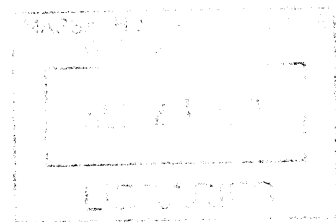


# Learning and Recognition of Hybrid Manipulation Tasks in Variable Environments using Probabilistic Flow Tubes

ARCHIVES



by

Shuonan Dong

Submitted to the Department of Aeronautics and Astronautics  
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012

© Massachusetts Institute of Technology 2012. All rights reserved.

Author .....  .....

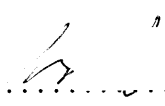
Department of Aeronautics and Astronautics

August 23, 2012

Certified by...  .....


Prof. Brian C. Williams

Thesis Supervisor

Certified by...  .....

Dr. Andreas Hoffmann

Thesis Committee

Certified by...  .....

Prof. Patrick Winston

Thesis Committee

Accepted by...  .....

Prof. Eytan Modiano

Chair, Committee on Graduate Students



# Learning and Recognition of Hybrid Manipulation Tasks in Variable Environments using Probabilistic Flow Tubes

by

Shuonan Dong

Submitted to the Department of Aeronautics and Astronautics  
on August 23, 2012, in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy

## Abstract

Robots can act as proxies for human operators in environments where a human operator is not present or cannot directly perform a task, such as in dangerous or remote situations. Teleoperation is a common interface for controlling robots that are designed to be human proxies. Unfortunately, teleoperation may fail to preserve the natural fluidity of human motions due to interface limitations such as communication delays, non-immersive sensing, and controller uncertainty.

I envision a robot that can learn a set of motions that a teleoperator commonly performs, so that it can autonomously execute routine tasks or recognize a user's motion in real time. Tasks can be either primitive activities or compound plans. During online operation, the robot can recognize a user's teleoperated motions on the fly and offer real-time assistance, for example, by autonomously executing the remainder of the task.

I realize this vision by addressing three main problems: (1) learning primitive activities by identifying significant features of the example motions and generalizing the behaviors from user demonstration trajectories; (2) recognizing activities in real time by determining the likelihood that a user is currently executing one of several learned activities; and (3) learning complex plans by generalizing a sequence of activities, through auto-segmentation and incremental learning of previously unknown activities.

To solve these problems, I first present an approach to learning activities from human demonstration that (1) provides flexibility and robustness when encoding a user's demonstrated motions by using a novel representation called a probabilistic flow tube, and (2) automatically determines the relevant features of a motion so that they can be preserved during autonomous execution in new situations. I next introduce an approach to real-time motion recognition that (1) uses temporal information to successfully model motions that may be non-Markovian, (2) provides fast real-time recognition of motions in progress by using an incremental temporal alignment approach, and (3) leverages the probabilistic flow tube representation to ensure robustness during recognition against varying environment states. Finally, I

develop an approach to learn combinations of activities that (1) automatically determines where activities should be segmented in a sequence and (2) learns previously unknown activities on the fly.

I demonstrate the results of autonomously executing motions learned by my approach on two different robotic platforms supporting user-teleoperated manipulation tasks in a variety of environments. I also present the results of real-time recognition in different scenarios, including a robotic hardware platform. Systematic testing in a two-dimensional environment shows up to a 27% improvement in activity recognition rates over prior art, while maintaining average computing times for incremental recognition of less than half of human reaction time.

Thesis Supervisor: Prof. Brian C. Williams

## Acknowledgments

First, I must thank my partner in life, Thomas Coffee, for the long hours talking through the major stumbling blocks in my research with me, reviewing and proof-reading my papers, and cooking for me so I can have more time to work.

This thesis would not have been possible without the support, insights, and encouragement of my research advisor, Brian Williams, and my committee members, Andreas Hofmann and Patrick Winston. Their patience and understanding throughout the thesis development and writing process has reduced a most stressful period into something approachable and even enjoyable.

To all my friends in the MERS lab, past and present, I say “thank you” for being such an important part of my life for the past number of years. In no particular order, *thank you*, Julie Shah, for all your advice and friendship inside the lab and out; David Wang for all those memories since freshman year; Hui Li, for introducing me to the wonders of Seattle; Larry Bush, for providing entertainment in the most unexpected ways; Lars Blackmore, for being such an inspirational go-getter; Bobby Effinger, for showing me what lies ahead; Hiro Ono, for being the role model of hard work; Cristi Wilcox, for reminding us that there is life outside of lab; Patrick Conrad, for all those long hours on the WAM and ATHLETE; Peng Yu, for his humble brilliance; Eric Timmons, for bringing social life to our group; Steve Levine, for making the WAM work like magic; Andrew Wang, for his leadership and initiatives; Pedro Santana, for his Brazilian sense of humor; and Simon Fang, for keeping the lab fun.

Special thanks also go to David Mittman, Sarah Osentoski, and Shuo Wang for their help in operating the different robots used in the demonstrations and experiments presented in this thesis. Special thanks also to Sonia Chernova for organizing an excellent Learning from Demonstration Challenge at AAAI 2011.

This research was in part funded by the National Science Foundation graduate fellowship and the National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a. Additional support was provided by a NASA JPL Strategic University Research Partnership.

*In memory of Darrell Cain, a space visionary.*

*August 25, 1985 - July 31, 2012*

# Contents

<b>1</b>	<b>Introduction</b>	<b>19</b>
1.1	Motivation . . . . .	19
1.2	Problem Description . . . . .	24
1.3	Approach Overview and Innovations . . . . .	26
1.3.1	Activity Learning . . . . .	26
1.3.2	Real-time Recognition . . . . .	28
1.3.3	Plan Learning . . . . .	29
1.4	Thesis Layout . . . . .	31
<b>2</b>	<b>Related Work</b>	<b>33</b>
2.1	Learning from Demonstration . . . . .	33
2.2	Motion Recognition . . . . .	40
2.3	Flow Tube Background from Plan Execution . . . . .	40
2.4	Adaptive Interfaces . . . . .	41
<b>3</b>	<b>Problem Statement</b>	<b>43</b>
3.1	Definitions of Inputs and Outputs . . . . .	45
3.2	Definition of an Activity Learning Problem . . . . .	48
3.3	Definition of an Activity Recognition Problem . . . . .	49
3.4	Definition of a Plan Learning Problem . . . . .	50
3.5	Relationship to Prior Art . . . . .	50
<b>4</b>	<b>Learning Single Activities</b>	<b>53</b>

4.1	Identifying Motion Variables . . . . .	55
4.2	Data Processing and Flow Tube Generation . . . . .	60
4.3	Pre-learning . . . . .	66
4.4	Enabling Autonomous Execution . . . . .	69
<b>5</b>	<b>Recognizing Motions Online</b>	<b>71</b>
5.1	Partial Flow Tube Matching . . . . .	73
5.2	Temporal Alignment of Partial Motion . . . . .	75
5.3	Compute Log Likelihoods . . . . .	76
<b>6</b>	<b>Learning High-level Plans</b>	<b>79</b>
6.1	Determining Activity Sequence . . . . .	83
6.2	Learning Unknown Activities . . . . .	87
6.3	Auto-segmentation . . . . .	89
6.4	Validate auto-segmentation . . . . .	98
6.5	Generating Plan-level Probabilistic Flow Tube . . . . .	99
<b>7</b>	<b>Experimental Results</b>	<b>103</b>
7.1	Validation of Activity Learning and Recognition . . . . .	103
7.1.1	Two-dimensional Variable Environment . . . . .	103
7.1.2	Two-dimensional Static Environment . . . . .	108
7.1.3	Hardware Validation of Real-time Recognition . . . . .	109
7.1.4	Hardware Demonstration of Autonomous Execution . . . . .	111
7.2	Validation of Plan Learning . . . . .	117
7.2.1	Two-dimensional Environment Tests . . . . .	117
7.2.2	Hardware Demonstration of Plan Learning . . . . .	123
7.3	Results Summary . . . . .	131
<b>8</b>	<b>Conclusions</b>	<b>133</b>
8.1	Future Extensions . . . . .	133
8.1.1	Compliant Execution . . . . .	133
8.1.2	Obstacle Avoidance . . . . .	133



8.1.3	Plan Recognition . . . . .	134
8.2	Conclusion . . . . .	136



# List of Figures

1-1	Docked Tri-ATHLETE robots transporting a habitat off of a lunar lander mock-up. The initial and goal states are shown in the first and last images, respectively. . . . .	20
1-2	Knowing only the previous state at the highlighted position will not distinguish the two motions. . . . .	22
1-3	Example visual representation of a probabilistic flow tube . . . . .	23
1-4	Architecture of learning and recognition problems . . . . .	25
1-5	Probabilistic flow tubes can look drastically different for different environment states . . . . .	27
1-6	Given a user's current execution (black arrow), the algorithm first determines correspondence points (pink dots) in each candidate learned motion . . . . .	28
1-7	Illustration of auto-segmentation . . . . .	30
2-1	Mixtures of Gaussians may not be an intuitive representation of continuous motions. . . . .	34

3-1	Illustrated example of a learning and recognition problem. Initially, a user demonstrates a set of motions a few times (1). The learning problem consists of generalizing the demonstrations into probabilistic flow tube representations for each of the three motions in the new environment state (2). Given a new user motion depicted as the shift in the red box position in the direction of the arrow, the recognition problem consists of determining which motion the user is most likely performing (3). . . . .	44
4-1	Example illustrations of five possible ways that motion variables can be relevant. Arrows refer to the robot end effector trajectories. . . . .	56
4-2	Four training examples of $\mathbf{P}_x$ position data at time steps when contact changes. Notice that across different training samples, the effector (blue) and box (red) positions match at the start of contact, and the effector (blue) and bin (green) positions match at the end of contact. These are the relevant features of the “move box to bin” motion. . . . .	57
4-3	Clustering identifies motion variables for a 2-D “move object to bin” motion. Left: robot effector and bin locations at the start and end of each demonstration. Middle: values of $(\mathbf{P}^{bin}(0) - \mathbf{P}^{eff}(0))$ to determine if <i>relEffStart</i> is a relevant motion variable for position—the large spread indicates low relevance. Right: values of $(\mathbf{P}^{bin}(N) - \mathbf{P}^{eff}(N))$ to determine if <i>relEffEnd</i> is a relevant motion variable for position—the narrow spread indicates that this variable is relevant to this motion. . . . .	58
4-4	Illustrated steps of the approach with three demonstrations of the “move the box to the bin” task in the two-dimensional simulation environment . . . . .	62
4-5	Illustration of how normalization works for position and orientation variables. . . . .	64

4-6	For two sets of environment states $p$ (“prelearned”) and $c$ (“current”) that correspond to the same set of five training sequences shown in blue, generating the PFT for $c$ directly from training sequences (left) is equivalent to generating the PFT for $c$ from normalizing the PFT for $p$ (right). . . . .	68
5-1	An example partial test motion (black) is compared to each learned PFT (blue) in a 2D environment initially with a red box, green bin, and stationary locations $x$ and $o$ as shown. The magenta marking on each PFT indicates the spot on the PFT that best matches the current execution (rightmost end of black motion) as determined by lines 2 to 6 in Algorithm 5.1. . . . .	74
5-2	During incremental dynamic time warping, the algorithm only needs to update the cost and back pointer matrices from the previously stored values (depicted by the inner box) to obtain a new temporal matching (red and purple path). . . . .	76
6-1	Summary of plan learning approach . . . . .	82
6-2	Illustration of performing online recognition on task sequences. In the two keyframe trials $\mathcal{Y}_1$ and $\mathcal{Y}_2$ , the log likelihood of activity 2 is highest during the first segment, activity 3 is highest during the second segment, and activity 1 is highest during the third segment. Therefore, it is likely that the activity sequence in this plan is $\{2, 3, 1\}$ . . . . .	85
6-3	Example process of determining the sequence of activities that compose the demonstrated plan from keyframe trials. Shading represents recognized log likelihood, and red values represent the percentage of time steps during which a particular activity was recognized in a keyframe segment. The keyframe trials “vote” on an average recognition frequency for each segment and agree on a recognized activity sequence for the plan. . . . .	86

6-4	Illustration of generating candidate keyframes on a non-keyframe trial using time proportions in a keyframe trial, and updating candidate keyframes based on motion variables observed in keyframe trial. . . .	91
6-5	Illustration of how a keyframe is selected based on motion variable geometry . . . . .	93
6-6	Generating the PFT for the entire plan involves concatenating the PFTs for each activity in the sequence initialized at evolving environment states . . . . .	101
7-1	Example learned PFTs in randomly generated initial environment states.	104
7-2	Example learned flow tubes of different activities overlaid in three different initial environment states. Blue PFTs represent “move box to bin,” red PFTs represent “move box left 1 unit,” and green PFTs represent “move box to $x$ .” . . . . .	105
7-3	Compare learned PFT and GMM models (blue) against user generated trajectories (red) in different initial environment states. Blue ellipses represent the range of 1 standard deviation. . . . .	106
7-4	Example log likelihoods over time for the same test cases using the PFT approach (top row) and the HMM approach (bottom row) . . .	107
7-5	Example output of an “encircle bin clockwise with box” motion. Left: the PFT model. Right: the GMM model. . . . .	108
7-6	WAM robot setups for the five motions: “move ball to bin”(A), “wind cable” (B), “unwind cable” (B), “anchor rope left then right” (C), and “anchor rope right then left” (C) . . . . .	110
7-7	WAM end effector trajectories . . . . .	111
7-8	ATHLETE move box to platform task . . . . .	112
7-9	Five teleoperated demonstrations of the “ATHLETE move box to platform” task. Autonomous execution of the task is shown in thick green.	112
7-10	Autonomous execution of ATHLETE move box to platform task . . .	113
7-11	PR2 and potential task setup . . . . .	114

7-12	Results of learning from 5 demonstrations of each motion. . . . .	115
7-13	Result of auto segmentation on non-keyframe trials (boxed) of “box to bin, ball to center” plan, from 3 user provided keyframe trials using motion variable inference approach. Compare with Figure 7-14. . . .	118
7-14	Result of auto segmentation on non-keyframe trials (boxed) of “box to bin, ball to center” plan, from 3 user provided keyframe trials using recognition optimization approach. Compare with Figure 7-13. . . . .	119
7-15	Example generated plan PFTs for “box to bin and ball to center” plan from different initial environment states. . . . .	122
7-16	Hardware environment setup . . . . .	124
7-17	Simulation environment . . . . .	125
7-18	Auto-generated PFT trajectories for the “red on green, pink on red” (“rgpr” for short) task plan for various new environment states . . . .	127
7-19	Auto-generated trajectory and PFT for the “red on green, pink on red” task plan for new environment state trial 2. . . . .	128
7-20	Auto-generated trajectory and PFT for the “red on green, pink on red” task plan for new environment state trial 6. . . . .	129
7-21	Auto-generated trajectory and PFT for the “red on green, pink on red” task plan for new environment state trial 9. . . . .	130
8-1	Obstacle avoidance may be achieved by overlaying a potential field that pushes away from obstacles in the environment . . . . .	134
8-2	To model a sequence of activities with different durations, one can explicitly model each activity at different time steps as separate states. Each activity time slice $a_r^{(\tau)}$ represents the $r^{\text{th}}$ activity in the plan trajectory at time step $\tau$ since the beginning of the activity. At each time step, an activity $a_r^{(\tau)}$ can transition either to itself $a_r^{(\tau+1)}$ (downward) or to the next activity $a_{r+1}^{(\tau)}$ (rightward). . . . .	135





# List of Algorithms

4.1	OFFLINEACTIVITYLEARNING ( $\mathcal{T}, L, T(0)$ )	54
4.2	IDENTIFYMOTIONVARS ( $\mathcal{S}$ )	59
4.3	MAKEPFT ( $\mathcal{S}, \mathcal{F}, T(0)$ )	61
4.4	PRELEARNPFTs ( $\mathcal{S}, \ell, \mathbf{T0}$ )	67
4.5	GETPFTsFROMHERE ( $\mathcal{L}, T(0)$ )	68
5.1	ONLINERECOGNITION ( $\mathbf{PFT}_L^{T^{curr}(0)}, L, T^{curr}, \mathcal{W}$ )	72
6.1	OFFLINEPLANLEARNING ( $\mathcal{S}, \ell', T(0), \mathcal{L}$ )	81
6.2	RECOGNIZETASKSUSINGKEYFRAMETRIALS ( $\mathcal{Y}, \mathcal{L}$ )	84
6.3	LEARNUNKNOWNACTIVITIESINPLAN ( $\mathcal{S}, \mathcal{L}, q, \mathbf{T0}$ )	88
6.4	UPDATEUSINGPOSITIONMOTIONVARS ( $T, q, \mathcal{F}, K$ )	95
6.5	UPDATEUSINGORIENTATIONMOTIONVARS ( $T, q, \mathcal{F}, K$ )	96
6.6	UPDATEUSINGRECOGOPTIMIZATION ( $T, q, \mathcal{L}, K$ )	97
6.7	GENERATEPLANPFT ( $\ell', q, \mathcal{L}, T0$ )	100



# Chapter 1

## Introduction

This thesis presents an approach to enable more natural interaction between humans and robots by allowing an agent to learn and recognize complex manipulation tasks based on human demonstrations. This capability is necessary for agents to play a more collaborative role in human-robot interactions, moving beyond the standard master-slave relationship of humans and computers today. This thesis provides an enabling capability for learning from user demonstration and recognizing human operated motions, through offline task learning and online recognition at both the primitive activity level and more complex plan level.

In this chapter, I discuss the motivations for this research in Section 1.1 and provide a problem description in Section 1.2. Next, I outline my approach to the problem in Section 1.3. Finally, I lay out the roadmap for the rest of the thesis in Section 1.4.

### 1.1 Motivation

Robots can act as proxies for human operators in environments where the human operator is not present or cannot directly perform the task. These situations are pervasive in our world today: robots have been designed to aid in the aftermath of the recent nuclear reactor meltdown in Japan [32], assist in cleaning up the oil spill in the Gulf of Mexico [51], perform intravehicular and extravehicular activities on the

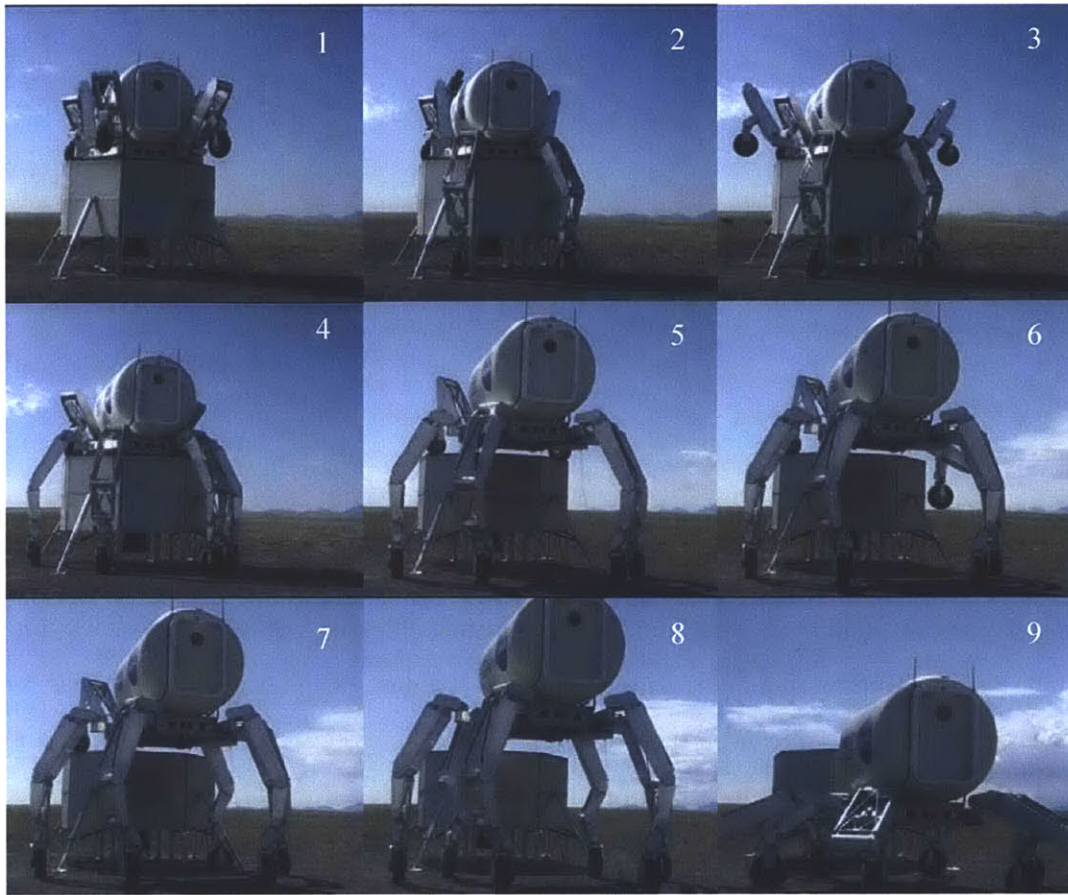


Figure 1-1: Docked Tri-ATHLETE robots transporting a habitat off of a lunar lander mock-up. The initial and goal states are shown in the first and last images, respectively.

International Space Station [4, 16], transport space habitats on the moon [74], disarm bombs in warring regions [10], extract injured soldiers from the battlefield [24], and even execute surgeries in the operating room [45].

Many current day robots are controlled manually from low level commands. An example is JPL's ATHLETE (All-Terrain Hex-Legged Extra-Terrestrial Explorer), which has 36 joints that can be independently controlled. It is designed to transport large payloads on the surface of the Moon. Figure 1-1 shows two units of a newer version of ATHLETE called Tri-ATHLETE docked together to carry a habitat off of a lunar lander mockup. Currently, the robots are commanded at the level of joint angles, with only a few higher level commands such as moving the end effector of the

robot in a certain direction. As a result, the process of moving the habitat off of the lander shown in Figure 1-1 spanned several hours during the field test.

Teleoperation is a common alternative interface for controlling robots that are designed to be human proxies. Unfortunately, motions that humans can produce naturally may not maintain the same fluidity through teleoperation due to limitations in the interface such as communications delay, non-immersive sensing, and controller uncertainty [64]. Such teleoperation interfaces can make it tedious and difficult for operators to perform a given task. A demonstrated undesirable consequence of fatigue during a telepresence operation is a reduction in the achieved precision during the task [70].

Instead of either direct low level commanding or continuous teleoperation, I envision a robot that can recognize a teleoperator's intended motion and autonomously continue the execution of recognized routine tasks. To do this, the robot first learns offline a library of generalized activities from a training set of user demonstrations. During online operations, the robot can perform real-time recognition of a user's teleoperated motions, and if requested, autonomously execute the remainder of an activity. The real-time motion recognition problem involves determining which motion in the learned activity library an operator is currently most likely executing.

In many real-world motions, local state information is not enough to identify the motion. For example, if a rope is made into an anchor loop around left and right anchors as shown in Figure 1-2, it is necessary to know if the loop passes around the left anchor first followed by the right one, or vice versa, in order to undo the loop properly. Locally, the looping motion around each anchor looks the same in both cases. Therefore, a model with a Markovian assumption will have difficulty distinguishing between the two motions due to its limited history capacity. Instead, I choose a representation that reflects the temporal history of the entire motion.

My approach builds upon the capabilities of existing motion recognition algorithms while providing three important features. First, since physical manipulation tasks are often non-Markovian, in that later parts of a motion may depend on past states, I use a model that can describe non-Markovian motions by leveraging temporal

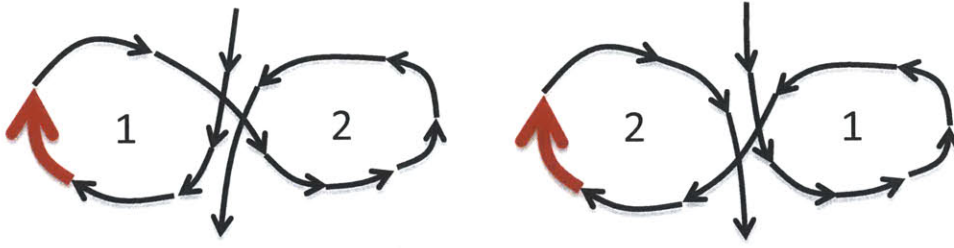


Figure 1-2: Knowing only the previous state at the highlighted position will not distinguish the two motions.

information. Second, my approach achieves real-time performance by efficiently sharing information across consecutive time steps. Third, the model of learned behaviors is robust to variations in initial environment states.

