point clouds for indoor scenes. In *Neural Information Processing Systems*, 2011.

[75] Rahul G Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

[76] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems*, 2012.

[77] O. Kroemer, E. Ugur, E. Oztop, and J. Peters. A kernel-based approach to direct action perception. In *International Conference on Robotics and Automation*, 2012.

[78] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, 2008.

[79] K. Lai, L. Bo, X. Ren, and D. Fox. Detection-based object labeling in 3d scenes. In *International Conference on Robotics and Automation*, 2012.

[80] K. Lai, L. Bo, and D. Fox. Unsupervised feature learning for 3d scene labeling. In *International Conference on Robotics and Automation*, 2014.

[81] C. H. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *Computer Vision and Pattern Recognition*, 2009.

[82] Tian Lan, Yang Wang, Weilong Yang, and Greg Mori. Beyond actions: Discriminative models for contextual group activities. In *Neural Information Processing Systems*, 2010.

[83] Ivan Laptev. On space-time interest points. *IJCV*, 64(2):107–123, 2005.

[84] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. *Robotics: Science and Systems*, 2013.

[85] Ian Lenz, Ross Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, 2015.

[86] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

[87] Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. *International Conference on Robotics and Automation*, 2015.

[88] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 2016.

[89] Congcong Li, TP Wong, Norris Xu, and Ashutosh Saxena. Feccm for scene understanding: Helping the robot to learn multiple tasks. In *Video contribution in International Conference on Robotics and Automation*, 2011.

[90] Li-Jia Li, Richard Socher, and Li Fei-Fei. Towards total scene understanding: Classification, annotation and segmentation in an automatic framework. In *Computer Vision and Pattern Recognition*, 2009.

[91] Qiang Li, Carsten Schürmann, Robert Haschke, and Helge J Ritter. A control framework for tactile servoing. In *Robotics: Science and Systems*, 2013.

[92] Zhe Li, Sven Wachsmuth, Jannik Fritsch, and Gerhard Sagerer. *Vision Systems: Segmentation and Pattern Recognition*, chapter 8, pages 131–148. InTech, 2007.

[93] Lin Liao, Dieter Fox, and Henry Kautz. Extracting places and activities from gps traces using hierarchical conditional random fields. *International Journal of Robotics Research*, 26(1):119–134, 2007.

[94] Jingen Liu, Saad Ali, and Mubarak Shah. Recognizing human actions using multiple features. In *Computer Vision and Pattern Recognition*, 2008.

[95] M. Lopes, F. S. Melo, and L. Montesano. Affordance-based imitation learning in robots. In *International Conference on Intelligent Robots and Systems*, 2007.

[96] O. Mangin, Pierre-Yves Oudeyer, et al. Unsupervised learning of simultaneous motor primitives through imitation. In *IEEE International Conference on Development and Learning and on Epigenetic Robotics*, 2011.

[97] Francisco Martinez-Contreras, Carlos Orrite-Urunuela, Elias Herrero-Jaraba, Hossein Ragheb, and Sergio A. Velastin. Recognizing human actions using silhouette-based hmm. In *Advanced Video and Signal Based Surveillance*, pages 43–48, 2009.

[98] Andrew Mccallum, Dayne Freitag, and Fernando Pereira. Maximum entropy markov models for information extraction and segmentation. In *International Conference on Machine Learning*, 2000.

[99] Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *CoRR*, 2013.

[100] S. Miller, J. Van Den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel. A geometric approach to robotic laundry folding. *International Journal of Robotics Research*, 2012.

[101] D. Misra, J. Sung, K. Lee, and A. Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *Robotics: Science and Systems*, 2014.

[102] Dipendra Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *Robotics: Science and Systems*, 2014.

[103] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[104] J. Moore, Shuo Chen, T. Joachims, and D. Turnbull. Learning to embed songs and tags for playlist prediction. In *Conference of the International Society for Music Information Retrieval (ISMIR)*, pages 349–354, 2012.

[105] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.