I have developed a representation called *probabilistic flow tubes* (PFTs) [20, 19, 21], which are used to generalize the demonstrated motion to be applicable for new situations. The covariance of the flow tube varies with the precision with which the demonstrated motions are performed. For example, a motion that moves a limb to the ground may have lots of variation in the trajectory initially, but when the limb reaches closer to the ground, the motion is executed with more precision to avoid crashing into the ground. Such a motion’s flow tube would have larger covariance initially and narrower covariance toward the ground. During execution, the controller should follow the flow tube trajectory as well as possible, with deviations from the trajectory penalized according to the covariance of the flow tube at that time. In this way, the probabilistic flow tube representation provides robustness during execution.

Constraint-based flow tubes have been used in the context of planning and execution to represent sets of trajectories with common characteristics [26, 38]. In this context, a flow tube defines a state region where valid trajectories of a motion can be feasibly achieved given constraints on the system dynamics. In the motion learning context, a probabilistic version of a flow tube is computed by inferring the desired Gaussian state distribution at each time step from human demonstrations.

Geometrically, as shown in Figure 1-3, the width of a probabilistic flow tube represents flexibility in the robot’s desired movement, enabling it to optimize additional performance criteria or recover from disturbances. The PFT representation produces

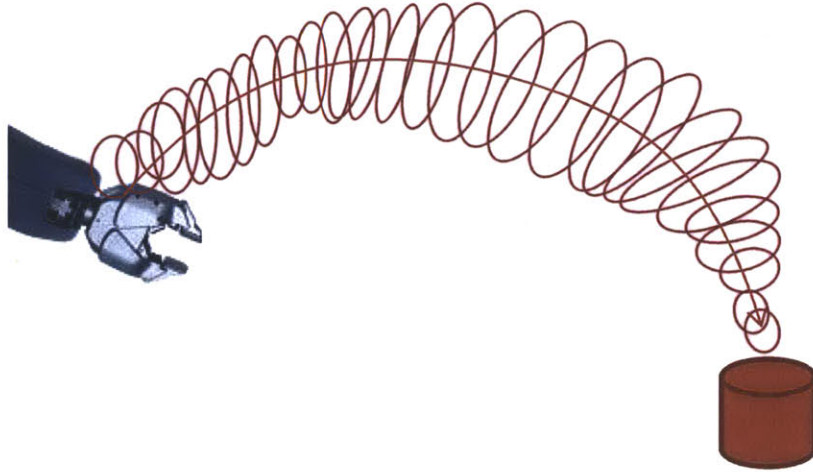


Figure 1-3: Example visual representation of a probabilistic flow tube

humanlike trajectories because it is directly modeled from the user’s demonstrations with minimal abstraction. In contrast, some motions generated by planners may not be intuitive for human collaborators, even though they may be valid. Furthermore, the PFT representation can be easily applied to situations with different initial conditions because it is parameterized by the relevant variables of the motion.

Manipulation tasks of interest in this work can be either primitive activities, which are described by important motion features at the start or end of the motion, or complex plans composed of sequences of primitive activities. An example of a primitive activity might be “reach for box,” where the robot end effector starts from some location and ends near the box location. An example of a complex plan might be “stack boxes in corner,” which might involve the sequence of primitive activities {“reach for box1”, “move box1 to corner”, “reach for box2”, “move box2 on box1”}. To learn a task, the system must keep track of all the important features in the task so that if the robot needs to autonomously execute the task in the future, the defining features of the task are preserved, even if the environment is different.

When a user demonstrates a complex plan through teleoperation or kinesthetic teaching, he or she might indicate when one subtask is complete and the next begins, using interfaces such as voice, keyboard, or other devices. However, this is an added layer of effort for the user, so a useful learning system should not require the user

to provide segmentations for every demonstrated trial. Thus, when learning complex plans from user demonstrations, the system also has the added challenge of inferring the points of segmentation in most trials from only a few pre-segmented trajectories provided by the user.

## 1.2 Problem Description

There are three main problems that this thesis focuses on: learning primitive motions, learning complex motions, and recognizing motions in real-time. The term *activity* is used to describe primitive motions, and *plan* is used to describe complex motions composed of sequences of activities. The term *task* is used to generically refer to either activities or plans. An overview of how the learning and recognition problems relate to each other is illustrated in Figure 1-4.

Task learning refers to the problem of determining the relevant features of a motion and creating a generalization of it (in the form of probabilistic flow tubes) that can be applied to new situations, given a set of labeled user demonstrations. The demonstration trajectories are recorded as a hybrid combination of continuous and discrete values. Specifically, the input data includes position and orientation information for the robot end effector during teleoperation or kinesthetic teaching, position and orientation of other sensed objects in the environment, other single dimensional continuous variables such as temperature or voltage, and discrete variables such as whether the power is on or off or whether the gripper is open or closed.

Both activity learning and plan learning aim to achieve the same goals of identifying relevant motion variables and creating a general representation of the motion to apply to different situations. The added complication in plan learning is that input user demonstrations may not necessarily be pre-segmented, since providing this extra information may be an additional burden on the user. I assume that out of all the user demonstrations for a particular motion label, only a few contain segmentation information. The system, therefore, must autonomously extrapolate this information onto the non-segmented demonstrations.



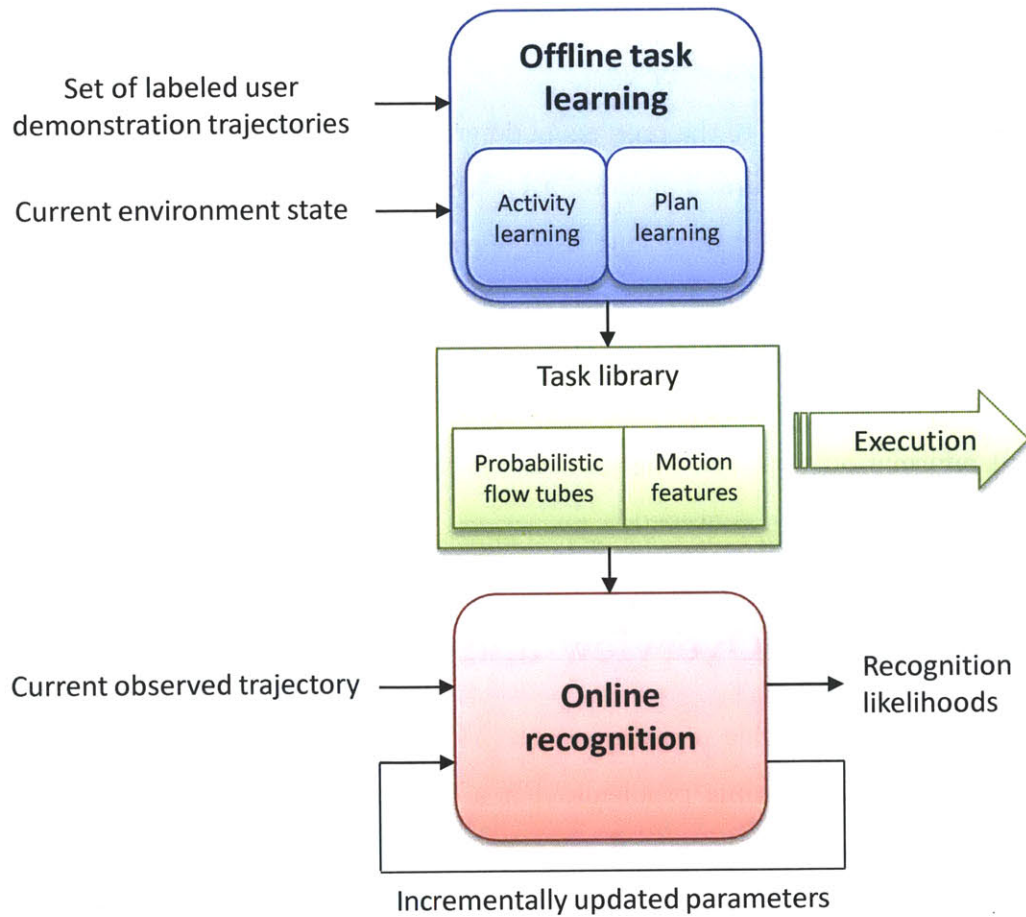


Figure 1-4: Architecture of learning and recognition problems

After a task or a set of tasks is learned in a generalized form, it can be stored into the task library for future use. For example, tasks generalized as probabilistic flow tubes can be sent to a controller for autonomous execution, or sent to the online motion recognizer to perform real-time recognition of a new user execution. Controller design is not within the scope of this thesis, but I envision a controller that can follow the mean trajectory of the probabilistic flow tube while dynamically adjusting the amount of allowable deviation based on the flow tube’s width over time by tuning parameters like stiffness, speed, or gain.

Task recognition refers to the problem of determining how likely a user is currently executing one of several learned motions, given the currently observed user execution trajectory. This is useful during long user teleoperation procedures, to enable the system to recognize which task the user is trying to perform, and potentially autonomously complete the task, saving the user some amount of effort. To enable the recognition process to compute in real-time, the system reuses as much as possible the analyzed information from one time step to the next, and only updates recognition parameters incrementally as needed.

## **1.3 Approach Overview and Innovations**

I now provide a brief overview of my approach to each of the activity learning, activity recognition, and plan learning problems. For a detailed discussion of each of these approaches, see Chapters 4 through 6.

### **1.3.1 Activity Learning**

There are two key innovations in my approach to activity learning. First, from observing different demonstration trials of a motion, the system is able to determine the important features that define the motion, without any a priori knowledge. For example, suppose a sensing system is constantly tracking the positions of a box, ball, and bin in the environment. Suppose the objects are initially located in different places for every demonstrated trial. If the demonstrated motion is “move box to

bin,” then only the box and bin positions are important while the position of the ball is irrelevant for this motion. By comparison, a motion to “move up 1 foot” is not concerned with any of the objects, but instead, is focused on the relative difference between the start and end positions of the robot. The approach determines what features are relevant or important in a motion by observing which features are persistent throughout different trials. When learning a new motion, the system asks the following questions: (1) Do the trials all start or end in the same absolute position? (2) Do the trials all start or end relative to a particular object or point of interest in the environment? (3) Do the trials all display a relative movement from the start to end positions? If a consistent pattern is noted across different demonstration trials, then the system identifies that pattern as a relevant motion characteristic. Once these characteristics are found for a motion, the algorithm can create a motion model in a new environment while ensuring that the same motion characteristics hold.

The second key innovation in my activity learning approach is the development of the probabilistic flow tube model to represent a motion. The idea of the probabilistic flow tube is grown from prior art in the planning and execution field, where constraint-based flow tubes have been used for feasibility analysis in controller design [26, 38, 66, 76]. In these applications, flow tubes are computed from the dynamics of the system to describe the region of possible trajectories. In the manipulation domain, the dynamics are difficult or impractical to compute, so activity learning addresses the inverse

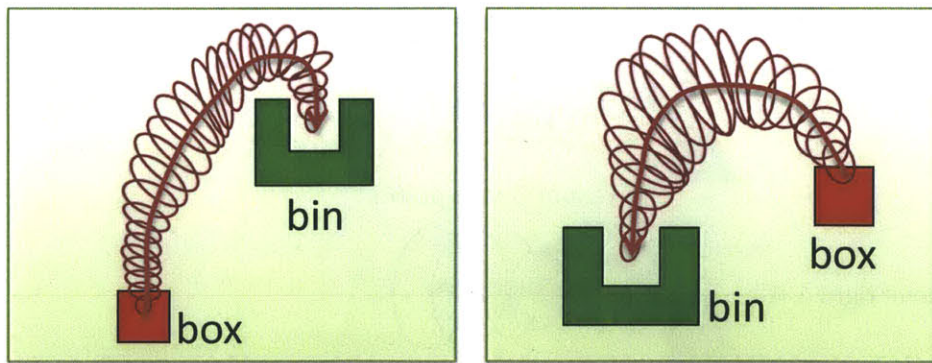


Figure 1-5: Probabilistic flow tubes can look drastically different for different environment states

problem of generating the flow tube probabilistically from example trajectories. A motion’s probabilistic flow tube is different for different initial environment states. For example, a robot’s motion during a “move box to bin” task may look drastically different if the box is initially to the left of the bin as opposed to the right, as shown in Figure 1-5. In a new environment state, the algorithm first determines what the start and end states of the motion should be given the identified motion variables. Next, it normalizes similar training trajectories to the desired start and end states to generate a mean and covariance trajectory, which make up the probabilistic flow tube. This probabilistic flow tube is then stored into the task library for future use such as autonomous execution or real-time recognition.

### 1.3.2 Real-time Recognition

My approach to real-time recognition has three main components. In order to determine the likelihood that a user’s current partial execution is of each learned motion, the approach first determines the time step in each learned probabilistic flow tube that best corresponds to the user’s current executed state. Suppose a user has currently moved the robot half a foot to the right, in the direction of a distant object, as illustrated in Figure 1-6. At this moment, an observer might estimate that the user could be executing a motion such as “move right 1 foot,” or “reach for the object.” If the correct motion were “move right 1 foot,” then the point in the learned PFT

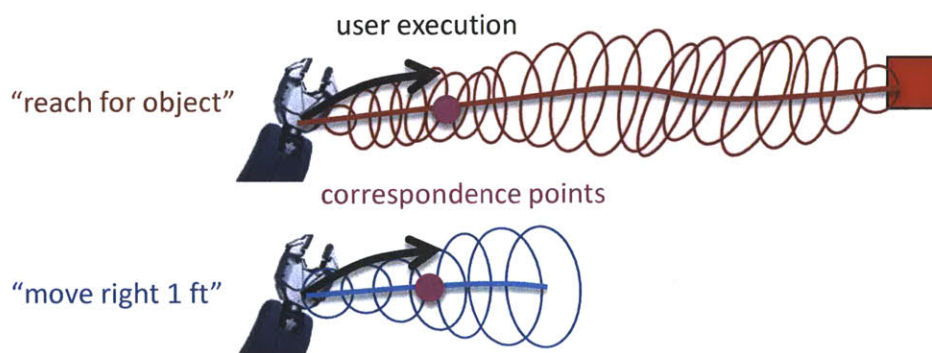


Figure 1-6: Given a user’s current execution (black arrow), the algorithm first determines correspondence points (pink dots) in each candidate learned motion

for the motion that best corresponds to the current executed state should be roughly in the middle of the motion. If the correct motion were “reach for the object,” then the correspondence point should be more toward the beginning of this motion’s PFT since the object is assumed to be quite far away in the environment, and the execution has not reached very far yet. The algorithm determines these correspondence points in the PFTs by looking at both how far the current executed state is spatially from each PFT and how much time has passed in the execution as compared with the trained models.

Once the system determines how much of each PFT corresponds to the user’s current execution, the next component in the recognition approach is to temporally align the user’s partial execution to the corresponding PFT portions to remove temporal variations among different executions. An existing algorithm called dynamic time warping exists to perform temporal matching between two trajectories, but my online recognition approach implements an incremental version of this algorithm that minimizes computation from one time step to the next by intelligently keeping appropriate parts of certain algorithmic parameters in memory.

Finally, after the user’s execution is temporally aligned with appropriate portions of the learned PFTs, the last component of my recognition approach uses the mean and covariances of the PFTs to compute the likelihood that the user’s partial execution actually belongs to each PFT. The result of online recognition is a set of such log likelihoods over the different motion labels at a particular time step of the execution. New information as the user’s execution progresses will result in updated recognition likelihoods at each new time step.

### **1.3.3 Plan Learning**

In order to teach a robot a compound activity, users generally find it more natural to provide a demonstration for an entire plan in one continuous movement rather than demonstrating each subtask separately. To generalize complex tasks consisting of multiple activities in sequence, I developed a plan learning approach to overcome two major challenges. First, not all the activities demonstrated in the plan are necessarily

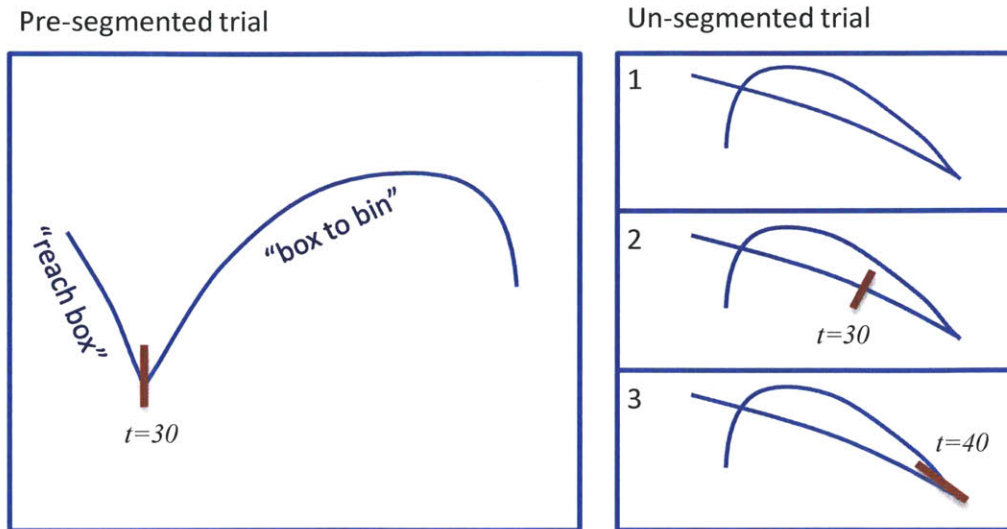


Figure 1-7: Illustration of auto-segmentation

in the task library or learned in the past. The system may be encountering certain activities in the plan for the first time. For these previously unknown activities, the system uses the segments of the plan demonstration trials that correspond to the unknown activity to learn that activity right away, as part of the plan learning process.

Secondly, I assume that only a small subset of the demonstrated trials have user provided segmentation information. My approach leverages the few pre-segmented trials to auto-generate segmentation points on all other trials. As shown in the illustrated example in Figure 1-7, the algorithm first initializes candidate segmentation points in un-segmented trials proportionately in time as the pre-segmented trials (panel 2), then updates each candidate segment to reflect what the system knows should happen based on the motion variables in the activity. Suppose running the recognizer over the pre-segmented trials reveals that the plan’s activity sequence is {“reach box”, “move box to bin”}, then the system knows that the point of segmentation between the two activities should be whenever the robot end effector meets the box (panel 3). Finally, the system can validate the auto-generated segmentation points with the user through an interactive display.

## 1.4 Thesis Layout

I present this thesis in the following manner: In Chapter 2, I perform a literature review of related work in the areas of learning from demonstration, motion recognition, flow tubes, and other related fields. In Chapter 3, I present formal definitions of the problems to be solved in this thesis. I describe the approach and algorithms of activity learning in Chapter 4, activity recognition in Chapter 5, and plan learning in Chapter 6. Next, I present results in a two-dimensional world as well as on several hardware platforms in Chapter 7. Finally, I discuss possible future advancements in Chapter 8. This concludes the introduction as I move into the details of the work.





# Chapter 2

## Related Work

### 2.1 Learning from Demonstration

Learning and recognizing human motions has long been an interest for human-robot interaction. [6, 40]. For example, learning human-taught policies has proven useful in the domains of underactuated pendulum control [7], autonomous helicopters [12], and vehicle navigation [1]. My work focuses on manipulation tasks, where interaction with objects in the environment becomes important. My approach is inspired by existing work in learning manipulation tasks from demonstration.

Peters and Campbell [52] demonstrated automatic skill acquisition by the humanoid Robonaut developed at NASA Johnson Space Center. Robonaut was shown several demonstrations of a task through teleoperation, and the data was time normalized and averaged to produce a characteristic sequence that can be linearly scaled to be autonomously executed in new situations. Their work showed that learning from teleoperated demonstrations is an attractive approach to controlling complex robots. With a more robust motion representation and better adaptability to new situations, this type of approach will become compelling for a wider range of applications.

A comparable approach to mine is that developed by Muhlig et al. [43], who demonstrated an imitation learning framework that allows a robot to autonomously perform tasks that were shown to it by human demonstration. Specifically, the demonstrated task was to pour fluid from a bottle to a cup, and experiments were run on

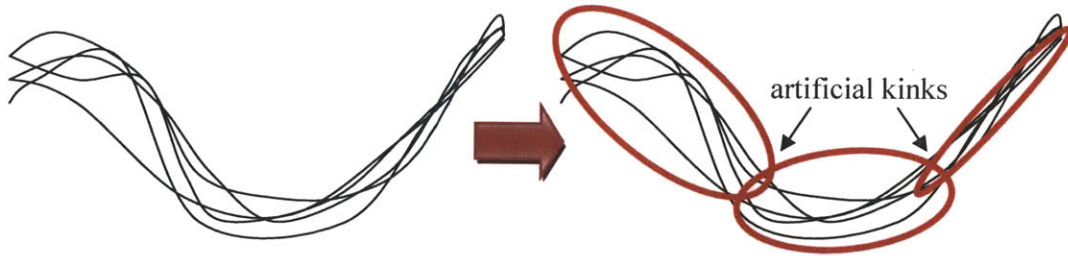


Figure 2-1: Mixtures of Gaussians may not be an intuitive representation of continuous motions.

Honda’s humanoid ASIMO robot. The authors used task spaces to model motions, which allowed them to track only object trajectories instead of full human postures. They then applied Dynamic Time Warping (DTW) [44] to normalize the temporal information in the demonstrated motions, and described the resulting normalized motion using Gaussian Mixture Models (GMM). To autonomously reproduce the motion, the authors used the learned probabilistic trajectory to initialize an attractor-based movement generation algorithm that takes into account additional criteria such as collision avoidance.

One major difference between my approach and that of Muhlig et al. is the use of probabilistic flow tubes instead of Gaussian Mixture Models to represent the probabilistic motions. Determining the optimal number of Gaussian components in a GMM approach can be somewhat arbitrary or cost-intensive [60]. Furthermore, I argue that while mixtures of Gaussians work well in applications concerning discrete clusters such as in localization [53] and classification [72], they do not describe continuous motions as intuitively as probabilistic flow tubes. As shown in Figure 2-1, GMMs can create artificial ‘kinks’ in the probability distribution of a learned trajectory.

Pastor et al. [49] also address the problem of learning general motions from demonstration. Their approach differs from mine because they represent the demonstrated movement with a set of differential equations describing a linear spring system perturbed by an external forcing term, whereas I represent motions with probabilistic flow tubes. Their representation is motivated by the dynamics of the system and can describe any smooth motions, whereas I learn the demonstrated trajectories directly,

and thus my learned motions closely reflect those of the human's, even if the motions are not smooth. Their actions are generalized by changing the goal parameter in the differential equations, and they record these actions in a library of movement primitives so that complex motions can be composed by sequencing. My approach also maintains a library, but consisting of both primitive and higher level motions.

Hsiao and Lozano-Perez [27] have developed a method that uses imitation learning through teleoperation to teach a robot to perform complex grasping of differently shaped objects. Their work differs from mine in their focus on the grasping task domain, where a large part of the problem is in identifying appropriate contact points and determining the appropriate amount of force to apply. Instead, my motion learning approach is more concerned with determining the trajectory that a human would take throughout a task.

An important part of my approach is the ability to identify the relevant features of a motion, so that a learned flow tube can be adjusted to new environments appropriately based on relations in existing data. Alissandrakis et al. [3] developed a system called Jabberwocky that, given a single demonstration and the desired effect metrics (which, in my terminology, are the motion characteristics), can produce the same effect of the motion in a new environment setup. The problem they address is quite different from my motion characteristic identification problem: in their problem, the metrics are known in advance and the goal is to produce an imitation of the demonstrated behavior using those metrics, whereas in my problem, the metrics (or motion characteristics) are not known in advance and have to be learned from observing patterns across many demonstrations. I agree with Alissandrakis et al. in their assessment that “the choice of metrics used is very important as it will have an impact on the quality and character of the imitation,” and thus I chose to allow the system to autonomously learn them. I note that the candidate metrics presented in their paper are very similar to mine, including “relative displacement,” “absolute position,” “relative position,” “rotation,” and “orientation,” suggesting that these candidate metrics are good choices for describing manipulation motions.

Calinon et al. [9] present an approach to learn demonstrated movements for mo-

tion generation. Their approach represents motions as HMMs learned using the EM algorithm, and employs Gaussian Mixture Regression to compute the desired velocities (and accelerations) for autonomous motion generation. The HMM representation is starkly different from the probabilistic flow tube (PFT) representation used in my approach. Learning an HMM using EM requires determining the optimal number of states using, for example, a BIC criteria, which can be computationally intensive. Furthermore, the resulting number of states is usually small: for example, 4 states in the “S” shape motion presented by Calinon et al. In contrast, probabilistic flow tubes approximate the region of flexibility in the trajectory by modeling a single Gaussian at every time step. Despite having many Gaussians, the lack of a need to compute Gaussian mixtures actually enables the PFT representation computation to be quite efficient. For these reasons, the HMM representation is more compact while the PFT representation is more detailed. The HMM representation presented by Calinon et al. is closely related to that in Martin et al. (2010), to which I compared my results, with the difference being that Calinon et al.’s HMM incorporates velocity data whereas Martin et al.’s HMM does not.

Calinon et al. handle landmarks (objects in the environment) by first expressing the training trajectories in the reference frames of each landmark, computing an HMM in each landmark’s reference frame, and upon the introduction of a new environment state, project the respective HMMs back onto the robot torso frame, after which taking a product of the corresponding Gaussians in the HMMs produces the final representation in the new environment. In this setup, the relevant landmarks must be given in advance. If irrelevant landmarks are included at random locations during each demonstration, they could skew the final motion representation in unnecessary or undesirable ways. Also, it is unclear if their approach would be able to handle motions that, for example, start relative to a landmark that can be moved around, and end at an absolute location, since it is unclear if their approach deals with motions that have “relative displacement” or “absolute position” characteristics. Furthermore, training trajectories for cases like these that could start all over the place and end in some absolute location can have a radial appearance, in which case it is unclear how the EM

learning will determine where the Gaussian mixtures of the HMM should go. These are all cases that are handled by the motion characteristic identification approach developed in this thesis.

Cederborg et al. [11] present an unsupervised motion learning and generation approach that first stores all of the demonstrated data of all different motions in one data structure, then when given a starting state, selects a set of training data points that are approximate nearest neighbors of that state, uses EM to learn a GMM with a predefined number of mixtures on the local points, and uses GMR to estimate the desired velocity for the next time step. Their motion representation is distinctly different from my probabilistic flow tube representation. Their learning process occurs entirely online at every time step during autonomous execution; after one time step is over, the previously learned local model is forgotten and the algorithm moves on to learn the next Gaussian mixture for the subsequent time step. My approach performs learning globally over entire motions offline and uses the learned flow tubes for online recognition. It is unclear in Cederborg et al. how their local algorithm handles, for example, motion trajectories that have different sized loops that tangentially meet at the same local state: for example, how will it know which branch is the correct one to take at that point? The global temporal information used in the PFT representation prevents my approach from getting "lost" locally.

Cederborg et al. consider three possible reference frames in which a motion can be performed: relative to the starting position, relative to the robot frame, and relative to an object position. The desired velocity estimated at every state is the weighted sum of the desired velocities in each reference frame, where the weights reflect how well the data trains in each frame. In all of their examples and results, the motion belonged to exactly one of the reference frames; no examples were given of motions that span multiple reference frames, such as moving an object to an absolute location, so it is uncertain how their algorithm will perform in these cases. My approach does not view the entire motion in multiple reference frames; instead, relevant motion characteristics are determined at the start and end of a motion (which can correspond to different reference frames in their approach), and the motion trajectories are adjusted

accordingly.

Lee and Ott [36] present an approach to combine teaching by observational demonstrations (where a robot watches what a human does) with kinesthetic demonstrations (where a human physically moves the robot around). Here the idea is that a robot will learn the bulk of the motion from observational demonstrations, and then refine the more precise manipulation parts of the movement from direct user guidance. They present a representation called a refinement tube that "helps the human teacher to correct only the desired part of the motion without accidentally disturbing other joints" and present an impedance controller that increases the robot's stiffness once outside the refinement tube. Despite the common word "tube" in the names of their representation and mine, the essence of the two representations are distinctly different. Their approach is as follows: use EM to learn an HMM on the training data where the states are a GMM and the number of states is given in advance, then use the Viterbi algorithm to determine the optimal state sequence through the HMM after adding in temporal data. This sequence of Gaussians in space-time is used to determine the refinement tube, whose radius is defined as a function of the conditional variance at a given time step. In the conducted experiments, Lee and Ott chose to set the number of learned HMM states to 10. The resulting refinement tube bulges at time steps close to the middle of each Gaussian, and narrows at time steps in between different Gaussians; had the number of states been chosen at 2, the refinement tube would have two bulges. It seems undesirable that the refinement tube may arbitrarily change shape depending on the chosen number of states. In my PFT representation, Gaussians exist in the spatial dimensions at each time step. If time steps were sampled less frequently, the model would end up with a coarser PFT, but it would maintain the same basic shape. The problem Lee and Ott are solving is also quite different from the goals of this thesis. While I focus on online recognition of user motions, they work toward a better paradigm of teaching robots.

Riley and Cheng [57] present an unsupervised motion learning and generation capability that segments motions based on curvature, groups similar trajectories together, and generalizes the motion groups using linear regression where potentially

non-linear features are represented with radial basis functions. My approach differs from theirs in three main ways: (1) My motion representation describes not only the demonstrated motion itself, but also a probabilistic region of flexibility around it; their representation is geared more towards autonomously generating specific trajectories that are reasonable. (2) My approach is concerned with variable environments during demonstrations whereas theirs maintains the same environment state throughout different demonstrations. (3) I am focused on the online recognition problem after supervised motion learning, while they are focused on autonomous motion generation after unsupervised motion clustering.

The problem addressed by my probabilistic flow tube representation is also different from regression using Gaussian processes [56]. Regression treats all demonstration data points independently. Gaussian processes can then be used to determine a distribution over functions that best models the underlying function. In contrast, I am interested in modeling the time sequence in a way that leverages the temporal information. Consequently, my model maintains the temporal ordering of the data points in each demonstration, and Gaussian distributions are used to describe the spatial variability of the motion.

I note several other related works in the broader area of learning human motions. Blackburn and Ribeiro [8] used isometric feature mapping (Isomap) [67] for dimensionality reduction in human motion recognition in videos abstracted as silhouettes, and then used Dynamic Time Warping for motion pattern matching. They reported that Isomap combined with DTW achieved recognition rates of over 95% on the particular motions they used, which performed better or at least comparable to methods using locally preserving projections (LPP) for dimension reduction with a hidden Markov model (HMM) approach for recognition, as presented by Wang and Suter [71]. This gives reassurance over the efficacy of approaches using dynamic time warping. Work by Jia and Yeung [30] introduced a manifold embedding method that considers both spatial and temporal discriminative structure of silhouetted actions in a supervised learning setting. Anthony [5] and Fitriani [22] developed an algorithm called Dynamic Time and Space Warping that was used for video matching and align-

ing. Although these approaches focus on domains different from manipulation tasks, they provide insight into the importance of utilizing both spatial and temporal information during motion learning and identification, which I also emphasize in my approach.

## 2.2 Motion Recognition

Motion recognition applications have ranged from visual gesture recognition [42, 73] to understanding domestic activities [25] to gait analysis [23], among others. I am mainly interested in learning and recognizing teleoperated manipulation tasks, although my approach is potentially extensible to other applications.

Recognition in the context of teleoperation has commonly been explored using Hidden Markov Models (HMMs) [39, 64, 75]. Specifically, Martin et al. [39] model motions learned from training data as sequences of HMM states, where each state refers to a mixture of Gaussians, and recognition can be performed using either the Viterbi algorithm or posterior probabilities during model learning. Their approach proved promising in recognizing grasping tasks and provides a good basis of comparison.

Using probabilistic flow tubes, my algorithm performs recognition of a new partial motion by computing the likelihood of its being described by each model, after temporally aligning relevant time steps. Real-time performance is achieved by storing parts of the computation in memory and using an incremental version of dynamic time warping [44] for temporal matching.

## 2.3 Flow Tube Background from Plan Execution

Hofmann and Williams [26] proposed a robust plan execution approach that allows for spatial and temporal plan flexibility for under-actuated systems performing mixed discrete-continuous motions. The example used in their paper is a biped walking on uneven terrain. Their approach compiles a temporally flexible plan, called a Qual-



itative State Plan (QSP), into a concurrent timed flow tube description called a Qualitative Control Plan (QCP) that represents all feasible control trajectories and their temporal coordination constraints. During execution, the plan dispatcher only needs to maintain state trajectories within the bounds of the flow tubes. Thus this execution approach is robust to disturbances as long as they do not perturb the states outside the flow tubes.

The Qualitative State Plan is an input to their system that is manually defined by the user. In applications where the user may not have a predefined idea of the exact steps of a task, I propose to allow the user to start controlling the robot at a lower level right away, and have the system learn possible plans along the way by observing the user’s interactions.

Furthermore, Hofmann and Williams compute the flow tubes in their Qualitative Control Plans by performing reachability analyses, given the kinematics of the system. Instead of deriving the flow tubes brute force, my approach uses human teleoperation to demonstrate a few feasible state trajectories, and then learns a probabilistic flow tube from the data. Learning from the user is advantageous because it can produce a more intuitive motion. For example, experience with the Barrett Whole Arm Manipulator robot has shown that it can reach a certain location in a normal forward motion, or in an awkward backward-handed way that is surprising to the user. In human-robot interaction, the human operator needs to be able to trust the robot, so the robot must behave in an intuitive way. The best way to determine what is intuitive to a human is to learn directly from them.

## 2.4 Adaptive Interfaces

Although my plan learning capability applies human-robot interaction during training and learning, it is different from prior art in adaptive interfaces such as [33, 62, 31, 29, 65, 68] where the focus is on designing user interfaces that change appropriately with the task. For example, Kazi et al. [33] describe a multimodal interface that uses speech and gesture along with stereo visual sensing to assist disabled persons,

and Serenko [62] describe an intelligent assistant for organizing e-mail. In contrast, I focus on learning to execute physical actions rather than adapting to behavioral tendencies of a user. I am also less concerned with the actual display of the interface than with enabling the agent to gather information through the interaction process.

# Chapter 3

## Problem Statement

An illustrated example of the learning and recognition problem is shown in Figure 3-1. In this example, user demonstration training data is collected in a 2D world consisting of three movable objects: “box,” “ball,” and “bin.” The user has given several demonstrations each of three different types of motion: “move box to bin,” “move box left,” and “move box home,” in which “home” refers to the center position of the environment. Given a new environment setup where the objects are in different locations, the offline learning algorithm computes a probabilistic flow tube in the new environment for each type of motion. As the user starts to execute a motion in this environment, the recognition algorithm determines in real time the likelihood that the user is executing each type of motion.

In my experiments, human motion data is collected directly using a teleoperated or kinesthetic teaching interface, where human driven motion is recorded through robot poses. Objects or other points of interest in the environment can be sensed through vision, laser range finding, infrared, motion capture, or any other standard devices.

In the rest of the chapter, I first define the variables involved in the inputs and outputs of the learning and recognition system, and then define three types of problems: activity learning, activity recognition, and plan learning.

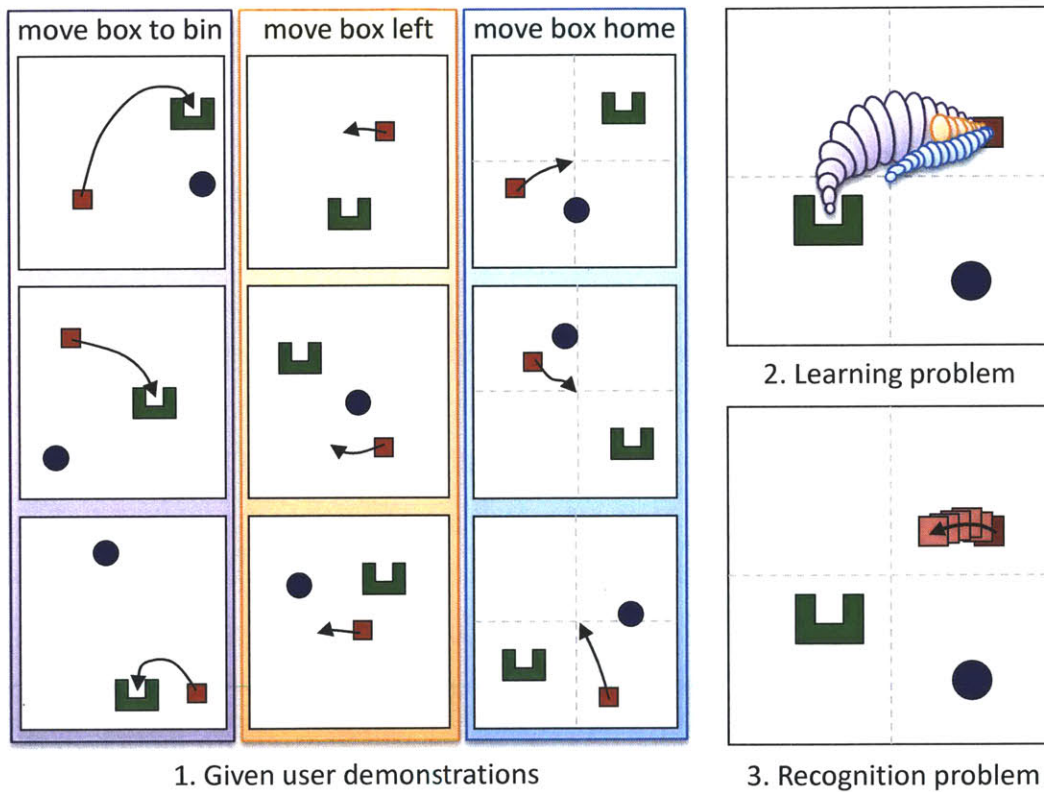


Figure 3-1: Illustrated example of a learning and recognition problem. Initially, a user demonstrates a set of motions a few times (1). The learning problem consists of generalizing the demonstrations into probabilistic flow tube representations for each of the three motions in the new environment state (2). Given a new user motion depicted as the shift in the red box position in the direction of the arrow, the recognition problem consists of determining which motion the user is most likely performing (3).

### 3.1 Definitions of Inputs and Outputs

A user demonstration trajectory is a hybrid mix of time-evolved continuous and discrete variables described in Definition 1. These variables capture the world state through time, including the position and orientation of the robot end effector, the position and orientation of objects or points of interest in the environment, other single dimensional continuous variables that describe the world, and other discrete variables in the world.

**Definition 1** A *demonstrated trajectory*  $T$  is a tuple  $\langle \mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q} \rangle$  denoting a sequence of values through time, where:

- $\mathbf{C}$  is a set of  $c$  single dimensional continuous variables at time steps  $t = 0 \dots N$ . Examples of such continuous variables include execution time, temperature, voltage, etc.
- $\mathbf{D}$  is a set of  $d$  discrete variables at time steps  $t = 0 \dots N$ . Examples of such discrete variables include gripper open/close, power on/off, etc.
- $\mathbf{P}$  is the set of Cartesian position variables  $x, y, z$  for each of  $b = 1 \dots B$  points of interest at time steps  $t = 0 \dots N$ .  $\mathbf{P}^{eff}$  denotes the position variables of the specific point of interest where  $b = eff$ , the robot end effector. The index  $eff$  is typically equal to 1.
- $\mathbf{Q}$  is the set of quaternion orientation variables  $q_1, q_2, q_3, q_4$  for each of  $B$  points of interests at time steps  $t = 0 \dots N$ . Similarly,  $\mathbf{Q}^{eff}$  refers to the orientation variables of the robot end effector.

Points of interest in the environment can be the robot end effector, objects or parts of objects that can be sensed, or other known markers in the environment. Future extensions of the learning algorithm may also utilize velocity  $\mathcal{V}$  and acceleration  $\mathcal{A}$  inputs for each point of interest over time, but is not within the scope of the current discussion. The manipulation tasks studied here are generally slow enough that position information alone is sufficient for good motion learning performance.

The state of the world at a particular time step  $t$  in the demonstrated trajectory is denoted by  $T(t)$ , which is the cross section of all the variables at that time step  $\langle \mathbf{C}(t), \mathbf{D}(t), \mathbf{P}(t), \mathbf{Q}(t) \rangle$ . Therefore the initial environment state of a trajectory is denoted as  $T(0)$ .  $N$  is the number of time steps for a particular demonstrated trajectory, and can be different for different demonstrations.

When a user is executing a trajectory in real-time, it is useful to record the partial execution of the motion by the current time step, especially for online recognition. This current execution of a trajectory is described in Definition 2.

**Definition 2** *A **current execution**  $T^{curr}$  is a user’s current partial execution of a trajectory from  $t = 0$  to the current time step  $t = t^{curr}$ .*

Every unique motion that the system learns is assigned a motion label, described in Definition 3.

**Definition 3** *A **motion label**  $\ell$  is a unique string associated with a task.*

Motion labels are typically provided by the user. In the absence of user interaction, motion labels can also be randomly generated by the system.

When the system is learning a task, it is typically provided multiple trials of user demonstrations for that task. Definition 4 describes the training set.

**Definition 4** *A set of **training sequences for a particular motion**  $S$  is the set of demonstrated trajectories  $\{T_k\}_{k=1..K}$  for a particular motion label  $\ell$ .*

The number of time steps in a training sequence  $N_k$  may vary among different sequences. To enable batch learning of multiple motions, the system can be provided a set of training sequences for multiple motions, described in Definition 5.

**Definition 5** *A set of **all training sequences**  $\mathcal{T}$  is the combined set of training sequences  $\{S_\ell\}_{\ell \in L}$  for all  $M$  motions with labels  $L = \{\ell_1, \dots, \ell_M\}$ .*

A probabilistic flow tube (PFT) is formally described in Definition 6. It is comprised of a mean trajectory and its corresponding covariances through time.

**Definition 6** A *probabilistic flow tube* associated with a motion label  $\ell$  and starting from environment state  $T(0)$  is denoted as  $PFT_\ell^{T(0)}$  and is composed of a tuple  $\langle T^{\text{eff}}, \Sigma^{\text{eff}} \rangle$ , where

- $T^{\text{eff}} = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}^{\text{eff}}, \mathbf{Q}^{\text{eff}} \rangle$  refers to a nominal desired robot end effector trajectory.
- $\Sigma^{\text{eff}} = \langle \sigma_{\mathbf{C}}, \sigma_{\mathbf{D}}, \Sigma_{\mathbf{P}}^{\text{eff}}, \Sigma_{\mathbf{Q}}^{\text{eff}} \rangle$  refers to the covariances throughout the trajectory at time steps corresponding to those in  $T^{\text{eff}}$ .

A probabilistic flow tube that is generated for a motion  $\ell$  is dependent on the starting environment state for which it was computed ( $T(0)$ ). PFTs created from different environment states can be quite different from one another. For example, a “move box to bin” motion trajectory will appear greatly different if the box is initially to the left of the bin versus if the box is initially to the right of the bin.

A task library, described in Definition 8, is composed of a set of task entries, described in Definition 7. Tasks can be either primitive *activities* composed of no subtasks, or *plans* composed of more than one subtask.

**Definition 7** A *task library entry*, or simply *task*, associated with a given label  $\ell$  is denoted by  $lib_\ell$  and is composed of a tuple  $\langle \mathbf{PFT}^{\mathbf{E}}, \mathbf{E}, \mathcal{F}, q \rangle_\ell$ , where:

- $\mathbf{E}_\ell$  is a set of initial environment states  $\{T_k(0)\}_{k \in K}$ .
- $\mathbf{PFT}_\ell^{\mathbf{E}}$  is a set of probabilistic flow tubes starting from the different initial environment states described by  $\mathbf{E}_\ell$ , or  $\left\{ PFT_\ell^{T(0)} \right\}_{T(0) \in \mathbf{E}_\ell}$ .
- $\mathcal{F}_\ell$  is a tuple of features  $\langle F_{\mathbf{C}}, F_{\mathbf{D}}, F_{\mathbf{P}}, F_{\mathbf{Q}} \rangle$ , also known as motion variables or characteristics that describe motion  $\ell$ . Each  $F_{\mathbf{X}}$  is a set  $\left\{ \langle \mu, \Sigma, \text{relevant} \rangle_{\mathbf{X}}^{\text{mode}} \right\}$ , where  $\text{mode} \in \{ \text{absStart}, \text{absEnd}, \text{relInit}, \text{relEffStart}, \text{relEffEnd} \}$ .
- $q_\ell$  is a sequence of motion labels  $\{\ell_1 \dots \ell_M\}$ , also known as subtasks, that compose task  $\ell$ . If  $\ell$  is a primitive activity, then  $q_\ell = \emptyset$ .

Here I only provide the organizational structure of the variables used in the system. For an in-depth description of motion variables  $\mathcal{F}$ , see Section 4.1.

**Definition 8** A *task library*  $\mathcal{L}$  is the library of task entries  $\{lib_\ell\}_{\ell \in L}$ , where  $L$  is the set of all task labels in the library.

When handling trajectories that represent plans, which consist of a sequence of activities, the system must know or compute the segmentation points between activities in the trajectory. These points of segmentation are described in Definition 9.

**Definition 9** A set of *keyframes*  $K^{key}$  are time steps in a plan trajectory that segments one subtask from the next in the plan. Keyframes are embedded into a demonstrated sequence  $T$  as part of the discrete variable  $D^{key} \in \mathbf{D}$ , where  $D^{key}(K^{key}) = 1$  and all other elements in  $D^{key}$  are zero.

Next, I formally define the different kinds of problems that the system is able to handle.

## 3.2 Definition of an Activity Learning Problem

The problem of learning an activity from user demonstrations is presented in Definition 10. The goal is to identify the important features of the activity and generate a probabilistic flow tube in a new environment state. The detailed approach to activity learning will be discussed in Chapter 4.

**Definition 10** Given a set of training sequences  $\mathcal{S}$  for activity label  $\ell$  and a starting environment state  $T(0)$ , determine the set of motion variables  $\mathcal{F}_\ell$  that are relevant in the activity, and generate a probabilistic flow tube  $PFT_\ell^{T(0)}$  that describes the robot end effector motion in the activity.

The outputs to the activity learning problem can be used for several purposes. First, the learned probabilistic flow tube can be sent to a controller for autonomous activity execution in the current environment state  $T(0)$ . The controller would try to follow the PFT’s nominal trajectory while using the covariances as a guide to the amount of deviation allowed.



Additionally, the resulting motion variables  $\mathcal{F}_\ell$  and probabilistic flow tube  $PFT_\ell^{T(0)}$  that are learned can be used to create a new task library entry

$$lib_\ell = \langle PFT_\ell^{T(0)}, T(0), \mathcal{F}_\ell, \emptyset \rangle$$

if the motion labeled  $\ell$  does not already exist in the task library. In the case that  $\ell$  already has an entry  $lib_\ell = \langle \mathbf{PFT}^{\mathbf{E}}, \mathbf{E}, \mathcal{F}, q \rangle_\ell$ , the entry can be updated with the new PFT and initial environment, producing

$$lib_\ell = \langle \{ \mathbf{PFT}^{\mathbf{E}}, PFT_\ell^{T(0)} \}, \{ \mathbf{E}, T(0) \}, \mathcal{F}, q \rangle_\ell.$$

The system is allowed to be able to learn multiple activities as a batch, so it is also possible to provide a set of training sequences for multiple activities  $\mathcal{T}$  with their corresponding labels  $L$  and expect a set of different motions' probabilistic flow tubes  $\mathbf{PFT}_L^{T(0)} = \{ PFT_\ell^{T(0)} \}_{\ell \in L}$  that all start from the same environment state as output.

### 3.3 Definition of an Activity Recognition Problem

The problem of real-time activity recognition as a user executes a motion is presented in Definition 11. Real-time recognition computes the likelihood that a user is currently executing any one of several activities. These likelihoods change at every time step with new user information. The detailed approach to activity recognition will be discussed in Chapter 5.

**Definition 11** *Given a user's current partial activity execution  $T^{curr}$ , a set of learned probabilistic flow tubes  $\mathbf{PFT}_L^{T^{curr}(0)}$  corresponding to activity labels  $L$  all starting at environment state  $T^{curr}(0)$ , compute in real-time, a set of log probabilities  $LL = \{ll_1, \dots, ll_M\}$  over the set of known labels  $L = \{\ell_1, \dots, \ell_M\}$  that reflect the likelihood that the label of the user's current motion is one of  $L$ .*

The resulting log likelihoods  $LL$  are computed for a specific moment in time, up to which the user's execution is  $T^{curr}$ . At the next time step, the user's execution

will have progressed, so a new set of log likelihoods would need to be computed.

### 3.4 Definition of a Plan Learning Problem

The problem of learning a plan from user demonstration trajectories is presented in Definition 12. I assume that a small subset of demonstration trajectories contain keyframes, or points of segmentation, indicating exactly when the trajectory transitions from one activity to the next. Users can provide this information through voice or keyboard inputs during demonstrations, or through data annotation post demonstration. Since providing this additional segmentation information is an additional effort for the user, I assume that most of the training demonstrations do not contain keyframes, and thus the system must extrapolate this information for most trials from the few provided pre-segmented trajectories. The detailed approach to plan learning will be discussed in Chapter 6.

**Definition 12** *Given a set of training sequences  $\mathcal{S}$  for plan label  $\ell$  (of which a subset  $\mathcal{Y} \subset \mathcal{S}$  contains keyframes), a starting environment state  $T(0)$ , and the current task library  $\mathcal{L}$ , generate a task entry  $lib_\ell$  that describes the plan and update the task library.*

Like activity learning, plan learning also involves generating a probabilistic flow tube that can be used for execution. Again it is worth pointing out that the plan’s PFT can vary vastly depending on the initial environment state. For example, the trajectory of a plan involving {“move box to bin”, “reach for ball”, “move ball to x”} can look drastically different for different initial locations of the objects. The system must utilize the important features of the subtasks in a plan to generate a correct PFT for a given environment.