[106] T. Mukai, S. Hirano, H. Nakashima, Y. Kato, Y. Sakaida, S. Guo, and S. Hosoe. Development of a nursing-care assistant robot riba that can lift a human in its arms. In *International Conference on Intelligent Robots and Systems*, 2010.

[107] K. Mülling, J. Kober, O. Kroemer, and J. Peters. Learning to select and generalize striking movements in robot table tennis. *International Journal of Robotics Research*, 32(3):263–279, 2013.

[108] Jim Mutch and David Lowe. Multiclass object recognition using sparse, localized features. In *Computer Vision and Pattern Recognition*, 2006.

[109] H. Nakai, M. Yamataka, T. Kuga, S. Kuge, H. Tadano, H. Nakanishi, M. Furukawa, and H. Ohtsuka. Development of dual-arm robot with multi-fingered hands. In *RO-MAN*, 2006.

[110] Andrew Y Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000.

[111] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *International Conference on Machine Learning*, 2011.

[112] H. Nguyen, M. Ciocarlie, J. Hsiao, and C. C. Kemp. Ros commander (rosco): Behavior creation for home robots. In *International Conference on Robotics and Automation*, 2013.

[113] Hai Nguyen, Cressel Anderson, Alexandor Trevor, Advait Jain, Zhe Xu, and Charles C. Kemp. El-e: An assistive robots and fetches objects from flat surfaces. In *Human-Robot Interaction*, 2008.

[114] Juan Niebles, Chih-Wei Chen, and Li Fei-Fei. Modeling temporal structure of decomposable motion segments for activity classification. In *European Conference on Computer Vision*, 2010.

[115] Huazhong Ning, T. X. Han, D. B. Walther, Ming Liu, and T. S. Huang. Hierarchical space-time model enabling efficient search for human actions. *IEEE Trans Circuits Sys. Video Tech.*, 19(6), 2009.

[116] Donald A Norman. *The design of everyday things*. Basic books, 1988.

[117] Jaeheung Park and Oussama Khatib. A haptic teleoperation approach based on contact force control. *International Journal of Robotics Research*, 25 (5-6):575–591, 2006.

[118] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *International Conference on Robotics and Automation*, 2009.

[119] Peter Pastor, Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, and Stefan Schaal. Skill learning and task outcome prediction for manipulation. In *International Conference on Robotics and Automation*, 2011.

[120] Mike Phillips, Victor Hwang, Sachin Chitta, and Maxim Likhachev.

Learning to plan for constrained manipulation from demonstrations. In *Robotics: Science and Systems*, 2013.

[121] Sudeep Pillai, Matthew Walter, and Seth Teller. Learning articulated motions from visual demonstration. In *Robotics: Science and Systems*, 2014.

[122] PrimeSense. Nite middleware. http://www.primesense.com/, 2011.

[123] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *International Conference on Robotics and Automation workshop on open source software*, 2009.

[124] Marc'Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition*, 2007.

[125] N. Ratliff, J. A. Bagnell, and M. Zinkevich. Maximum margin planning. In *International Conference on Machine Learning*, 2006.

[126] Nathan Ratliff, David Silver, and J Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 2009.

[127] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[128] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 2012.

[129] Mikel D. Rodriguez, Javed Ahmed, and Mubarak Shah. Action mach: A spatio-temporal maximum average correlaton height filter for action recognition. In *Computer Vision and Pattern Recognition*, 2008.

[130] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach. Vol. 2*. Prentice Hall, 2010.

[131] R. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *International Conference on Robotics and Automation*, 2011.

[132] Radu Bogdan Rusu. Semantic 3d object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, 2010.

[133] J Kenneth Salisbury. Active stiffness control of a manipulator in cartesian coordinates. In *Decision and Control including the Symposium on Adaptive Processes, 1980 19th IEEE Conference on*, 1980.

[134] Brian Sallans. Learning factored representations for partially observable markov decision processes. In *Neural Information Processing Systems*, pages 1050–1056. Citeseer, 1999.

[135] Benjamin Sapp, David Weiss, and Ben Taskar. Parsing human motion with stretchable models. In *Computer Vision and Pattern Recognition*, 2011.