### 3.5 Relationship to Prior Art

As discussed in Chapter 2, many researchers have addressed the learning from demonstration problem [52, 12, 49, 43], but in each instance, the important features of the

task is known in advance. For example, Pastor et al. [49] teach a robot arm to place a cup on a saucer while avoiding a ball obstacle. The roles of all the objects in the world are known in advance, and the system just needs to reproduce the task. In the problems addressed in this thesis, however, the relationships among objects and the robot end effector that are important for one task or another are not known in advance, and must be learned from the user demonstrations. For example, there may be a dozen objects detected in the world, and only objects 1 and 2 are relevant for task A while only object 10 is relevant for task B. This information is not known in advance in the problems addressed here, while approaches presented in prior art assume it is given.

I have now formally defined the inputs and outputs of the learning and recognition system as well as presented the problem statements of the activity learning, activity recognition, and plan learning problems. The next three chapters discuss my approach in addressing each of these problems.



# Chapter 4

## Learning Single Activities

The activity learning procedure is summarized in Algorithm 4.1. The inputs include the training set for all motions  $\mathcal{T}$ , the set of motion labels  $L$ , and a new environment state  $T(0)$  in which the test motion will be performed. The output is a set of activity models describing the training motions in the new environment. Additionally, certain computations are stored in memory during the offline activity modeling phase in order to achieve fast performance during online recognition later.

There are two major steps in Algorithm 4.1. First, for each labeled motion, the algorithm determines the important features or relations in the demonstrations, which is called *motion variables*  $\mathcal{F}$ . Then it uses the training sequences  $\mathcal{S}$  to create a probabilistic flow tube defined as  $\langle T^{eff}, \Sigma_{eff} \rangle$  that abides by the same relations ( $\mathcal{F}$ ) in the new environment  $T(0)$ .

The nominal robot end effector trajectory  $T^{eff}$  is defined as the tuple

$$\langle \mathbf{C}, \mathbf{D}, \mathbf{P}^{eff}, \mathbf{Q}^{eff} \rangle,$$

where  $\mathbf{C}$  and  $\mathbf{D}$  are collections of  $c$  and  $d$  single-dimensional continuous and discrete variables, respectively, through time,  $\mathbf{P}^{eff}$  is the 3-dimensional position variable of the robot end effector through time, and  $\mathbf{Q}^{eff}$  is the 4-dimensional quaternion orientation variable of the robot end effector through time. Correspondingly,  $\Sigma^{eff} = \langle \sigma_{\mathbf{C}}, \sigma_{\mathbf{D}}, \Sigma_{\mathbf{P}}^{eff}, \Sigma_{\mathbf{Q}}^{eff} \rangle$  is the covariance trajectory through time.

---

**Algorithm 4.1** OFFLINEACTIVITYLEARNING ( $\mathcal{T}, L, T(0)$ )

---

**Input:**

$\mathcal{T} = \{\mathcal{S}_\ell\}_{\ell \in L}$ ;  $\mathcal{S}_\ell$ , training set for motion  $\ell \in L$   
 $L$ , set of labels of all learned motions  
 $T(0)$ , a new environment state

**Output:**

$\mathbf{PFT}_L^{T(0)} = \{PFT_\ell\}_{\ell \in L}$ , set of augmented probabilistic flow tubes  
 $\mathcal{F}$ , set of relevant motion variables

**Notable local variables:**

$T^{eff} = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}^{eff}, \mathbf{Q}^{eff} \rangle$ , PFT end effector trajectory  
 $\Sigma^{eff} = \langle \sigma_{\mathbf{C}}, \sigma_{\mathbf{D}}, \Sigma_{\mathbf{P}}^{eff}, \Sigma_{\mathbf{Q}}^{eff} \rangle$ , PFT covariance trajectory

- 1: **for**  $\ell \in L$  **do**
  - 2:    $\mathcal{F} \leftarrow \text{IDENTIFYMOTIONVARIABLES}(\mathcal{S}_\ell)$
  - 3:    $\langle T^{eff}, \Sigma^{eff} \rangle \leftarrow \text{MAKEPFT}(\mathcal{S}_\ell, \mathcal{F}, T(0))$
  - 4:    $\mathbf{I} = \{\Sigma^{eff}(n)^{-1}\}_{n=1..N_\ell}$
  - 5:    $\mathbf{G} = \left\{ -\log \left( (2\pi)^{\frac{\dim(T^{eff})}{2}} |\Sigma^{eff}(n)|^{\frac{1}{2}} \right) \right\}_{n=1..N_\ell}$
  - 6:    $PFT_\ell = \langle T^{eff}, \Sigma^{eff}, \mathbf{I}, \mathbf{G} \rangle$
  - 7: **end for**
- 

Aside from the two major steps in lines 2 to 3 of Algorithm 4.1, the probabilistic flow tube is also augmented with some additional values that will be used during recognition. The recognition problem involves determining how closely an unknown motion follows each flow tube by computing probability density values based on the nominal trajectories and corresponding covariances of each PFT. To generate these probabilities quickly, the algorithm pre-computes the inverse covariance matrices

$$\mathbf{I} = \{\Sigma^{eff}(n)^{-1}\}_{n=1..N_\ell}$$

and probability density factors

$$\mathbf{G} = \left\{ -\log \left( (2\pi)^{\frac{\dim(T^{eff})}{2}} |\Sigma^{eff}(n)|^{\frac{1}{2}} \right) \right\}_{n=1..N_\ell},$$

where  $\dim(T^{eff})$  is the dimension of the trajectory, and  $N_\ell$  is the number of time steps activity  $\ell$ . Any function applied to the effector trajectory  $T^{eff}$  or covariance

trajectory  $\Sigma^{eff}$  is equivalent to applying that function to its subcomponents:

$$f(T^{eff}, \Sigma^{eff}) = \left\langle f(\mathbf{X}^{eff}, \Sigma_{\mathbf{X}}^{eff}) \right\rangle_{\forall \mathbf{X} \in \{\mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q}\}}.$$

Finally, all of these values computed for motion label  $\ell$  are stored in a data structure  $PFT_{\ell} = \langle T^{eff}, \Sigma^{eff}, \mathbf{I}, \mathbf{G} \rangle$ .

I now proceed to describe each of the two main steps in detail.

## 4.1 Identifying Motion Variables

A key feature of the learning system is the ability to autonomously determine what features or relations, if any, are characteristic of a particular demonstrated motion. I use the general term *motion variables* to describe a class of potentially important features or characteristics of a motion. Of these, the *relevant* motion variables are those preserved over different demonstrated trials of that motion, while other motion variables may vary due to changes in the environment or the human’s movement.

For example, demonstrated sequences of the motion “move box to bin” will show a pattern whereby the robot end effector starts at the location of the box, makes contact with it, moves to the location of the bin, and breaks contact with the box. The system will learn that the distance between the robot effector and the box is a relevant motion variable at the beginning of the motion, and that the distance between the robot effector and the bin is a relevant motion variable at the end of the motion. The system will also learn that the positions of any other objects known in the environment are not relevant to this motion.

In the implementation described here, motions are segmented at a subset of time steps determined by the operator, and the learning algorithm determines relevant motion variables at the endpoints of these segments. Typically these are time points corresponding to a qualitative change in the behavior of the task, such as the robot making or breaking contact with an object. In Chapter 6, I extend the implementation to automatically determine these temporal segmentation points from changes

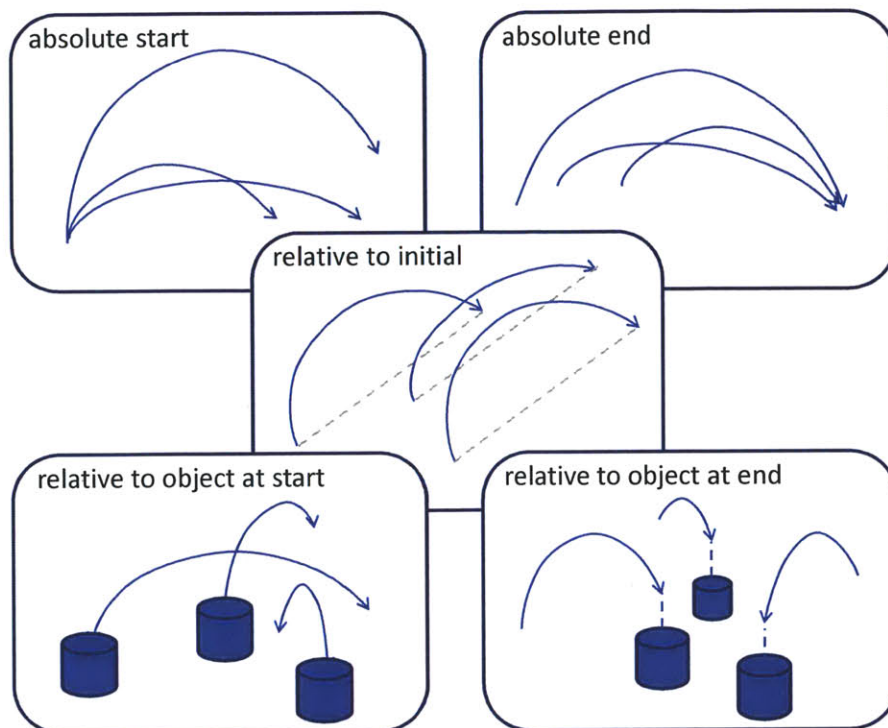


Figure 4-1: Example illustrations of five possible ways that motion variables can be relevant. Arrows refer to the robot end effector trajectories.

in the discrete motion variables. Thus I assume in the remainder of this section that demonstrated motions consist of a single segment.

For each of the continuous, discrete, position, and orientation input variables of a demonstrated motion, I consider up to five possible modes for candidate motion variables, as shown in Figure 4-1:

- *absStart*: Does this variable generally start at the same value across the demonstrations?
- *absEnd*: Does this variable generally end at the same value across the demonstrations?
- *relInit*: Does this variable generally shift the same amount from start to end across the demonstrations?
- *relEffStart*: Is the starting position or orientation of the robot end effector generally the same relative to certain points of interest across the demonstrations?



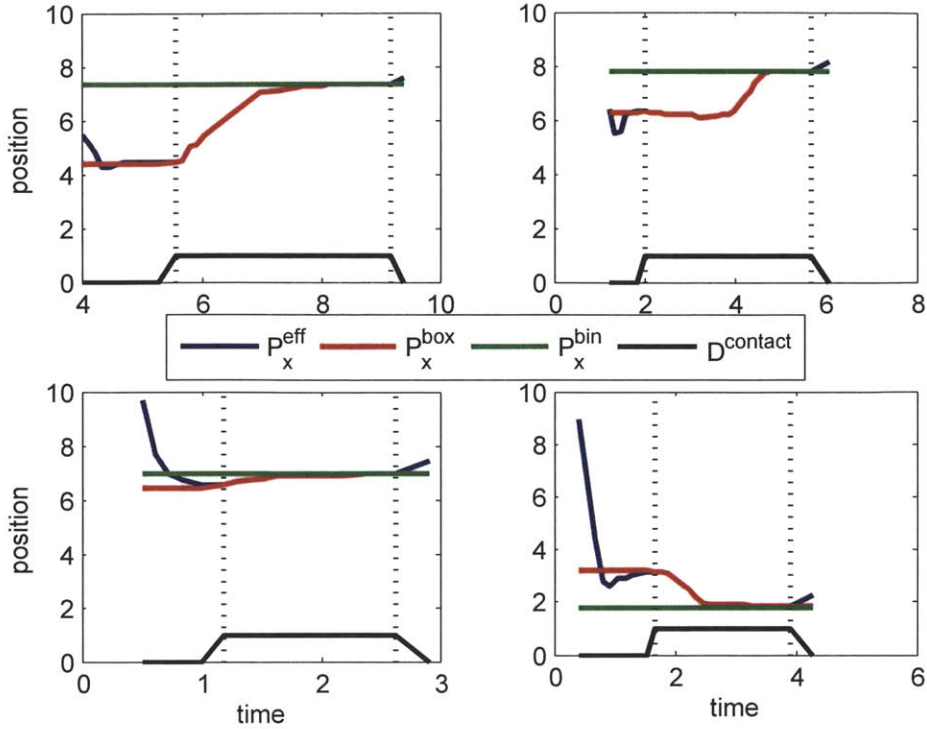


Figure 4-2: Four training examples of  $\mathbf{P}_x$  position data at time steps when contact changes. Notice that across different training samples, the effector (blue) and box (red) positions match at the start of contact, and the effector (blue) and bin (green) positions match at the end of contact. These are the relevant features of the “move box to bin” motion.

- *relEffEnd*: Is the ending position or orientation of the robot end effector generally the same relative to certain points of interest across the demonstrations?

Since the continuous and discrete variables  $\mathbf{C}$  and  $\mathbf{D}$  (such as time and power on/off) are independent of the points of interest (such as objects) in an environment, they are not considered in the *relEffStart* and *relEffEnd* modes.

This particular set of candidate motion variables was chosen based on commonalities observed among many practical robot manipulation tasks. A more thorough study of other possible modes may be warranted for other task domains. The learning approach described here remains applicable for additional types of motion variables.

After identifying all of the candidate motion variables, the algorithm uses clustering to determine if patterns exist across different training samples, such as those shown in Figure 4-2, for each candidate  $\mathbf{X} \in \{\mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q}\}$  across the different modes.

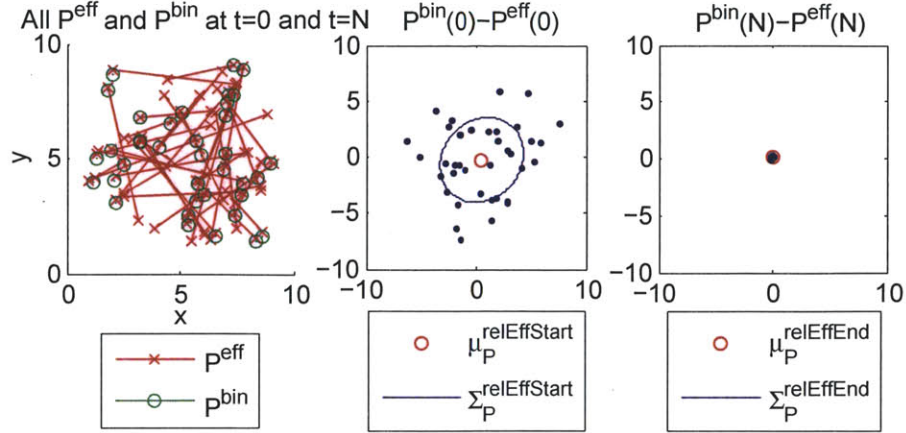


Figure 4-3: Clustering identifies motion variables for a 2-D “move object to bin” motion. Left: robot effector and bin locations at the start and end of each demonstration. Middle: values of  $(\mathbf{P}^{bin}(0) - \mathbf{P}^{eff}(0))$  to determine if *relEffStart* is a relevant motion variable for position—the large spread indicates low relevance. Right: values of  $(\mathbf{P}^{bin}(N) - \mathbf{P}^{eff}(N))$  to determine if *relEffEnd* is a relevant motion variable for position—the narrow spread indicates that this variable is relevant to this motion.

A narrow spread in the values of a motion variable across many training samples indicates that the variable is relevant to the motion. Algorithm 4.2 describes the details of the approach.

The approach determines which of the parameters listed above are statistically similar over the different training sequences by fitting a Gaussian  $\langle \boldsymbol{\mu}, \boldsymbol{\Sigma} \rangle$  for each parameter at the start and end of all the trials. Resulting Gaussians with very narrow spread compared to the range of the motion, i.e.

$$\max(\text{eig}(\boldsymbol{\Sigma}_{\mathbf{X}}^{mode})) < \epsilon \max(\text{range}(\mathbf{X}^{mode})),$$

indicate that the motion variable in question is relevant. Figure 4-3 visualizes the process of determining the relevance of the variables *relEffStart* and *relEffEnd* for position during a “move object to bin” motion. Judging by the large spread in  $\mathbf{P}^{bin}(0) - \mathbf{P}^{eff}(0)$  values, one can conclude that *relEffStart* is not a relevant feature of the candidate variable  $\mathbf{P}$ , or that the relative positions between the bin and effector does not matter at the start of the motion. On the other hand,  $\mathbf{P}^{bin}(N) - \mathbf{P}^{eff}(N)$  values have a very narrow spread, indicating that *relEffEnd* is relevant for  $\mathbf{P}$ , or that

---

**Algorithm 4.2 IDENTIFYMOTIONVARS ( $\mathcal{S}$ )**


---

**Input:**
 $\mathcal{S}$ , set of  $K$  demonstrated sequences  $\{T_k\}_{k=1..K}$ , where  $T = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q} \rangle$ 
**Output:**
 $\mathcal{F}$ , tuple of features  $\langle F_{\mathbf{C}}, F_{\mathbf{D}}, F_{\mathbf{P}}, F_{\mathbf{Q}} \rangle$ , where each  $F = \langle \boldsymbol{\mu}, \boldsymbol{\Sigma}, \text{relevant} \rangle$ 
**Notable local variables:**
 $\mathcal{M} = \{absStart, absEnd, relInit, relEffStart, relEffEnd\}$ 
 $\mathbf{q}_k^{\vec{b}\vec{1}} = \text{TOQUATERNION}(\mathbf{P}_k^b - \mathbf{P}_k^{eff})$ , orientation of ray from object  $b$  to robot end effector ( $eff = \text{index } 1$ ) for trial  $k$ 
 $\epsilon$ , small ratio to determine relevant motion variables, e.g., 0.01

$$\Delta(\mathbf{X}_1, \mathbf{X}_2) = \begin{cases} \mathbf{X}_1^{-1} \mathbf{X}_2 & \text{if } \mathbf{X} = \mathbf{Q} \\ \mathbf{X}_2 - \mathbf{X}_1 & \text{otherwise} \end{cases}$$

- 1: **for**  $\mathbf{X} \in \{\mathbf{C}, \mathbf{D}, \mathbf{P}^{eff}, \mathbf{Q}^{eff}\}$  **do**
  - 2:  $\langle \boldsymbol{\mu}, \boldsymbol{\Sigma} \rangle_{\mathbf{X}} \left\{ \begin{array}{l} absStart \\ absEnd \\ relInit \end{array} \right\} \leftarrow \text{FITGAUSS} \left\{ \begin{array}{l} \{\mathbf{X}_k(0)\}_{k=1..K} \\ \{\mathbf{X}_k(N_k)\}_{k=1..K} \\ \{\Delta(\mathbf{X}_k(0), \mathbf{X}_k(N_k))\}_{k=1..K} \end{array} \right\}$
  - 3: **end for**
  - 4:  $\langle \boldsymbol{\mu}, \boldsymbol{\Sigma} \rangle_{\mathbf{P}} \left\{ \begin{array}{l} relEffStart \\ relEffEnd \end{array} \right\} \leftarrow \text{FITGAUSS} \left\{ \begin{array}{l} \left\{ \mathbf{P}_k^b(0) - \mathbf{P}_k^{eff}(0) \right\}_{k=1..K, b=2..B} \\ \left\{ \mathbf{P}_k^b(N_k) - \mathbf{P}_k^{eff}(N_k) \right\}_{k=1..K} \end{array} \right\}$
  - 5:  $\langle \boldsymbol{\mu}, \boldsymbol{\Sigma} \rangle_{\mathbf{Q}} \left\{ \begin{array}{l} relEffStart \\ relEffEnd \end{array} \right\} \leftarrow \text{FITGAUSS} \left\{ \begin{array}{l} \left\{ (\mathbf{Q}_k^b(0))^{-1} \mathbf{q}_k^{\vec{b}\vec{1}}(0) \right\}_{k=1..K, b=2..B} \\ \left\{ (\mathbf{Q}_k^b(N_k))^{-1} \mathbf{q}_k^{\vec{b}\vec{1}}(N_k) \right\}_{k=1..K} \end{array} \right\}$
  - 6: **for**  $\mathbf{X} \in \{\mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q}\}$ ,  $mode \in \mathcal{M}$ , and  $b = 1 \dots B$  as applicable **do**
  - 7:  $\text{relevant}_{\mathbf{X}}^{mode} = 0$
  - 8: **if**  $\max(\text{eig}(\boldsymbol{\Sigma}_{\mathbf{X}}^{mode,b})) < \epsilon \max(\text{range}(\mathbf{X}^{mode,b}))$  **then**
  - 9:  $\text{relevant}_{\mathbf{X}}^{mode,b} = 1$
  - 10: **end if**
  - 11:  $F_{\mathbf{X}} = \langle \boldsymbol{\mu}, \boldsymbol{\Sigma}, \text{relevant} \rangle_{\mathbf{X}}^{\forall mode \in \mathcal{M}}$
  - 12: **end for**
-

the relative position of the effector to the bin at the end of the motion is consistently similar across demonstrations.

## 4.2 Data Processing and Flow Tube Generation

After identifying the motion variables, the system knows which points of interest in the environment are relevant to the motion in the new situation. The next step is to process the training data into a format with which the system can create the probabilistic flow tube. Algorithm 4.3 describes this process.

The algorithm can be summarized intuitively in the following steps, with numbers corresponding to the illustrations in Figure 4-4:

1. *Collect training data*: Training data includes continuous, discrete, position, and orientation information. In Figure 4-4, there are three recorded training sequences.
2. *Identify the relevant motion variables*: Using the motion variable identification algorithm, the system determines the relevant features of the demonstrated motion. In the “move box to bin example,” the locations of the box and bin are identified as relevant, while the ball location is not.
3. *Extract relevant states in new environment* (line 1): During autonomous execution of the demonstrated task in the new environment, the system can use sensing to determine the new environment states. This step simply extracts those states that are relevant based on the identified motion variables. In the example, the system will examine the new scene and extract the locations of the box and bin for the next steps.
4. *Gather similarly initialized demonstrations* (line 2): Identify a subset of the demonstrated data sequences that have a relevant initial environment most similar to the new situation. In the example, the system will select a subset of demonstrations in which the relative initial positions of the box and bin are most similar to those in the new situation.

---

**Algorithm 4.3** MAKEPFT ( $\mathcal{S}, \mathcal{F}, T(0)$ )

---

**Input:**

$\mathcal{S}$ , set of  $K$  demonstrated sequences  $\{T_k\}_{k=1..K}$ ,  
where  $T = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q} \rangle$   
 $\mathcal{F}$ , tuple of features  $\langle F_{\mathbf{C}}, F_{\mathbf{D}}, F_{\mathbf{P}}, F_{\mathbf{Q}} \rangle$   
 $T(0)$ , a new environment state

**Output:**

$T^{eff} = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}^{eff}, \mathbf{Q}^{eff} \rangle$ , robot trajectory  
 $\Sigma^{eff}$ , covariances at each corresponding time step

**Notable local variables:**

$\mathbf{w}$ , temporal matching indexes for two trajectories

```
1:  $T'(0) \leftarrow \text{EXTRACTRELEVANTOBJECTS}(T(0), \mathcal{F})$ 
2:  $\mathcal{S}' \subseteq \mathcal{S} \leftarrow \text{SIMILARINITCONDSEQS}(\mathcal{S}, T'(0))$ 
3:  $\mathcal{S}^{norm} \leftarrow \text{NORMALIZESCALEROTATE}(\mathcal{S}', \mathcal{F}, T'(0))$ 
4:  $T^{eff} = T_1^{norm}$ , where  $T_k^{norm} \in \mathcal{S}^{norm}$ 
5: for  $k = 2$  to  $K$  do
6:    $\mathbf{w} \leftarrow \text{FASTDTW}([P, Q]^{eff}, [P, Q]_k^{norm})$ 
7:    $T^{eff} \leftarrow \frac{1}{k} [(k-1)T_{eff}(\mathbf{w}_{:,1}) + T_k^{norm}(\mathbf{w}_{:,2})]$ 
8: end for
9: for  $k = 1$  to  $K$  do
10:   $\mathbf{w} \leftarrow \text{FASTDTW}([P, Q]^{eff}, [P, Q]_k^{norm})$ 
11:   $T_k^{interp} \leftarrow \text{INTERPOLATE}(T_k^{norm}(\mathbf{w}_{:,2}), |T^{eff}|)$ 
12: end for
13: for  $n = 1$  to  $|T^{eff}|$  do
14:   $\Sigma^{eff}(n) \leftarrow \text{COVARIANCE}\{T_k^{interp}(n) : k = 1..K\}$ 
15: end for
```

---

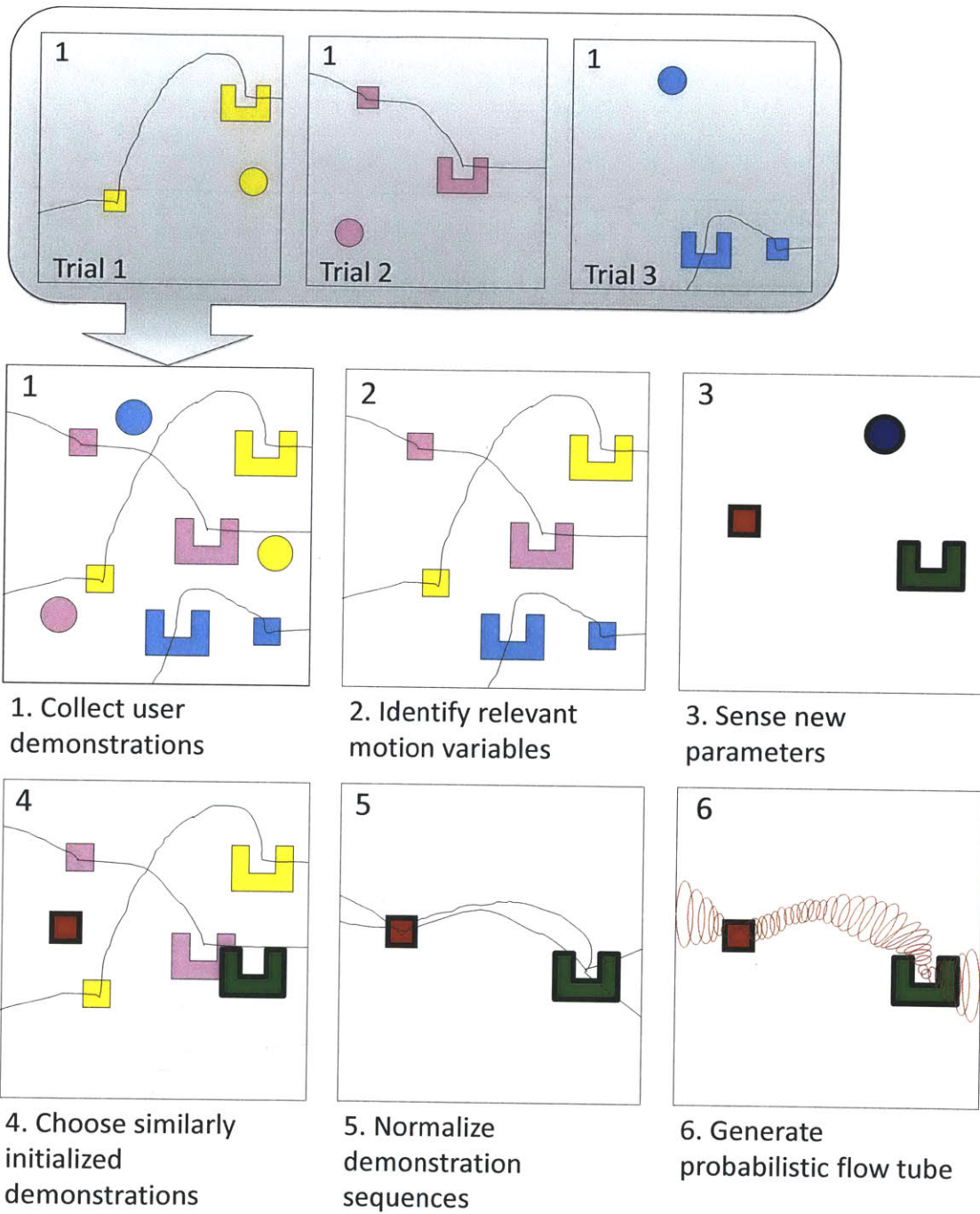


Figure 4-4: Illustrated steps of the approach with three demonstrations of the “move the box to the bin” task in the two-dimensional simulation environment

5. *Normalize demonstration sequences* (line 3): Normalize the selected subset of demonstrated data sequences to fit the values of the motion variables for the new situation. In the example, the system will scale, translate, and rotate the data sequences so that the robot end effector location upon initial contact with the box matches the box location in the new situation, and the effector location upon releasing contact with the box matches the bin location in the new situation.

The steps during normalization in *position* is shown on the left of Figure 4-5: Suppose there exists a demonstrated sequence shown in black, and the current environment state desires the motion to start from point A and end at point B. First, the sequence is scaled to the proper size. Next, the sequence is translated to match the start with the desired start location. Finally, the sequence is rotated so the end matches the desired end location. The steps during normalization in *orientation* is shown on the right of Figure 4-5: Suppose a demonstrated sequence has initial orientations depicted as black arrows, and the current environment state desires the motion to start in the orientation marked in green and end in the orientation marked in red. First, all orientations throughout the motion are rotated by an amount equal to the offset between the start orientation and the desired start orientation. Next, incremental rotations are computed using spherical linear interpolation (slerp) from the resulting end orientation and the desired end orientation. Finally, the incremental rotations are applied to the orientations along the trajectory such that the start orientation is rotated by zero amount and remains at the desired start orientation, and the end orientation is rotated by the full offset to end up at the desired end orientation.

6. *Generate flow tube* (lines 4-15): Temporally match all the space-normalized sequences, and create a probabilistic flow tube to be used for autonomous execution. This is accomplished through the use of dynamic time warping, which is discussed next.

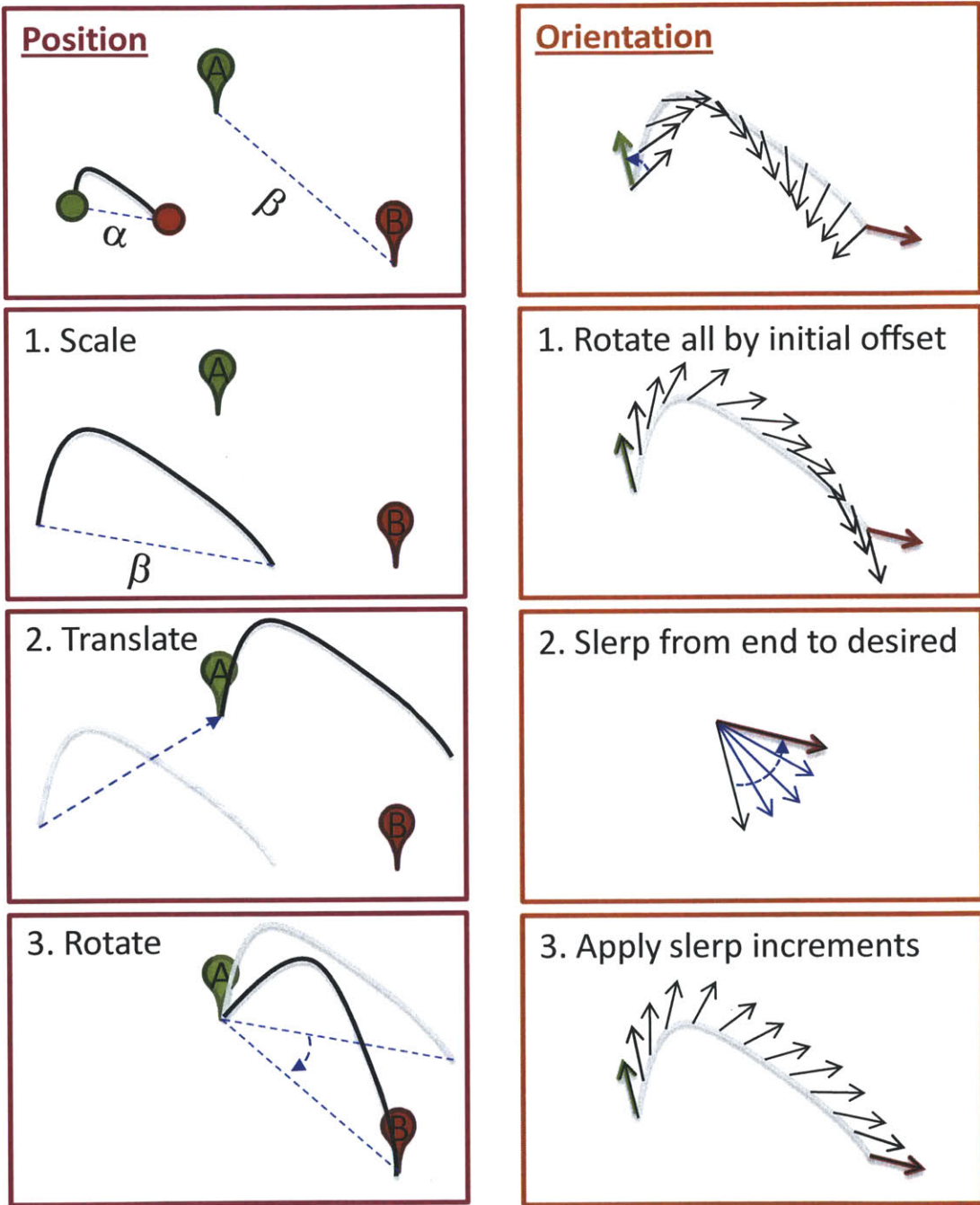


Figure 4-5: Illustration of how normalization works for position and orientation variables.



The algorithm uses dynamic time warping (DTW) [44, 61] to temporally match observed sequences. Intuitively, dynamic time warping temporally deforms two sequences to minimize the overall difference between them. The basic algorithm takes two recorded state sequences  $\mathbf{R} = [R_1, R_2 \dots, R_m]^T$  and  $\mathbf{S} = [S_1, S_2 \dots, S_n]^T$ , and creates an  $m \times n$  local cost matrix with entries containing the pairwise distances between all the data points in both sequences,  $c_{ij} = |R_i - S_j|$ , where  $i \in \{1 \dots m\}$  and  $j \in \{1 \dots n\}$ .

Any temporal matching of the two sequences corresponds to a traversal of the cost matrix from the element matching the origin of the two sequences,  $c_{11}$ , to the opposite corner,  $c_{mn}$ . Thus the problem of finding the best temporal matching reduces to finding the traversal of the cost matrix that results in the least total cost. Dynamic programming is employed to find this optimal matching by computing the minimal cumulative cost:

$$C_{i,j} = \begin{cases} \infty, & \text{if } i = 0 \text{ or } j = 0 \\ c_{ij}, & \text{if } i, j = 1 \\ c_{ij} + \min \begin{cases} C_{i-1,j-1} \\ C_{i-1,j} \\ C_{i,j-1} \end{cases}, & \text{otherwise} \end{cases}$$

The minimal cumulative cost at the last entry,  $C_{m,n}$ , is the minimal total cost, and the path taken to achieve it reflects the best matching between the two sequences. If the two sequences are very similar, the traversal of the cost matrix will be near diagonal. This optimal matching is represented as a  $p \times 2$  matrix  $\mathbf{w}$  containing the indices of  $\mathbf{R}$  and  $\mathbf{S}$  such that  $\mathbf{R}(\mathbf{w}_{i,1})$  is aligned with  $\mathbf{S}(\mathbf{w}_{i,2})$ , where  $p \geq m, n$  is the number of elements along the matched path.

My implementation uses a fast DTW algorithm developed by Salvador and Chan [58], which uses a projection algorithm to recursively refine a solution from coarse resolution approximations, resulting in comparable accuracy performance but linear time and space complexity.

Using dynamic time warping, the algorithm can determine the mean of two tra-

jectories  $\mathbf{R}$  and  $\mathbf{S}$  as  $\frac{1}{2}(\mathbf{R}(\mathbf{w}_{i,1}) + \mathbf{S}(\mathbf{w}_{i,2}))$ . Referring back to the set of normalized demonstrated sequences  $\mathcal{S}^{norm}$  in Algorithm 4.3, the approach iteratively computes a representative mean sequence using this procedure (lines 4-8). This is the output trajectory sequence of the robot end effector  $T^{eff}$ . For orientation variables, the algorithm takes the arithmetic mean of the quaternions. The result is very close to the true quaternion mean if the collection of observations are clustered near each other. The true quaternion mean can be used instead, but is more computationally intensive [50].

The demonstrated sequences may have different numbers of data entries, so fast dynamic time warping is used again to temporally match each of the normalized demonstrated sequences in  $\mathcal{S}^{norm}$  to the mean sequence  $T^{eff}$ , and interpolate them so that all have the same number of data entries (lines 9-12). Finally, the algorithm computes covariances at each corresponding time step across the temporally matched normalized demonstrated sequences (lines 13-15).

### 4.3 Pre-learning

To enable faster performance online, the approach can pre-learn a large set of probabilistic flow tubes randomly initialized at different environment states offline. These pre-learned PFTs are stored so that during online recognition, the PFT with environment states most similar to the online environment is selected and normalized to the online environment. The pre-learning process is described in Algorithm 4.4. The inputs to pre-learning include a set of demonstrated sequences  $\mathcal{S}$ , a task label  $\ell$ , and a set of initial environment states  $\mathbf{T0}$ , potentially randomly generated. The essence of pre-learning involves calling `OFFLINEACTIVITYLEARNING` for each environment state in  $\mathbf{T0}$ , and then storing all of the pre-learned PFTs in the library entry for activity  $\ell$ .

Once PFTs have been pre-learned for many activities in the library, new activity PFTs can be easily generated given a new environment state by selecting the stored PFT that has the most similar initial environment state. The process of generating

---

**Algorithm 4.4** PRELEARNPFTs ( $\mathcal{S}, \ell, \mathbf{T0}$ )

---

**Input:** $\mathcal{S}$ , set of  $K$  demonstrated sequences  $\{T_k\}_{k=1..K}$ , where  $T = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q} \rangle$  $\ell$ , the label of the task $\mathbf{T0}$ , set of initial environment states from where to prelearn PFTs**Output:** $lib_\ell = \langle \mathbf{PFT}^{\mathbf{E}}, \mathbf{E}, \mathcal{F}, q \rangle_\ell$ , library entry containinga set of prelearned PFTs ( $\mathbf{PFT}$ ),a set of corresponding initial environment states ( $\mathbf{E}$ ),relevant motion characteristics ( $\mathcal{F}$ ),and subtasks ( $q$ )1: **for**  $T0 \in \mathbf{T0}$  **do**2:    $\langle PFT_\ell^{T0}, \mathcal{F} \rangle \leftarrow \text{OFFLINEACTIVITYLEARNING}(\mathcal{S}, \ell, T0)$ 3: **end for**4:  $lib_\ell \leftarrow \langle \{PFT_\ell^{T0}\}_{T0 \in \mathbf{T0}}, \mathbf{T0}, \mathcal{F}, \emptyset \rangle$ 

---

a PFT for a new environment state given pre-learned flow tubes is described in Algorithm 4.5. The inputs include the library of learned tasks  $\mathcal{L}$ , where pre-learned PFTs are stored, and a new environment state  $T(0)$ . For each entry in the library, GETPFTSFROMHERE first determines the relevant components of the initial environment state using each library entry's identified motion variables. It next selects among the pre-learned PFTs in the library entry the one PFT that has the most similar relevant initial environment state components as the current situation. Finally, the selected PFT is normalized through scaling, translation, and rotation in position and spherical interpolation in orientation (as depicted in Figure 4-5) to fit the current environment state.

The idea of pre-learning PFTs is made useful by the following property: two environment states that correspond to the same set of  $k$  similarly initialized demonstrations will produce PFTs that are equivalent after normalization. Figure 4-6 illustrates this property using an example: Suppose there exists some set of training demonstrations, of which the five shown in blue are most similar in initial environment state to the environment state depicted by  $p$ . There may exist other training sequences not depicted in the illustration. Suppose also that among all training sequences, the five with most similar initial environment states to that of  $c$  also happen to be the same

---

**Algorithm 4.5** GETPFTSFROMHERE ( $\mathcal{L}, T(0)$ )
 

---

**Input:**

- $\mathcal{L}$ , library of previously learned task entries  $\{lib_\ell\}_{\ell \in L}$   
 where each entry  $lib_\ell = \langle \mathbf{PFT}, \mathbf{E}, \mathcal{F}, q \rangle_\ell$
- $T(0)$ , a new environment state

**Output:**

- $\mathbf{PFT}_L^{T(0)}$ , set of probabilistic flow tubes for different labels starting from  $T(0)$

- 1: **for**  $\ell \in L$  **do**
  - 2:  $T_\ell(0) \leftarrow \text{EXTRACTRELEVANTOBJECTS}(T(0), \mathcal{F}_\ell)$
  - 3:  $PFT_\ell^{E \in \mathbf{E}} \leftarrow \text{SIMILARINITCONDSEQ}(\mathbf{PFT}_\ell^E, T_\ell(0))$
  - 4:  $PFT_\ell^{T(0)} \leftarrow \text{NORMALIZESCALEROTATE}(PFT_\ell^E, \mathcal{F}_\ell, T_\ell(0))$
  - 5: **end for**
  - 6:  $\mathbf{PFT}_L^{T(0)} \leftarrow \{PFT_\ell^{T(0)}\}_{\ell \in L}$
- 

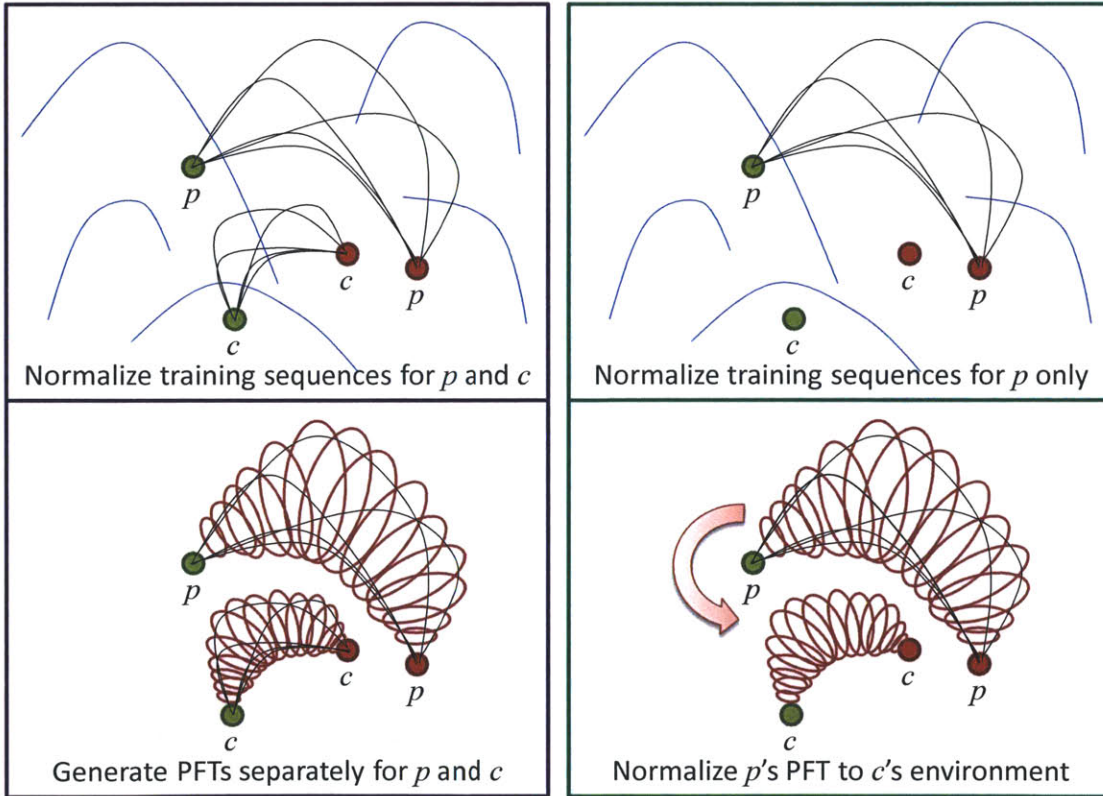


Figure 4-6: For two sets of environment states  $p$  (“prelearned”) and  $c$  (“current”) that correspond to the same set of five training sequences shown in blue, generating the PFT for  $c$  directly from training sequences (left) is equivalent to generating the PFT for  $c$  from normalizing the PFT for  $p$  (right).

five blue trajectories as before. Given the training set, one can generate a probabilistic flow tube for the environment state given by  $c$  using the `OFFLINEACTIVITYLEARNING` algorithm. This option, shown on the left side of Figure 4-6, requires performing dynamic time warping twice along the entire sequence of the trajectory, which takes on the order of 0.5 seconds to run on an Intel Core i7 processor for trajectories on the order of 30 data points. Alternatively, one can pre-learn the PFT associated with environment state  $p$  (by calling `PRELEARNPFTs` offline), and normalize its flow tube to the environment state given by  $c$  (by calling `GETPFTsFROMHERE`). This option is shown on the right side of Figure 4-6. Generating a PFT from pre-learned PFTs takes on the order of 0.01 seconds, and is thus far more desirable when it is necessary to generate a PFT on the fly, such as when preceding online recognition.

The property described above is only useful if there are enough pre-learned PFTs to provide good coverage of the environment states. Without specific domain knowledge of the tasks of interest, one can safely assume that randomly generating environment states from which to pre-learn PFTs will give good coverage as the number of pre-learned PFTs goes toward infinity. To achieve reasonably good coverage of the state space in the implementation, I typically use a large number of randomly generated initial environment states (e.g. 100 to 500) from whence to pre-learn PFTs. Since the pre-learning process is performed offline in advance and storing a large number of PFTs in memory is not a problem for modern machines, a generous number of PFTs is chosen to be pre-learned for good performance. For specific applications, domain knowledge may be used to intelligently select the set of environment states from which to generate pre-learned PFTs.

## 4.4 Enabling Autonomous Execution

The probabilistic flow tubes generated by the learning algorithm can be used to perform autonomous execution in new environment states. In a simple implementation, the mean trajectory of a PFT can be down sampled and sent to a controller as waypoints to follow. This approach assumes the existence of an inverse kinematics solver

on the robot to generate appropriate robot poses that move the end effector to the target waypoint positions in Cartesian space. A more sophisticated implementation can incorporate the covariances as cost functions for deviations from the nominal trajectory. This approach is presented as a possible future extension in Section 8.1.1.

# Chapter 5

## Recognizing Motions Online

After a library of probabilistic flow tube models of all the different motions  $\ell \in L$  from the current environment state  $T^{curr}(0)$  is learned by calling either

$$\langle \mathbf{PFT}_L^{T^{curr}(0)}, \mathcal{F} \rangle \leftarrow \text{OFFLINEACTIVITYLEARNING}(\mathcal{T}, L, T^{curr}(0))$$

or

$$\mathbf{PFT}_L^{T^{curr}(0)} \leftarrow \text{GETPFTSFROMHERE}(\mathcal{L}, T^{curr}(0))$$

if the motion was pre-learned, the system is ready to perform real-time recognition of a user's executed motion online.

The real-time motion recognition approach is summarized in Algorithm 5.1. The inputs include the data structure  $\mathbf{PFT}$  obtained from activity model learning, the set of motion labels  $L$ , a current execution trajectory  $T^{curr}$  containing  $N^{curr}$  data points, and a set of utility parameters  $\mathcal{W}$  that is incrementally updated every time the current execution proceeds. When  $\text{ONLINERECOGNITION}$  is called for the first time, i.e. an operator is starting a new motion to be recognized,  $\mathcal{W}$  is initially empty. As the execution progresses, the algorithm reuses parameters in  $\mathcal{W}$  as much as possible to avoid redundant computations and enable real-time recognition.