[136] Ashutosh Saxena, Justin Driemeyer, and Andrew Ng. Learning 3-d object orientation from images. In *International Conference on Robotics and Automation*, 2009.

[137] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by the visual cortex. In *Computer Vision and Pattern Recognition*, 2005.

[138] Ken Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH Computer Graphics*, 19(3):245–254, 1985.

[139] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[140] C. Sminchisescu, A. Kanaujia, Zhiguo Li, and D. Metaxas. Conditional models for contextual human motion recognition. In *International Conference on Computer Vision*, 2005.

[141] R. Socher, B. Huval, B. Bhat, C. Manning, and A. Ng. Convolutional-recursive deep learning for 3d object classification. In *Neural Information Processing Systems*, 2012.

[142] Richard Socher, Jeffrey Pennington, Eric Huang, Andrew Ng, and Christopher Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *EMNLP*, 2011.

[143] Kihyuk Sohn, Wenling Shang, and Honglak Lee. Improved multimodal deep learning with variation of information. In *Neural Information Processing Systems*, 2014.

[144] Nitish Srivastava and Ruslan R Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Neural Information Processing Systems*, 2012.

[145] J. Sturm, C. Stachniss, and W. Burgard. A probabilistic framework for learning kinematic models of articulated objects. *JAIR*, 41(2):477–526, 2011.

[146] Jaeyong Sung, Colin Ponce, Bart Selman, and Ashutosh Saxena. Unstructured human activity detection from rgbd images. In *International Conference on Robotics and Automation*, 2012.

[147] Jaeyong Sung, Bart Selman, and Ashutosh Saxena. Synthesizing manipulation sequences for under-specified tasks using unrolled markov random fields. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[148] 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*, 2015.

[149] Jaeyong Sung, Seok Hyun Jin, Ian Lenz, and Ashutosh Saxena. Robobarista: Learning to manipulate novel objectsvia deep multimodal embedding. 2016.

[150] Jaeyong Sung, Ian Lenz, and Ashutosh Saxena. Deep multimodal embedding: Manipulating novel objects with point-clouds, language and trajectories. In *International Conference on Robotics and Automation (ICRA)*, 2017.

[151] Jaeyong Sung, J. Kenneth Salisbury, and Ashutosh Saxena. Learning to represent haptic feedback for partially-observable tasks. In *International Conference on Robotics and Automation (ICRA)*, 2017.

[152] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[153] Adriana Tapus, Cristian Ţăpuş, and Maja J. Matarié. User-robot personality matching and assistive robot behavior adaptation for post-stroke rehabilitation therapy. *Intel. Ser. Robotics*, 1(2):169–183, 2008.

[154] Stefanie Tellex, Ross Knepper, Adrian Li, Thomas Howard, Daniela Rus, and Nicholas Roy. Asking for help using inverse semantics. *Robotics: Science and Systems*, 2014.

[155] J. Tenenbaum, V. De Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[156] T. Theodoridis, A. Agapitos, Huosheng Hu, and S. M. Lucas. Ubiquitous robotics in physical human action recognition: A comparison between dynamic anns and gp. In *International Conference on Robotics and Automation*, 2008.

[157] Sebastian Thrun, Wolfram Burgard, Dieter Fox, et al. *Probabilistic robotics*. MIT press Cambridge, 2005.

[158] R Toris and S Chernova. Robotsfor. me and robots for you. In *Proceedings of the Interactive Machine Learning Workshop, Intelligent User Interfaces Conference*, pages 10–12, 2013.

[159] Russell Toris, David Kent, and Sonia Chernova. The robot management system: A framework for conducting human-robot interaction studies through crowdsourcing. *Journal of Human-Robot Interaction*, 3(2):25–49, 2014.

[160] Sebastian Trimpe, Alexander Millane, Simon Doessegger, and Raffaello DAndrea. A self-tuning lqr approach demonstrated on an inverted pendulum. In *IFAC World Congress*, page 11, 2014.