The three main components of the recognition approach are: determine where in each probabilistic flow tube the current partial motion might correspond, temporally align the identified portion of the PFT with the current partial motion, and compute

---

**Algorithm 5.1** ONLINERECOGNITION  $\left(\mathbf{PFT}_L^{T^{curr}(0)}, L, T^{curr}, \mathcal{W}\right)$

---

**Input:**

$\mathbf{PFT}_L^{T^{curr}(0)} = \{PFT_\ell\}_{\ell \in L}$ , where  $PFT_\ell = \langle T^{eff}, \Sigma^{eff}, \mathbf{I}, \mathbf{G} \rangle$ , and

$T^{eff} = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}^{eff}, \mathbf{Q}^{eff} \rangle$

$\Sigma^{eff} = \langle \sigma_{\mathbf{C}}, \sigma_{\mathbf{D}}, \Sigma_{\mathbf{P}}^{eff}, \Sigma_{\mathbf{Q}}^{eff} \rangle$

$L$ , set of labels of all learned motions

$T^{curr}$ , current observed trajectory

$\mathcal{W}$ , cost matrices for dynamic time warping

**Output:**

$LL = \{ll_\ell\}_{\ell \in L}$ , set of log likelihoods for each motion in  $L$

$\mathcal{W}$ , updated cost matrices for DTW

**Notable local variables:**

$\mathbf{C}^{time}, \sigma_{\mathbf{C}}^{time}$ , temporal component of  $\mathbf{C}, \sigma_{\mathbf{C}}$  in  $PFT$

$\mathbf{d}$ , spatial distance between a PFT and current state

$\mathbf{p}$ , probability densities evaluated at time steps in PFT

$N_\ell$ , length of flow tube  $\ell$

$N^{curr}$ , length of current trajectory  $T^{curr}$

$t^{curr}$ , time at current position  $T^{curr}(N^{curr})$

$\mathbf{w}$ , temporal matching indexes for two trajectories

$\pi_\ell$ , prior log likelihood of flow tube  $\ell$

- 1: **for**  $\ell \in L$  **do**
  - 2:    $\mathbf{d} = \left\{ \left\| [\mathbf{P}, \mathbf{Q}]^{curr}(N^{curr}) - [\mathbf{P}, \mathbf{Q}]_\ell^{eff}(n) \right\| \right\}_{n=1 \dots N_\ell}$
  - 3:    $\mathbf{p} = \left\{ \mathcal{N}(\mathbf{C}_\ell^{time}(n), \sigma_{\mathbf{C}_\ell}^{time}(n)) \Big|_{t^{curr}} \right\}_{n=1 \dots N_\ell}$
  - 4:    $\mathbf{p} \leftarrow \frac{\mathbf{p}}{\max(\mathbf{p})}$
  - 5:    $\mathbf{d}' = \left\{ \frac{\mathbf{d}_n}{\mathbf{p}_n} \right\}_{n=1 \dots N_\ell}$
  - 6:    $n^* = \operatorname{argmin}_n(\mathbf{d}')$
  - 7:    $PFT_\ell^* = PFT_\ell(1 \dots n^*)$
  - 8:    $\langle \mathbf{w}, \mathcal{W}_\ell \rangle \leftarrow \text{INCREMENTDTW}(T_\ell^{eff*}, T^{curr}, \mathcal{W}_\ell)$
  - 9:    $ll_\ell = \pi_\ell + \frac{1}{|\mathbf{w}|} \sum_{j=1}^{|\mathbf{w}|} \left( \mathbf{G}_\ell(\mathbf{w}_{j,1}) - \frac{1}{2} \boldsymbol{\delta}^T \mathbf{I}_\ell(\mathbf{w}_{j,1}) \boldsymbol{\delta} \right)$   
       where  $\boldsymbol{\delta} = T^{curr}(\mathbf{w}_{j,2}) - T_\ell^{eff}(\mathbf{w}_{j,1})$
  - 10: **end for**
-



the log likelihood that the current partial motion is recognized as each PFT. I now discuss each component in more detail.

## 5.1 Partial Flow Tube Matching

To determine the location in a probabilistic flow tube that best corresponds to the current executed state (lines 2 to 6 in Algorithm 5.1), the approach looks at both how far the current executed state is from the PFT, and how much time has passed in the execution as compared with the trained models. Intuitively, the point in the PFT that best corresponds to the current executed state should have a small spatial distance to it while having been executed at around the same time.

The algorithm first computes the distances from the current position and orientation in  $T^{curr} (N^{curr})$  to those in the nominal trajectory  $T^{eff}$  for motion  $\ell$  through all the time steps in the PFT, or

$$\mathbf{d} = \left\{ \left\| [\mathbf{P}, \mathbf{Q}]^{curr} (N^{curr}) - [\mathbf{P}, \mathbf{Q}]_{\ell}^{eff} (n) \right\| \right\}_{n=1 \dots N_{\ell}}.$$

Small values in  $\mathbf{d}$  may help indicate which time steps in the PFT correspond to the current executed state.

Next, the algorithm represents how temporally different the current execution time  $t^{curr}$  is from the points in the temporal component of the PFT, or  $\langle \mathbf{C}^{time}, \boldsymbol{\sigma}_{\mathbf{C}}^{time} \rangle$ , by evaluating the probability density of the current time at each point in the tube, or

$$\mathbf{p} = \left\{ \mathcal{N} (\mathbf{C}_{\ell}^{time} (n), \boldsymbol{\sigma}_{\mathbf{C}_{\ell}}^{time} (n)) \Big|_{t^{curr}} \right\}_{n=1 \dots N_{\ell}}.$$

The distances are weighted by the temporal similarity measure to obtain

$$\mathbf{d}' = \left\{ \frac{\mathbf{d}_n}{\mathbf{p}_n} \right\}_{n=1 \dots N_{\ell}}.$$

The time step in the PFT that corresponds to the current executed state occurs when the weighted distance is smallest.

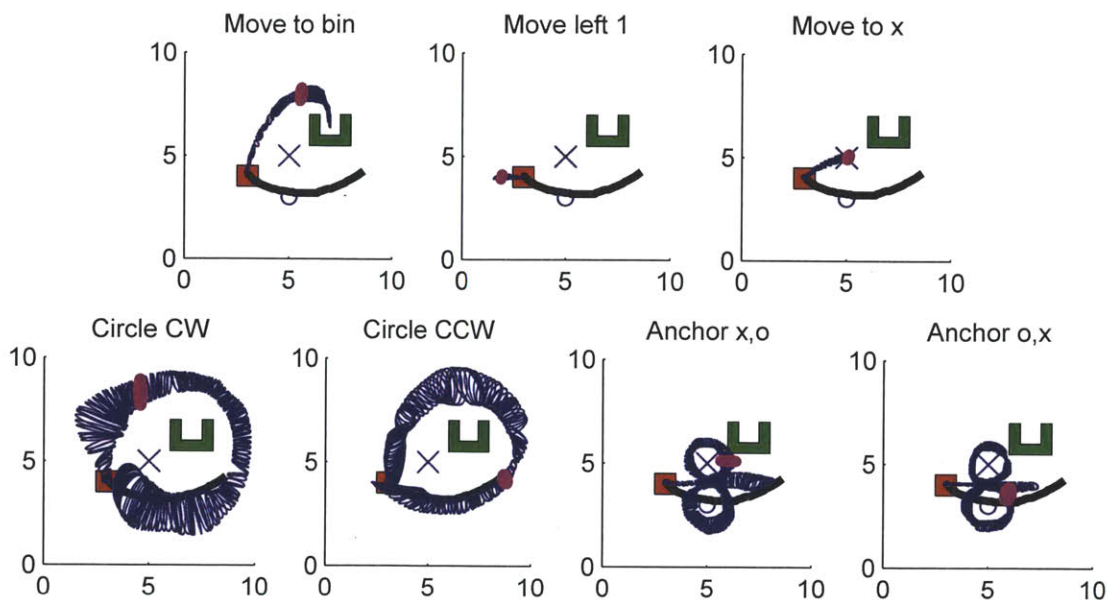


Figure 5-1: An example partial test motion (black) is compared to each learned PFT (blue) in a 2D environment initially with a red box, green bin, and stationary locations  $x$  and  $o$  as shown. The magenta marking on each PFT indicates the spot on the PFT that best matches the current execution (rightmost end of black motion) as determined by lines 2 to 6 in Algorithm 5.1.

I choose to use the actual distances between the current state and the points in the nominal PFT trajectory (i.e.,  $\|T^{curr}(N^{curr}) - T^{eff}(n)\|$ ) to represent spatial consistency instead of computing the spatial probability densities of the current state evaluated through all the Gaussians in the PFT (i.e.,  $\mathcal{N}(T^{eff}(n) \Sigma^{eff}(n))|_{T^{curr}(N^{curr})}$ ) because of the real-time recognition requirement. While spatial probability densities give a more accurate estimate of a point’s deviation from the flow tube, they take longer to compute due to the higher dimensionality of spatial states. I have found direct distances to be good estimates of spatial consistency for motions with flow tube widths that do not vary greatly with high frequency, as is true in most robotic tasks.

Figure 5-1 shows the result of determining the best correspondence points in 7 different learned PFTs for a partially executed test motion. The data is collected in a 2D world consisting of two movable objects “box” and “bin” and two stationary locations  $x$  and  $o$ . The trained motions are “move box to bin,” “move box left 1 unit,” “move box to point  $x$ ,” “circle the box around bin in clockwise direction,”

“circle the box around bin in counter-clockwise direction,” “make an anchor loop (or ‘figure 8’), first around  $x$ , then around  $o$ ,” and “make an anchor loop, first around  $o$ , then around  $x$ .” The highlighted magenta markings indicate the positions in each PFT that was determined to best correspond to the current position of the partial test motion in black. Notice that the identified position in the “circle clockwise” PFT is not spatially near the current test position, but rather in a more reasonable position that is temporally consistent with the current execution.

## 5.2 Temporal Alignment of Partial Motion

The next step in the recognition approach is to temporally align the identified portion of each PFT to the current execution trajectory in order to compute likelihoods. While the algorithm could use basic dynamic time warping to perform temporal matching in the recognition problem, it is undesirable that the algorithm would recompute similar cost matrices each time the test motion progresses.

To leverage as much information as possible from one time step to the next, the algorithm performs incremental dynamic time warping [17] by keeping previously computed cost and back pointer matrices in memory and updating as necessary.

The INCREMENTALDTW algorithm used in line 8 of ONLINERECOGNITION accepts two input trajectories  $\mathbf{R}$  and  $\mathbf{S}$  of lengths  $M$  and  $N$ , respectively, and an input parameter  $\mathcal{W} = \langle \mathbf{c}, \mathbf{C}, \mathbf{b} \rangle$ , where  $\mathbf{c}$ ,  $\mathbf{C}$ , and  $\mathbf{b}$  are the  $m \times n$  cost matrix, cumulative cost matrix, and back pointer matrix, respectively, computed at the previous time step. Incremental DTW reuses these matrices when temporally aligning  $\mathbf{R}$  and  $\mathbf{S}$  by computing new values only for rows  $m + 1$  to  $M$ , and columns  $n + 1$  to  $N$ . A new temporal matching is found by tracing the updated back pointer matrix from the end to the beginning, as shown in Figure 5-2. The temporal alignment is represented as a two column matrix  $\mathbf{w}$  of corresponding indexes of the two input trajectories.

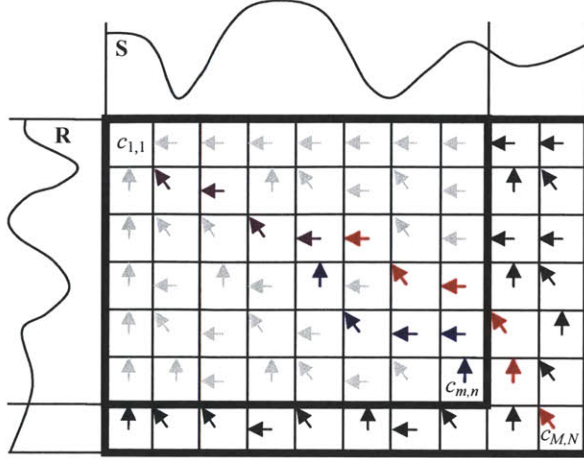


Figure 5-2: During incremental dynamic time warping, the algorithm only needs to update the cost and back pointer matrices from the previously stored values (depicted by the inner box) to obtain a new temporal matching (red and purple path).

### 5.3 Compute Log Likelihoods

Finally, after temporally matching the current partial trajectory with the corresponding portions of each probabilistic flow tube, the algorithm can proceed to compute the likelihood that the current trajectory belongs to a particular modeled motion. Formally, I define random variables  $\mathcal{L}$  and  $\mathcal{O}$  to represent motion labels and observations, respectively. The probability that the motion is  $\ell$  given the current observation sequence  $T^{curr}$  is  $p_{\mathcal{L}|\mathcal{O}}(\ell|T^{curr})$ , and applying Bayes Rule gives  $p_{\mathcal{L}|\mathcal{O}}(\ell|T^{curr}) \propto p_{\mathcal{O}|\mathcal{L}}(T^{curr}|\ell) p_{\mathcal{L}}(\ell)$ . The prior probability of each motion label  $p_{\mathcal{L}}(\ell)$  is obtained by recording the number of times the motion was used during training weighted by how far along the current motion is in the flow tube, i.e.  $p_{\mathcal{L}}(\ell) = \frac{\#_{\ell} n_{\ell}^*}{\#_{all} N_{\ell}}$ . The problem that remains is computing the likelihood of observing the current trajectory given a particular flow tube.

I model the observation likelihood  $p_{\mathcal{O}|\mathcal{L}}(T^{curr}|\ell)$  as the product of the probability densities of each spatial distribution in the flow tube evaluated at the temporally aligned points in the current trajectory, i.e.,

$$\prod_{j=1}^{|\mathbf{w}|} \left[ \mathcal{N} \left( T_{\ell}^{eff}(\mathbf{w}_{j,1}), \Sigma_{\ell}^{eff}(\mathbf{w}_{j,1}) \right) \Big|_{t^{curr}(\mathbf{w}_{j,2})} \right]^{\frac{1}{|\mathbf{w}|}}.$$

Since the length of the temporal matching matrix  $|\mathbf{w}|$  can vary between  $\max(M, N)$  and  $M + N - 1$ , I use the exponent  $(\frac{1}{|\mathbf{w}|})$  to perform a multiplicative renormalization over the resampled points to ensure that the overall contribution of the probability values for the trajectory is not inflated by the resampling process.

Taking the log of the observation probability gives

$$\frac{1}{|\mathbf{w}|} \sum_{j=1}^{|\mathbf{w}|} \left[ -\log \left( (2\pi)^{\frac{d}{2}} \left| \Sigma_{\ell(\mathbf{w}_{j,1})}^{eff} \right|^{\frac{1}{2}} \right) - \frac{1}{2} \boldsymbol{\delta}^T \Sigma_{\ell(\mathbf{w}_{j,1})}^{eff}{}^{-1} \boldsymbol{\delta} \right]$$

where  $d = \dim(T^{eff})$  and  $\boldsymbol{\delta} = T^{curr}(\mathbf{w}_{j,2}) - T_{\ell}^{eff}(\mathbf{w}_{j,1})$ . Finally, the approach uses the pre-computed inverse covariances  $\mathbf{I}$  and probability density coefficients  $\mathbf{G}$  obtained from model learning to efficiently compute the posterior log likelihood as shown in line 9 of Algorithm 5.1,

$$\pi_{\ell} + \frac{1}{|\mathbf{w}|} \sum_{j=1}^{|\mathbf{w}|} \left[ \mathbf{G}_{\ell}(\mathbf{w}_{j,1}) - \frac{1}{2} \boldsymbol{\delta}^T \mathbf{I}_{\ell}(\mathbf{w}_{j,1}) \boldsymbol{\delta} \right],$$

where  $\pi_{\ell} = \log(p_{\mathcal{L}}(\ell))$  is the log of the prior on motion label  $\ell$ .

In the case that the trajectory does not belong to any of the motion labels in the library, the computed log likelihoods of all labels will be very small in value. I define the log likelihood that the motion is “unknown” to be

$$\tau - \max_{\ell} (LL),$$

where  $\tau$  is a user-generated threshold, and is set to -1000 in experiments. In practice, “unknown” is treated as an additional entry in the task library with its computed log likelihood value. During recognition, the most likely recognized activity can be any of the activities in the library, including “unknown.”



# Chapter 6

## Learning High-level Plans

The two challenges when extending learning and recognition to high-level tasks consisting of activity sequences are (1) determining where activities should be segmented in a sequence and (2) identifying and learning previously unseen activities embedded in a sequence. Existing research typically assume that either sequences are already pre-segmented into discrete action labels, or no segmentation information is provided and unsupervised learning is required to cluster portions of motion trajectories into activities.

In practical applications of teleoperation or kinesthetic teaching, users may naturally provide segmentation information to a subset of the training demonstrations. Akgun et al. [2] describe an intuitive kinesthetic teaching method that enables users to easily provide important points called *keyframes* during a training demonstration through a voice commanding interface. Kinesthetic teaching involves a user physically moving a robot's end effector around in the environment. The goal of Akgun et al. is to provide easier and more intuitive ways for users to generate demonstrations during teaching. In their experiments, users could choose to provide only keyframes and not worry about the trajectories between keyframes, relying on a planner to autonomously proceed from one keyframe to another during autonomous execution. I borrow their idea of keyframes and use them as segmentation points overlaid on the demonstrated trajectories that describe the behavior of the activities within the segments. Akgun et al. show that trajectory demonstrations without providing keyframes require less

time for the user [2], so I assume that of a set of  $K$  demonstrations of a particular task plan, only a small subset contain keyframes.

As shown in the setup of Algorithm 6.1, the inputs of plan learning consist of a set of demonstrations  $\mathcal{S}$  of the sequence of activities that compose the plan, a new label  $\ell'$  that is used to describe the plan, a new environment state  $T(0)$ , and the current library of learned tasks  $\mathcal{L}$ . Of the demonstrations, a subset  $\mathcal{Y}$  contain keyframes recorded as an additional discrete variable in the demonstrations. My plan learning approach as illustrated in Figure 6-1 achieves the following: (1) determines the sequence of activities that compose the demonstrated plan if it is a new plan, (2) learns any previously unseen activities in the plan, (3) auto-segments non-keyframe trials by bootstrapping on keyframe trials, and (4) generates a new probabilistic flow tube for the entire plan in the new environment state. The resulting output of plan learning is an updated task library with the new learned plan and any additional previously unknown activities.

The plan learning approach also allows for and encourages an interactive mode that actively probes the user for validation of the bootstrapped auto-segmentation if the system is below a certain level of confidence. In practice this feature is natural and unencumbering for the user as it mimics a student asking a teacher for validation when performing a task, which is common in human teacher-student interactions. More discussion of the user validation capability can be found in Section 6.3.

An overview of the plan learning approach is shown in Algorithm 6.1. The algorithm first checks whether the input plan label  $\ell'$  already exists in the set of library task labels  $L$  (line 1), and if so, updates the entry in the library  $lib_{\ell'}$  with a new plan PFT from the input environment state  $T(0)$  (line 2). More steps are involved if the input demonstrations refer to a new label not already in the library. I now discuss each major step in more detail.



---

**Algorithm 6.1** OFFLINEPLANLEARNING ( $\mathcal{S}, \ell', T(0), \mathcal{L}$ )

---

**Input:**

- $\mathcal{S}$ , set of  $K$  demonstrated sequences  $\{T_k\}_{k=1..K}$  for a particular plan, where  $T = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q} \rangle$
- $\ell'$ , a new plan label
- $T(0)$ , a new environment state
- $\mathcal{L}$ , library of previously learned task entries  $\{lib_\ell\}_{\ell \in L}$ , where each entry  $lib_\ell = \langle \mathbf{PFT}, \mathbf{E}, \mathcal{F}, q \rangle$  has a label ( $\ell$ ), a set of pre-learned PFTs ( $\mathbf{PFT}$ ), a set of corresponding initial environment states ( $\mathbf{E}$ ), relevant motion characteristics ( $\mathcal{F}$ ), and a list of subtasks ( $q$ )

**Output:**

- $\mathcal{L}$ , updated task library

**Notable local variables:**

- $\mathcal{Y} \subseteq \mathcal{S}$ , trials with user-provided segmentation keyframes
- $\mathcal{Z} = \mathcal{S} \setminus \mathcal{Y}$ , trials without user-provided segmentation keyframes
- $D^{key} \in \mathbf{D}$ , discrete keyframe variables in  $\mathcal{Y}$  trials
- $K^{key} = \{D^{key} == 1\}$ , indexes of discrete keyframes in  $\mathcal{Y}$  trials
- $\gamma$ , confidence in bootstrapped auto-segmentation

```
1: if  $\ell' \in L$  then
2:    $lib_{\ell'} \leftarrow \text{GENERATEPLANPFT}(\ell', q_{\ell'}, \mathcal{L}, T(0))$ 
3: else
4:    $\mathbf{LL} \leftarrow \text{RECOGNIZETASKSUSINGKEYFRAMETRIALS}(\mathcal{Y}, \mathcal{L})$ 
5:    $q \leftarrow \text{VOTEACTIVITYSEQUENCE}(\mathbf{LL})$ 
6:   while  $\mathcal{Z} \neq \emptyset$  do
7:      $\langle \mathcal{L}', q' \rangle \leftarrow \text{LEARNUNKNOWNACTIVITIESINPLAN}(\mathcal{Y}, \mathcal{L}, q, \{T_k(0)\}_{\forall T_k \in \mathcal{Z}})$ 
8:     for  $j = 1 \dots |\mathcal{Z}|$  do
9:        $K_j^{key} \leftarrow \text{INITIALIZEPROPORTIONATELYINTIME}(\mathcal{Z}_j, \mathcal{Y})$ 
10:       $\langle K_j^{key}, \gamma_j, u \rangle \leftarrow \text{UPDATEUSINGMOTIONVARS}(\mathcal{Z}_j, q', \{\mathcal{F}_\ell\}_{\ell \in L'}, K_j^{key})$ 
11:      if ! $u$  then
12:         $\langle K_j^{key}, \gamma_j \rangle \leftarrow \text{UPDATEUSINGRECOGOPTIMIZATION}(\mathcal{Z}_j, q', \mathcal{L}', K_j^{key})$ 
13:      end if
14:    end for
15:     $\langle \mathcal{Y}, \mathcal{Z} \rangle \leftarrow \text{VALIDATEAUTOSEGMENTATIONS}(\mathbf{K}^{key}, \gamma)$ 
16:  end while
17:   $\mathbf{T0} \leftarrow \text{RANDOMENVIRONMENTSTATES}()$ 
18:   $\langle \mathcal{L}, q' \rangle \leftarrow \text{LEARNUNKNOWNACTIVITIESINPLAN}(\mathcal{Y}, \mathcal{L}, q, \mathbf{T0})$ 
19:   $lib_{\ell'} \leftarrow \text{GENERATEPLANPFT}(\ell', q', \mathcal{L}, T(0))$ 
20:   $\mathcal{L} \leftarrow \{\mathcal{L}, lib_{\ell'}\}$ 
21: end if
```

---

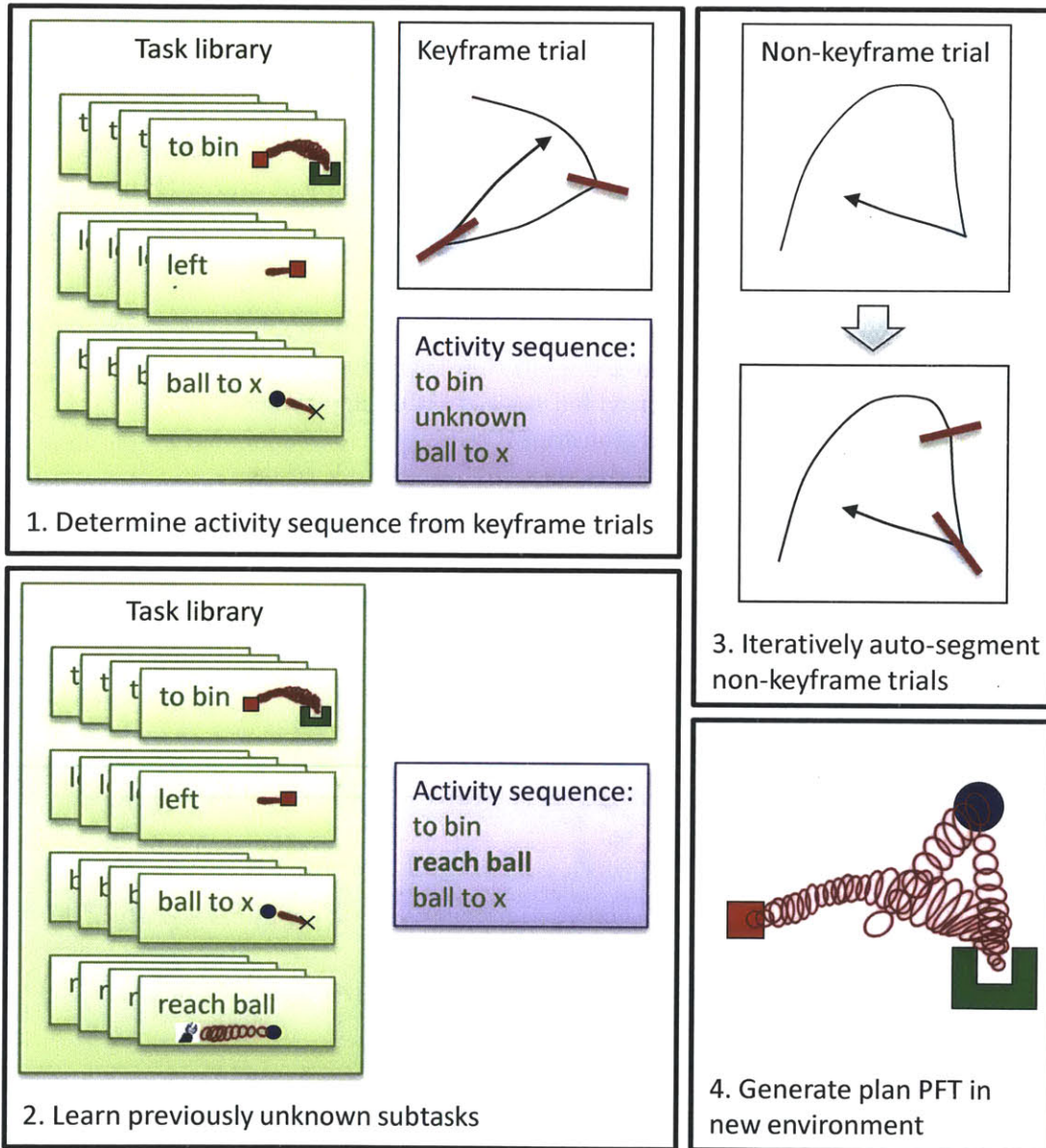


Figure 6-1: Summary of plan learning approach

## 6.1 Determining Activity Sequence

First, to determine the sequence of activities that compose the demonstrated plan, the algorithm performs recognition on the keyframe training sequences (line 4 in Algorithm 6.1), which are the pre-segmented demonstrations of the plan. `RECOGNIZETASKSUSINGKEYFRAMETRIALS` (Algorithm 6.2) is a wrapper function that calls `ONLINERECOGNITION` for all the segments in the keyframe trials  $\mathcal{Y}$ . Its inputs include the set of keyframe trials  $\mathcal{Y}$  and the task library  $\mathcal{L}$ , and it outputs a set of log likelihoods through time  $\mathbf{LL}$  that reflects which label each activity segment describes. For each trial in  $\mathcal{Y}$ , the algorithm walks through each time step  $t$  in the trial (lines 1 and 3). If the time step marks the beginning of a new activity segment because it is a keyframe for trial  $i$  ( $t \in K_i^{key}$ ), then the algorithm makes a note of the environment state at that time step  $\mathcal{Y}_i(t)$ , and generates PFTs from that environment state using the pre-learned PFTs in the library (line 7). Once the PFTs for an activity segment are obtained, it can call `ONLINERECOGNITION` on each partial segment  $\mathcal{Y}_i(t_{prev} \dots t)$  in the activity (lines 10 and 11) until the next keyframe is reached. In addition to the generated PFTs from the current environment state, the set of motion labels, and the current partial execution, `ONLINERECOGNITION` also requires a set of parameters  $\mathcal{W}$  for incremental dynamic time warping, which is initialized to empty in the beginning of each activity segment (line 8). For each time step in a trial, `ONLINERECOGNITION` outputs a set of log likelihoods  $LL_i(t) = \{LL_\ell\}_{\ell \in L}$  that reflect how likely the current segment  $\mathcal{Y}_i(t_{prev} \dots t)$  belongs to each activity label  $\ell$  in the library.

An example illustration of the log likelihoods is shown in Figure 6-2. With time progressing horizontally across, suppose there are two keyframe trials represented by the two blue lines (simplified down to one dimension for illustration purposes) with keyframes indicated by red markers. The goal of the `RECOGNIZETASKSUSINGKEYFRAMETRIALS` algorithm is to generate the log likelihoods depicted as green lines, so that after a bit of processing, the system can derive the most likely sequence of activity labels that is represented by the keyframe trials. Judging by the illustrated log likelihoods in the example, one might eventually conclude that the activity

---

**Algorithm 6.2** RECOGNIZETASKSUSINGKEYFRAMETRIALS ( $\mathcal{Y}, \mathcal{L}$ )

---

**Input:**

- $\mathcal{Y}$ , set of keyframe trials
- $\mathcal{L}$ , library of previously learned task entries  $\{lib_\ell\}_{\ell \in L}$   
where each entry  $lib_\ell = \langle \mathbf{PFT}, \mathbf{E}, \mathcal{F}, q \rangle$

**Output:**

- $\mathbf{LL}$ , set of log likelihoods

**Notable local variables:**

- $D^{key} \in \mathbf{D}$ , discrete keyframe variables in  $\mathcal{Y}$  trials
- $K^{key} = \{D^{key} == 1\}$ , indexes of discrete keyframes in  $\mathcal{Y}$  trials

```
1: for  $i = 1 \dots |\mathcal{Y}|$  do
2:    $t_{prev} \leftarrow 0$ 
3:   for  $t = 0 \dots |\mathcal{Y}_i|$  do
4:     if  $t \in K_i^{key}$  then
5:        $t_{prev} \leftarrow t$ 
6:        $T(0) \leftarrow \mathcal{Y}_i(t_{prev})$ 
7:        $\{PFT_\ell\}_{\ell \in L} \leftarrow \text{GETPFTSFROMHERE}(\mathcal{L}, T(0))$ 
8:        $\mathcal{W} \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$ 
9:     end if
10:     $T^{curr} \leftarrow \mathcal{Y}_i(t_{prev} \dots t)$ 
11:     $\langle LL_i(t), \mathcal{W} \rangle \leftarrow \text{ONLINERECOGNITION}(\{PFT_\ell\}_{\ell \in L}, L, T^{curr}, \mathcal{W})$ 
12:  end for
13: end for
14:  $\mathbf{LL} \leftarrow \{LL_i\}_{i=1 \dots |\mathcal{Y}|}$ 
```

---

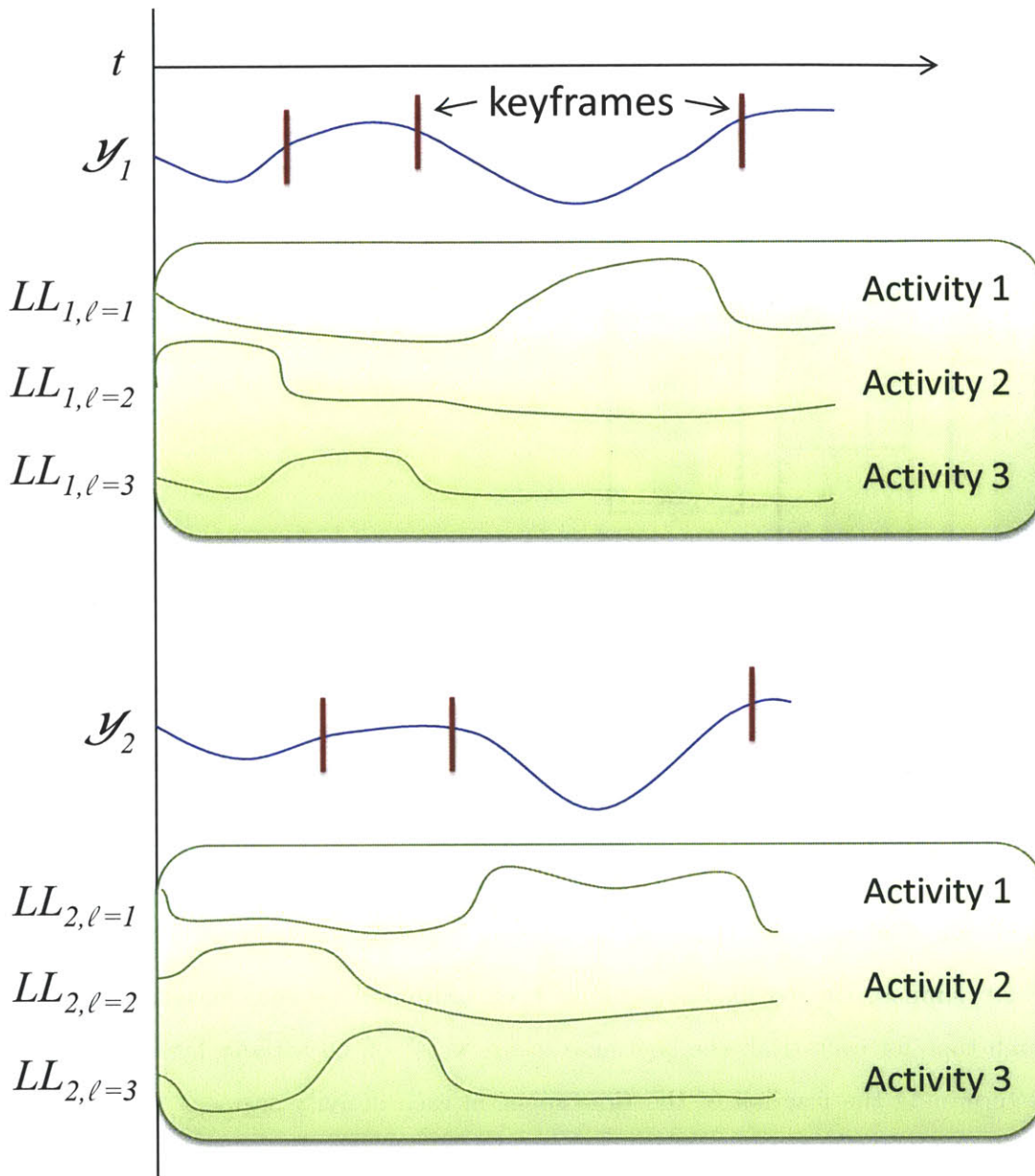


Figure 6-2: Illustration of performing online recognition on task sequences. In the two keyframe trials  $\mathcal{Y}_1$  and  $\mathcal{Y}_2$ , the log likelihood of activity 2 is highest during the first segment, activity 3 is highest during the second segment, and activity 1 is highest during the third segment. Therefore, it is likely that the activity sequence in this plan is  $\{2, 3, 1\}$ .

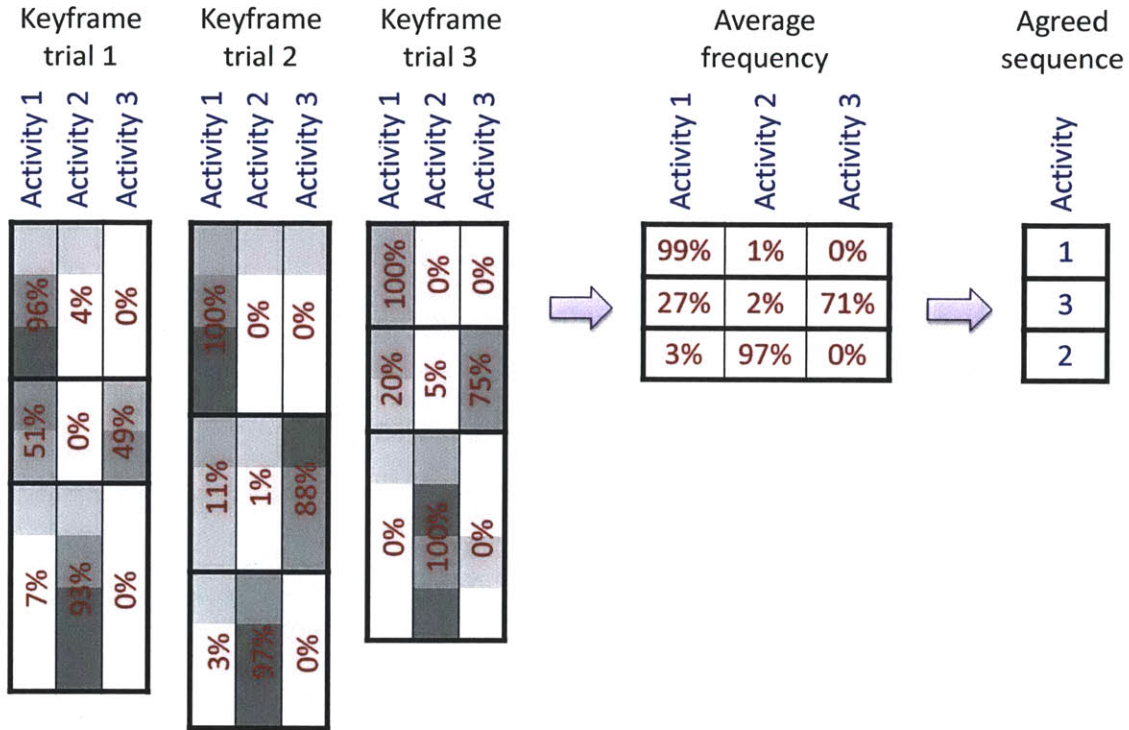


Figure 6-3: Example process of determining the sequence of activities that compose the demonstrated plan from keyframe trials. Shading represents recognized log likelihood, and red values represent the percentage of time steps during which a particular activity was recognized in a keyframe segment. The keyframe trials “vote” on an average recognition frequency for each segment and agree on a recognized activity sequence for the plan.

sequence for  $\mathcal{Y}$  is activity 2, followed by activity 3, followed by activity 1.

Next, suppose the log likelihoods have been computed for each activity segment through time for each trial, the keyframe trials “vote” on an activity label sequence by determining the fraction of the time spent in each activity segment recognizing each of the labels (Algorithm 6.1 line 5). The label with the most “votes” is deemed the recognized activity for that segment.

Figure 6-3 illustrates this process in an example: Suppose there are three keyframe trials, and time progresses downwards vertically. Keyframes are depicted as dark black lines that segment the trials, and the computed log likelihoods are represented as gray scale shading along the trials for activity labels 1, 2, and 3. Focusing on a single activity segment in a trial and comparing the likelihoods, the algorithm

identifies the percentage of time the recognizer labels the activity segment with a certain label (in this case, “1”, “2”, or “3”). For example, in the first segment of trial 1, the recognizer found that 96% of the time spent in that segment, activity 1 had the highest log likelihood among the three activity labels. If this pattern persists across different trials, then the system can conclude that the first segment is activity 1. Sometimes, a keyframe trial can be noisy, causing errors in recognition. These can be averaged out among the different trials. For example, the second segment of trial 1 is recognized as most likely activity 1, but after averaging the recognition frequencies from the other two trials, it becomes clear that the label of the second segment is most likely activity 3.

## 6.2 Learning Unknown Activities

The resulting activity sequence may contain activities in the task library or previously unknown activities. For every activity segment in the sequence that was recognized most likely as “unknown,” the training segments in the keyframe trials  $\mathcal{Y}$  are used to learn new PFTs in the task library for the previously unseen activities using the `LEARNUNKNOWNACTIVITIESINPLAN` algorithm. A temporary new library  $\mathcal{L}'$  is created, and a new activity sequence  $q'$  is generated from the new library. For example, a previously recognized activity sequence  $q$  might be “activity 2”, “unknown”, “activity 4”, “unknown”, and after calling `LEARNUNKNOWNACTIVITIESINPLAN`, the updated activity sequence  $q'$  might be {“activity 2”, “activity 5”, “activity 4”, “activity 6”}.

If all input demonstration trials are pre-segmented with keyframes, the algorithm directly pre-learns PFTs (using the methods presented in Section 4.3) for a large set of randomly generated environment states (lines 17-18 in Algorithm 6.1). Otherwise, an iterative process is used to determine segmentation for the non-keyframe trials, the first step of which is to generate PFTs starting at all the initial environment states of the non-keyframe trials (line 7 in Algorithm 6.1). More discussion about the auto-segmentation process is described in Section 6.3.

Details of the `LEARNUNKNOWNACTIVITIESINPLAN` function can be found in Al-

---

**Algorithm 6.3** LEARNUNKNOWNACTIVITIESINPLAN ( $\mathcal{S}, \mathcal{L}, q, \mathbf{T0}$ )

---

**Input:**

$\mathcal{S}$ , set of  $K$  pre-segmented demonstrated sequences  $\{T_k\}_{k=1..K}$  for a plan,  
where  $T = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q} \rangle$  and  
 $D^{key} \in \mathbf{D}$  contains segmentation information  
 $\mathcal{L}$ , library of previously learned task entries  $\{lib_\ell\}_{\ell \in L}$   
 $q$ , sequence of activities in plan, including unknown activities  
 $\mathbf{T0}$ , a set of initial environment states

**Output:**

$\mathcal{L}'$ , updated task library  
 $q'$ , updated sequence of activities in plan

**Notable local variables:**

$\{T_k^i\}_{k=1..K}$ , set of all  $i^{\text{th}}$  segments of trials in  $\mathcal{S}$

```
1:  $q' \leftarrow q$ 
2: for all  $i : q_i == 'unknown'$  do
3:    $q'_i \leftarrow \text{GETNEWLABEL}()$ 
4:    $lib_i \leftarrow \text{PRELEARNPFTS}(\{T_k^i\}_{k=1..K}, q'_i, \mathbf{T0})$ 
5: end for
6:  $\mathcal{L}' \leftarrow \left\{ \mathcal{L}, \{lib_i\}_{i:q_i == 'unknown'} \right\}$ 
```

---

gorithm 6.3. The inputs to the algorithm include a set of pre-segmented demonstrated sequences  $\mathcal{S}$  (when calling the function, the keyframe trials  $\mathcal{Y}$  can be given as input), the existing task library  $\mathcal{L}$  that will be updated, the activity sequence  $q$  determined by the algorithms in Section 6.1 that contains “unknown” activities, and a set of initial environment states  $\mathbf{T0}$ , possibly randomly generated, from which to pre-learn PFTs.

The algorithm first searches through the activity sequence in the plan for previously unknown activities (line 2), and generates a new label for each new activity by calling `GETNEWLABEL` (line 3). `GETNEWLABEL` is a simple function that queries the user for a new label on the activity, or if the system is run without user interaction, randomly generates a new string label by default. Next, the new activities are pre-learned using the initial environment states specified by the input (line 4), and the library is updated with the new entries (line 6).

When `LEARNUNKNOWNACTIVITIESINPLAN` is used in the context of learning a plan, it is first called using the keyframe trials that are provided as training input.



This works well if there are many such keyframe trials provided. However, in typical operations, because of the additional overhead of providing keyframes, I assume that the user only provides a few keyframe trials, while a greater number of trials do not have keyframes. In order to also take advantage of the non-keyframe trials, the `OFFLINEPLANLEARNING` algorithm iterates the process of learning previously unknown activities from keyframe trials and using the newly learned activities to aid in auto-segmentation, which converts more non-keyframe trials into keyframe trials. The details of auto-segmentation is discussed next.

### 6.3 Auto-segmentation

Since it is easier for a user to demonstrate entire plan trajectories without having to indicate where one activity stops and the next starts, the system auto-segment the activities in a plan for most trials from observing just a few fully pre-segmented trials. In this approach, auto-segmentation proceeds incrementally in lines 6 to 16 of Algorithm 6.1, in which the goal is to determine a set of key time frames  $K^{key}$  that correctly segments the corresponding set of previously unsegmented trials  $\mathcal{Z}$ .

For every non-keyframe trial in  $\mathcal{Z}$ , the algorithm first initializes its candidate keyframes proportionately in time based on average keyframe trials (line 9 in Algorithm 6.1) in order to provide a rough temporal estimate on where keyframes are located. For example, suppose a non-keyframe trial  $\mathcal{Z}_j$  of a plan containing three activities has 120 time steps, and there are three keyframe trials  $\mathcal{Y}$  with keyframes  $[5, 20]$ ,  $[30, 60]$ ,  $[25, 75]$  and total time steps 40, 80, and 100, respectively. The initial temporal estimate of the keyframes for trial  $z$  should be  $\frac{120}{3} \left[ \left( \frac{5}{40} + \frac{30}{80} + \frac{25}{100} \right), \left( \frac{20}{40} + \frac{60}{80} + \frac{75}{100} \right) \right]$ , or  $[30, 80]$ .

Next, the candidate keyframes are updated based on geometric information from the motion characteristics of each activity segment. As an illustrative example, consider the “move box to bin” motion that always ends at the same position relative to the bin (to within a small epsilon). This relevant motion variable reveals that a task sequence containing this activity can be reliably segmented at the point where

the effector position reaches the observed position relative to the bin.

Figure 6-4 illustrates an example of determining candidate segmentation points on a non-keyframe trial using information provided by a keyframe trial. In this example, there are three objects—a red box, a green bin, a blue ball—and a stationary location marked by  $x$ . The demonstrated task sequence in both trials is {“reach box”, “move box to bin”, “reach ball”, “move ball to  $x$ ”, “done with ball”}. The shaded regions in the keyframe trial (panel 1) depicts objects’ motions. Let’s assume the keyframes fall at time steps 20, 60, 70, and 80 of a 100 time step keyframe trajectory, as shown panel 3. Given a new non-keyframe trial as illustrated in panel 2, candidate keyframes are initially assigned at the same temporal proportions as in the keyframe trial (panel 4). These temporally assigned candidate keyframes typically give a good estimate of where the true keyframes lie.

Next, in panels 5 through 8, each candidate keyframe is updated based on the identified motion variables from the keyframe trial, following procedures in Algorithms 6.4 and 6.5. The search window for updating the  $i^{\text{th}}$  candidate keyframe ( $K_i$ ) is from keyframe  $K_{i-1}$  to  $K_{i+1}$ , where  $K_0$  and  $K_{|K|+1}$  are set to the beginning and end time points of the entire trial. Limiting the search space when updating the candidate keyframes based on spatial motion variables helps to prevent large temporal discrepancies caused by spatial reasoning alone. Such an error can occur when earlier and later activity segments in a task sequence revisit the same spatial feature—in this case, spatial reasoning alone may confuse the two segments, but limiting the temporal focus on the generated candidate keyframes can help reduce the confusion.

The function `UPDATEUSINGMOTIONVARS` handles both position and orientation motion variables, given in Algorithms 6.4 and 6.5, respectively. The inputs to each algorithm include an unsegmented plan trajectory  $T$ , the sequence of activities  $q$  that the trajectory is supposed to describe based on other keyframe trials, a set of learned motion characteristics  $\mathcal{F}$  for activities in the library, and the set of candidate keyframes  $K$  initialized by temporally matching keyframe trials. Each algorithm steps through the candidate keyframes and makes adjustments to the target end effector position  $\mathbf{P}^{*eff}$  or the target end effector orientation  $\mathbf{Q}^{*eff}$ , respectively, based on which

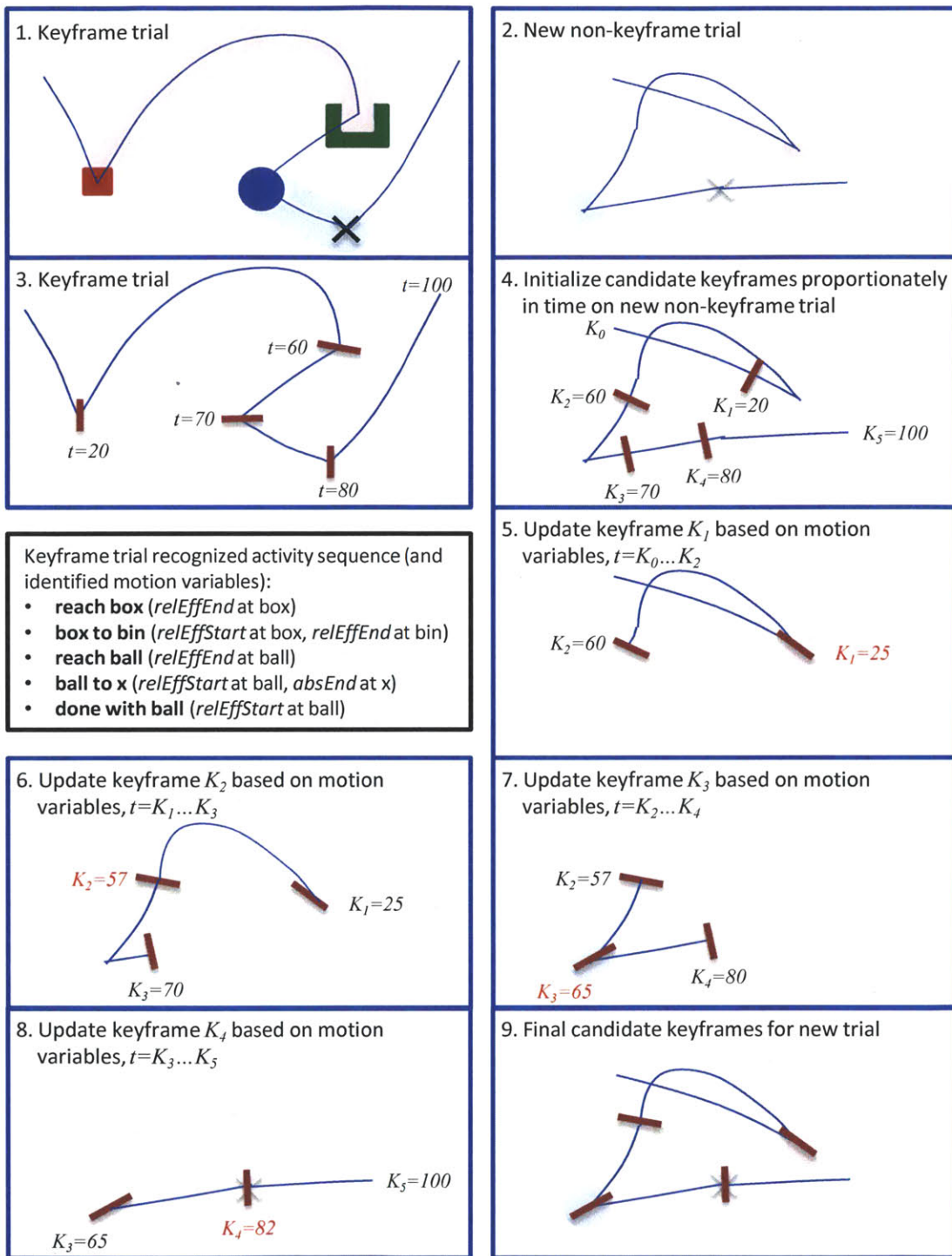


Figure 6-4: Illustration of generating candidate keyframes on a non-keyframe trial using time proportions in a keyframe trial, and updating candidate keyframes based on motion variables observed in keyframe trial.

motion variables are relevant for each activity  $q_i$  being considered.