[161] Tran The Truyen, Dinh Q. Phung, Hung H. Bui, and Svetha Venkatesh. Hi-

erarchical semi-markov conditional random fields for recursive sequential data. In *Neural Information Processing Systems*, 2008.

[162] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *International Conference on Machine Learning*. ACM, 2004.

[163] I. Tsochantaridis, T. Joachims, T. Hofmann, Y. Altun, and Y. Singer. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(9), 2005.

[164] Abhinav Valada, Luciano Spinello, and Wolfram Burgard. Deep feature learning for acoustics-based terrain classification. In *International Symposium on Robotics Research*, 2015.

[165] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[166] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[167] Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 2009.

[168] Ngo Anh Vien and Marc Toussaint. Touch based pomdp manipulation via sequential submodular optimization. In *Humanoids*, 2015.

[169] Francisco Vina, Yasemin Bekiroglu, Christian Smith, Yiannis Karayiannidis, and Danica Kragic. Predicting slippage and learning manipulation affordances through gaussian process regression. In *Humanoids*, 2013.

[170] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*, 2008.

[171] Niklas Wahlström, Thomas B Schön, and Marc Peter Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*, 2015.

[172] Xiaogang Wang, Xiaoxu Ma, and W.E.L. Grimson. Unsupervised activity perception in crowded and complicated scenes using hierarchical bayesian models. *Pattern Analysis and Machine Intelligence*, 2009.

[173] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Neural Information Processing Systems*, 2015.

[174] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. In *Neural Information Processing Systems*, 2005.

[175] Daniel Weinland, Edmond Boyer, and Remi Ronfard. Action recognition from arbitrary views using 3d exemplars. In *International Conference on Computer Vision*, 2007.

[176] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *International Joint Conference on Artificial Intelligence*, 2011.

[177] Steven Wieland, D Gonzalez-Aguirre, Nikolaus Vahrenkamp, Tamim Asfour, and Rüdiger Dillmann. Combining force and visual feedback for physical interaction tasks in humanoid robots. In *Humanoid Robots*, 2009.

[178] Andrew D. Wilson and Aaron F. Bobick. Parametric hidden markov models for gesture recognition. *Pattern Analysis and Machine Intelligence*, 1999.

[179] Shu-Fai Wong, Tae-Kyun Kim, and Roberto Cipolla. Learning motion categories using both semantic and structural information. In *Computer Vision and Pattern Recognition*, 2007.

[180] Chenxia Wu, Ian Lenz, and Ashutosh Saxena. Hierarchical semantic labeling for task-relevant rgb-d perception. In *Robotics: Science and Systems*, 2014.

[181] Chenxia Wu, Ian Lenz, and Ashutosh Saxena. Hierarchical semantic labeling for task-relevant rgb-d perception. In *Robotics: Science and Systems*, 2014.

[182] Jianxin Wu, Adebola Osuntogun, Tanzeem Choudhury, Matthai Philipose, and James M. Rehg. A scalable approach to activity recognition based on object use. In *International Conference on Computer Vision*, 2007.

[183] Q. Yang, K. Wu, and Y. Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171, 2007.

[184] Weilong Yang, Yang Wang, and Greg Mori. Recognizing human actions from still images with latent poses. In *Computer Vision and Pattern Recognition*, 2010.

[185] Bangpeng Yao and Li Fei-Fei. Modeling mutual context of object and human pose in human-object interaction activities. In *Computer Vision and Pattern Recognition*, 2010.

[186] C.-N. Yu and T. Joachims. Learning structural svms with latent variables. In *International Conference on Machine Learning*, 2009.

[187] Matthew D Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[188] Matthew D Zeiler, M Ranzato, Rajat Monga, et al. On rectified linear units for speech processing. In *International Conference on Acoustics, Speech and Signal Processing*, 2013.

[189] Richard Zhang, Stefan A Candra, Kai Vetter, and Avideh Zakhor. Sensor fusion for semantic segmentation of urban scenes. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.

[190] Hankz Hankui Zhuo, Qiang Yang, Derek Hao Hu, and Lei Li. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 2010.