For example, if any *relEffEnd* motion variables are relevant in position for a particular activity  $q_i$ , then  $\mu_{\mathbf{P}, \ell=q_i}^{relEffEnd, b}$  reflects what the mean difference between the end effector position and a relevant object  $b$ 's position should be at the end of activity  $q_i$ . In the training trajectory, an object  $b$ 's motion is recorded as  $\mathbf{P}^b(t)$ , and the end effector's motion is  $\mathbf{P}^{eff}(t)$ . When the difference in these two trajectories closely matches what the mean indicates it should be, then that time step is the point that marks the end of activity  $q_i$ , and where the keyframe should be placed. Figure 6-5 illustrates this in an example: three (faded) training samples of a “move to bin” activity suggest that for a new bin location (in dark green), a “move to bin” activity should end at the location marked by the asterisk. Given an executed plan trajectory (in dark blue) that contains the “move to bin” activity followed by some other activity, the algorithm selects the point in the trajectory that is closest to the target end location (red) to mark the segmentation between the two activities. This type of analysis is performed for each mode (*absStart*, *absEnd*, *relInit*, *relEffStart*, *relEffEnd*) in both position (Algorithm 6.4) and orientation (Algorithm 6.5).

For a motion with multiple identified motion variables, the variables are prioritized in the following order:

1. *relEffEnd*
2. *relInit*
3. *absEnd*
4. *relEffStart*
5. *absStart*

For example, let's say a “move box to bin” motion is trained in environments where the bin is always positioned at the stationary point  $x$ , then both *relEffEnd* at bin and *absEnd* at  $x$  would be relevant motion variables. In this case, the higher priority motion variable (*relEffEnd* at bin) is used for geometrically determining trajectory segmentation. In other words, if in a new trial the bin were placed at a different location from  $x$ , the activity's trajectory should end up at the bin location, not at

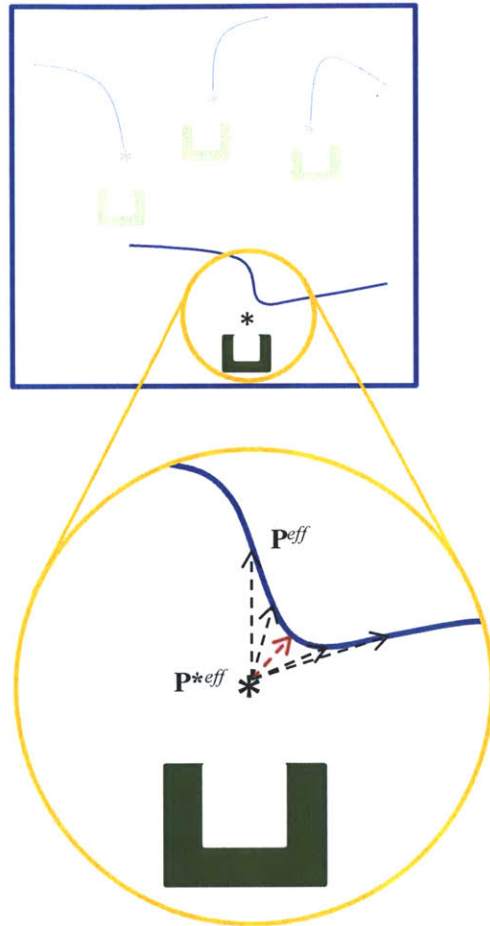


Figure 6-5: Illustration of how a keyframe is selected based on motion variable geometry

$x$ , and `UPDATEUSINGPOSITIONMOTIONVARS` would look for the point in the plan trajectory closest to the bin location to place the segmentation keyframe.

How closely the position or orientation of the end effector can match the geometric targets described by the motion variables is an indicator of how reliable the keyframe placement is. For example, if a position mode is relevant, a keyframe is placed at the time step  $t$  when the difference between the end effector position and the target end effector position  $|\mathbf{P}^{eff}(t) - \mathbf{P}^{*eff}(t)|_{t=K_{i-1} \dots K_{i+1}}$  is minimum for the keyframe between activity  $q_i$  and  $q_{i+1}$ . But how much is too much difference? From the motion variables, the algorithm knows that the spread of the of the motion variable  $\Sigma$  is an indicator of how much variation the motion variable had during training. Although  $\Sigma$  was computed from different variables (effector versus object positions), by properties of linear transformation of Gaussian distributions, the system can evaluate the current end effector position difference against the Gaussian distribution centered at zero, with covariance  $\Sigma$  to reflect how good the keyframe placement is, as shown in line 21 of Algorithm 6.4. A normalization factor  $\alpha = \frac{1}{\mathcal{N}(0, \Sigma)_0}$  is applied to obtain values between 0 and 1. The confidence measure of the quality of keyframes is defined as the product of these “keyframe goodness” values across all candidate keyframes.

If certain candidate segmentation points have no associated relevant motion variables, then a gradient-free optimization method is used to determine the points of segmentation based on recognition likelihood of candidate subtasks (line 12 in Algorithm 6.1). Conceptually, the points of segmentation should occur when activity segments, or subtasks, adjacent to the segmentation points are recognized with highest likelihood as compared to those generated from any other set of candidate segmentation points. The optimization function is defined by the sum of the recognized log likelihood of each activity in the subtask sequence. My implementation employs the Nelder-Mead simplex algorithm as the optimizer [35]. Since this algorithm is a minimizer, the optimization function is the negative sum of the subtask log likelihoods. The Nelder-Mead simplex algorithm is gradient-free and is widely used for unconstrained optimization problems due to its ability to produce rapid initial decreases in function values.

---

**Algorithm 6.4** UPDATEUSINGPOSITIONMOTIONVARS ( $T, q, \mathcal{F}, K$ )

---

**Input:**

$T = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q} \rangle$ , a training trajectory for a plan  
 $q$ , sequence of activities in plan  
 $\mathcal{F} = \{\mathcal{F}_\ell\}_{\ell \in L}$ , motion characteristics of previously learned task entries in library, where each  $\mathcal{F} = \langle F_{\mathbf{C}}, F_{\mathbf{D}}, F_{\mathbf{P}}, F_{\mathbf{Q}} \rangle$ , and each  $F = \langle \boldsymbol{\mu}, \boldsymbol{\Sigma}, \text{relevant} \rangle$   
 $K$ , candidate keyframes

**Output:**

$K$ , updated keyframes  
 $\gamma$ , confidence in keyframes  
 $u$ , whether or not there exist relevant position motion variables

**Notable local variables:**

$N$ , number of time steps in  $T$   
 $b$ :  $\text{relevant}_{\mathbf{P}, \ell=q_i}^{\text{mode}, b} == 1$ , indexes of any relevant objects in subtask  $q_i$  for position motion variable  $\text{mode} \in \{ \text{absStart}, \text{absEnd}, \text{relInit}, \text{relEffStart}, \text{relEffEnd} \}$

```
1:  $K_0 \leftarrow 1$ 
2:  $K_{|K|+1} \leftarrow N$ 
3:  $u \leftarrow \text{true}$ 
4: for  $i = 1 \dots |K|$  do
5:   if  $\forall \text{relevant}_{\mathbf{P}, \ell=q_i}^{\text{relEffEnd}}$  then
6:      $\left\{ \mathbf{P}^{*eff}(t) \leftarrow \text{avg}_b \left( \left\{ \mathbf{P}^b(t) - \boldsymbol{\mu}_{\mathbf{P}, \ell=q_i}^{\text{relEffEnd}, b} \right\}_{\forall b} \right) \right\}_{t=K_{i-1} \dots K_{i+1}}$ 
7:   else if  $\text{relevant}_{\mathbf{P}, \ell=q_i}^{\text{relInit}}$  then
8:      $\left\{ \mathbf{P}^{*eff}(t) \right\}_{t=K_{i-1} \dots K_{i+1}} \leftarrow \mathbf{P}^{eff}(K_{i-1}) + \boldsymbol{\mu}_{\mathbf{P}, \ell=q_i}^{\text{relInit}}$ 
9:   else if  $\text{relevant}_{\mathbf{P}, \ell=q_i}^{\text{absEnd}}$  then
10:     $\left\{ \mathbf{P}^{*eff}(t) \right\}_{t=K_{i-1} \dots K_{i+1}} \leftarrow \boldsymbol{\mu}_{\mathbf{P}, \ell=q_i}^{\text{absEnd}}$ 
11:   else if  $\forall \text{relevant}_{\mathbf{P}, \ell=q_{i+1}}^{\text{relEffStart}}$  then
12:      $\left\{ \mathbf{P}^{*eff}(t) \leftarrow \text{avg}_b \left( \left\{ \mathbf{P}^b(t) - \boldsymbol{\mu}_{\mathbf{P}, \ell=q_{i+1}}^{\text{relEffStart}, b} \right\}_{\forall b} \right) \right\}_{t=K_{i-1} \dots K_{i+1}}$ 
13:   else if  $\text{relevant}_{\mathbf{P}, \ell=q_{i+1}}^{\text{absStart}}$  then
14:      $\left\{ \mathbf{P}^{*eff}(t) \right\}_{t=K_{i-1} \dots K_{i+1}} \leftarrow \boldsymbol{\mu}_{\mathbf{P}, \ell=q_{i+1}}^{\text{absStart}}$ 
15:   else
16:      $\left\{ \mathbf{P}^{*eff}(t) \right\}_{t=K_{i-1} \dots K_{i+1}} \leftarrow \mathbf{P}^{eff}(K_i)$ 
17:      $u \leftarrow \text{false}$ 
18:   end if
19:    $K_i \leftarrow \text{argmin}_t \left( \left| \mathbf{P}^{eff}(t) - \mathbf{P}^{*eff}(t) \right|_{t=K_{i-1} \dots K_{i+1}} \right)$ 
20: end for
21:  $\gamma \leftarrow \prod_{i=1}^{|K|} \left( \alpha \mathcal{N} \left( 0, \boldsymbol{\Sigma}_{\mathbf{P}, q_i}^{\text{mode}, b} \right) \Big|_{\min_t \left| \mathbf{P}^{eff}(t) - \mathbf{P}^{*eff}(t) \right|_{t=K_{i-1} \dots K_{i+1}}} \right)$ 
```

---

---

**Algorithm 6.5** UPDATEUSINGORIENTATIONMOTIONVARS  $(T, q, \mathcal{F}, K)$ 


---

**Input:**

- $T = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q} \rangle$ , a training trajectory for a plan
- $q$ , sequence of activities in plan
- $\mathcal{F} = \{\mathcal{F}_\ell\}_{\ell \in L}$ , motion characteristics of previously learned task entries in library, where each  $\mathcal{F} = \langle F_{\mathbf{C}}, F_{\mathbf{D}}, F_{\mathbf{P}}, F_{\mathbf{Q}} \rangle$ , and each  $F = \langle \boldsymbol{\mu}, \boldsymbol{\Sigma}, \text{relevant} \rangle$
- $K$ , candidate keyframes

**Output:**

- $K$ , updated keyframes
- $\gamma$ , confidence in keyframes
- $u$ , whether there exist relevant orientation motion variables

**Notable local variables:**

- $N$ , number of time steps in  $T$
- $b : \text{relevant}_{\mathbf{Q}, \ell=q_i}^{\text{mode}, b} == 1$ , indexes of any relevant objects in subtask  $q_i$  for orientation motion variable  $\text{mode} \in \{ \text{absStart}, \text{absEnd}, \text{relInit}, \text{relEffStart}, \text{relEffEnd} \}$
- $\mathbf{q}^{\vec{b}1} = \text{TOQUATERNION}(\mathbf{P}^b - \mathbf{P}^{\text{eff}})$ , orientation of ray from object  $b$  to robot end effector ( $\text{eff} = \text{index } 1$ ) for input trial  $T$

- 1:  $K_0 \leftarrow 1$
  - 2:  $K_{|K|+1} \leftarrow N$
  - 3:  $u \leftarrow \text{true}$
  - 4: **for**  $i = 1 \dots |K|$  **do**
  - 5:   **if**  $\forall \text{relevant}_{\mathbf{Q}, \ell=q_i}^{\text{relEffEnd}}$  **then**
  - 6:      $\left\{ \mathbf{Q}^{\text{eff}}(t) \leftarrow \text{avg}_b \left( \left\{ \mathbf{q}^{\vec{b}1}(t) \left( \boldsymbol{\mu}_{\mathbf{Q}, \ell=q_i}^{\text{relEffEnd}, b} \right)^{-1} \right\}_{\forall b} \right) \right\}_{t=K_{i-1} \dots K_{i+1}}$
  - 7:   **else if**  $\text{relevant}_{\mathbf{Q}, \ell=q_i}^{\text{relInit}}$  **then**
  - 8:      $\left\{ \mathbf{Q}^{\text{eff}}(t) \right\}_{t=K_{i-1} \dots K_{i+1}} \leftarrow \mathbf{Q}^{\text{eff}}(K_{i-1}) \boldsymbol{\mu}_{\mathbf{Q}, \ell=q_i}^{\text{relInit}}$
  - 9:   **else if**  $\text{relevant}_{\mathbf{Q}, \ell=q_i}^{\text{absEnd}}$  **then**
  - 10:      $\left\{ \mathbf{Q}^{\text{eff}}(t) \right\}_{t=K_{i-1} \dots K_{i+1}} \leftarrow \boldsymbol{\mu}_{\mathbf{Q}, \ell=q_i}^{\text{absEnd}}$
  - 11:   **else if**  $\forall \text{relevant}_{\mathbf{Q}, \ell=q_{i+1}}^{\text{relEffStart}}$  **then**
  - 12:      $\left\{ \mathbf{Q}^{\text{eff}}(t) \leftarrow \text{avg}_b \left( \left\{ \mathbf{q}^{\vec{b}1}(t) \left( \boldsymbol{\mu}_{\mathbf{Q}, \ell=q_{i+1}}^{\text{relEffStart}, b} \right)^{-1} \right\}_{\forall b} \right) \right\}_{t=K_{i-1} \dots K_{i+1}}$
  - 13:   **else if**  $\text{relevant}_{\mathbf{Q}, \ell=q_{i+1}}^{\text{absStart}}$  **then**
  - 14:      $\left\{ \mathbf{Q}^{\text{eff}}(t) \right\}_{t=K_{i-1} \dots K_{i+1}} \leftarrow \boldsymbol{\mu}_{\mathbf{Q}, \ell=q_{i+1}}^{\text{absStart}}$
  - 15:   **else**
  - 16:      $\left\{ \mathbf{Q}^{\text{eff}}(t) \right\}_{t=K_{i-1} \dots K_{i+1}} \leftarrow \mathbf{Q}^{\text{eff}}(K_i)$
  - 17:      $u \leftarrow \text{false}$
  - 18:   **end if**
  - 19:    $K_i \leftarrow \text{argmin}_t \left( \arccos \left| \mathbf{Q}^{\text{eff}}(t) \cdot \mathbf{Q}^{\text{eff}}(t) \right|_{t=K_{i-1} \dots K_{i+1}} \right)$
  - 20: **end for**
  - 21:  $\gamma \leftarrow \prod_{i=1}^{|K|} \left( \alpha \mathcal{N} \left( 0, \boldsymbol{\Sigma}_{\mathbf{Q}, q_i}^{\text{mode}, b} \right) \Big|_{\text{min}_t \left| \mathbf{Q}^{\text{eff}}(t) \cdot \mathbf{Q}^{\text{eff}}(t) \right|_{t=K_{i-1} \dots K_{i+1}}} \right)$
-



---

**Algorithm 6.6** UPDATEUSINGRECOGOPTIMIZATION ( $T, q, \mathcal{L}, K$ )

---

**Input:**

$T = \langle \mathbf{C}, \mathbf{D}, \mathbf{P}, \mathbf{Q} \rangle$ , a training trajectory for a plan  
 $q$ , sequence of activities in plan  
 $\mathcal{L}$ , library of previously learned task entries  $\{lib_\ell\}_{\ell \in L}$   
 $K$ , candidate keyframes

**Output:**

$K$ , updated keyframes  
 $\gamma$ , confidence in keyframes

**Notable local variables:**

$y$ , the same trial as  $T$ , but with keyframes  $K$  incorporated  
 $J$ , the optimization function  
 $\tau$ , log likelihood threshold, set at -1000, below which the trajectory being evaluated is considered not the current activity

```
1: loop
2:    $y \leftarrow \text{ADDKEYFRAMESTOTRIAL}(T, K)$ 
3:    $LL \leftarrow \text{RECOGNIZETASKSUSINGKEYFRAMETRIALS}(y, \mathcal{L})$ 
4:    $J := K \mapsto -\sum_t (LL_q)$ 
5:    $K \leftarrow \text{NELDERMEADITERATE}(J, K)$ 
6: end loop
7:  $\gamma \leftarrow \frac{\exp(\sum_t LL_q)}{\exp(\sum_t \max_\ell LL)}$ 
```

---

The UPDATEUSINGRECOGOPTIMIZATION function is described in Algorithm 6.6. The inputs include a plan trajectory  $T$ , the previously determined activity sequence  $q$  that the plan should contain, the task library  $\mathcal{L}$ , and the initial guess of keyframe locations  $K$ . Using the initial keyframes, the system first converts the plan trajectory  $T$  into keyframe trial format  $y$  (line 2). Next, running RECOGNIZETASKSUSINGKEYFRAMETRIALS on the candidate keyframe trial  $y$  produces log likelihoods that the trial describes each activity in the library, computed at each time step. Recall from Figure 6-2 that these log likelihoods can be visualized as the green lines in one of the green boxes. Parts of the log likelihoods corresponding to the previously determined activity sequence  $q$  are then collected and concatenated as  $LL_q$ . In Figure 6-2, if  $q$  were {"activity 2", "activity 3", "activity 1"}, then  $LL_q = \{LL_{\ell=2}(0 \dots K_1), LL_{\ell=3}(K_1 \dots K_2), LL_{\ell=1}(K_2 \dots K_3)\}$ . The larger the log likelihood values are across all time steps, the more likely the found keyframes are the correct

ones. Thus, the algorithm uses the negative sum of the log likelihoods over time as the optimization function to hand to the Nelder-Mead minimizer. This whole process loops until convergence.

To assess how good the set of computed keyframes are, the algorithm compares the recognition likelihoods of the identified activity sequence,  $LL_q$ , with the maximum likelihood over all activity labels,  $\max_{\ell} LL$ , accumulated over all time steps in the trajectory. The maximum likelihood over all labels  $\max_{\ell} LL$  represents the maximum likelihood potential of the trial trajectory given what has been learned in the library, so if the activity sequence likelihood  $LL_q$  is very close to the max potential, then the system can be very confident about the keyframe placements that come out of the optimizer. To compare the two, the algorithm converts them both into probabilities first by taking the exponential, and then take the ratio to represent the confidence level (line 7 in Algorithm 6.6).

## 6.4 Validate auto-segmentation

Once the optimal segmentation points in the plan sequence are determined, the system validates the auto-generated segmentation points (line 15 in Algorithm 6.1) with the user depending on the confidence level of the auto-segmentation. If the confidence is below a certain threshold, the system will ask the user to verify the correctness of the auto-segmentation. During this process, the user interface first displays all auto-segmentation results of non-keyframe trials to the user, and then asks the user to indicate which trials, if any, had been segmented incorrectly.

If none of the additional trials were correctly segmented, i.e. no improvement was made on the non-keyframe trials and the user indicated that all auto-segmented trials were incorrect, then the interface will ask the user for the correct keyframes on one of the previously non-keyframe trials. Asking the user to specify keyframes for just one additional trial minimizes the amount of work the user has to perform, while ensuring that the algorithm improves on at least one trial during each iteration.

Finally, the “keyframe” ( $\mathcal{Y}$ ) and “non-keyframe” ( $\mathcal{Z}$ ) sets of trials can be up-

dated. Those among the non-keyframe trials that have been newly segmented, either autonomously with high confidence, or validated by the user, are removed from the “non-keyframe” set and added to the “keyframe” set. The updated keyframe trials can now provide more information for learning unknown activities and autonomous segmentation, so the process described in Sections 6.2 through 6.4 iterates until no more non-keyframe trials exist (lines 6 to 16 in Algorithm 6.1).

## 6.5 Generating Plan-level Probabilistic Flow Tube

After all plan trajectories are successfully segmented, i.e. all trials have been assigned keyframes, a final pass is made to officially pre-learn the previously unknown activities in the plan from a set of randomly generated initial environment states (lines 17 and 18 in Algorithm 6.1). The new pre-learned activities are saved into the original plan library  $\mathcal{L}$ , and a new subtask sequence  $q'$  is generated using the updated plan library entries.

The final step in plan learning is to create a new library entry for the plan and add it to the library (lines 19 and 20 in Algorithm 6.1). Each library entry  $lib_\ell$  with label  $\ell$  is composed of a tuple  $\langle \mathbf{PFT}, \mathbf{E}, \mathcal{F}, q \rangle$  consisting of a set of pre-learned probabilistic flow tubes  $\mathbf{PFT}$ , a set of corresponding initial environment states  $\mathbf{E}$ , relevant motion characteristics  $\mathcal{F}$ , and a list of subtasks  $q$ . The `GENERATEPLANPFT` function is described by Algorithm 6.7. Inputs include the plan label  $\ell$ , the plan’s activity sequence  $q$ , the existing task library  $\mathcal{L}$  which contains motion variable and PFT information for all learned tasks, and a new environment state  $T0$  in which to generate the plan PFT.

As illustrated in Figure 6-6, to generate the library entry, the set of initial environment states is set to include the new environment state, and the corresponding plan PFT is piecewise constructed from component activity PFTs (also called subtask PFTs) such that the environment state at the end of one subtask becomes the initial environment state in which to generate the PFT for following subtask (lines 1 to 6). If the plan already exists in the library, then the new PFT generated from the new

---

**Algorithm 6.7** GENERATEPLANPFT ( $\ell', q, \mathcal{L}, T0$ )

---

**Input:**

- $\ell'$ , new plan label
- $q$ , sequence of activities in plan
- $\mathcal{L}$ , library of previously learned task entries  $\{lib_\ell\}_{\ell \in L}$
- $T0$ , new environment state in which to generate PFT

**Output:**

- $lib = \langle \mathbf{PFT}, \mathbf{E}, \mathcal{F}, q \rangle$ , new library entry describing the plan

**Notable local variables:**

- $L$ , set of task labels that exist in library  $\mathcal{L}$
- $M = |q|$ , number of activities in sequence

- 1:  $T0^{next} \leftarrow T0$
  - 2: **for**  $i = 1 \dots M$  **do**
  - 3:  $PFT_{q_i}^{T0^{next}} \leftarrow \text{GETPFTSFROMHERE}(q_i, T0^{next})$
  - 4:  $PFT_{\ell'}^{T0} \leftarrow \{PFT_{\ell'}^{T0}, PFT_{q_i}^{T0^{next}}\}$
  - 5:  $T0^{next} \leftarrow \text{ENVSTATEATEND}(PFT_{q_i}^{T0^{next}}, T0^{next}, \mathcal{F}_{q_i})$
  - 6: **end for**
  - 7: **if**  $\ell' \in L$  **then**
  - 8:  $lib_{\ell'} \leftarrow \langle \{PFT_{\ell'}, PFT_{\ell'}^{T0}\}, \{E_{\ell'}, T0\}, \mathcal{F}_{\ell'}, q \rangle$
  - 9: **else**
  - 10:  $\{F_{\ell', \mathbf{X}}^{absStart} \leftarrow F_{q_1, \mathbf{X}}^{absStart}\}_{\mathbf{X} \in \{C, D, P, Q\}}$
  - 11:  $\{F_{\ell', \mathbf{X}}^{relEffStart} \leftarrow F_{q_1, \mathbf{X}}^{relEffStart}\}_{\mathbf{X} \in \{C, D, P, Q\}}$
  - 12:  $\{F_{\ell', \mathbf{X}}^{absEnd} \leftarrow F_{q_M, \mathbf{X}}^{absEnd}\}_{\mathbf{X} \in \{C, D, P, Q\}}$
  - 13:  $\{F_{\ell', \mathbf{X}}^{relEffEnd} \leftarrow F_{q_M, \mathbf{X}}^{relEffEnd}\}_{\mathbf{X} \in \{C, D, P, Q\}}$
  - 14:  $\mathcal{F}_{\ell'} = \langle F_C, F_D, F_P, F_Q \rangle_{\ell'}$
  - 15:  $lib_{\ell'} \leftarrow \langle PFT_{\ell'}^{T0}, T0, \mathcal{F}_{\ell'}, q \rangle$
  - 16: **end if**
-

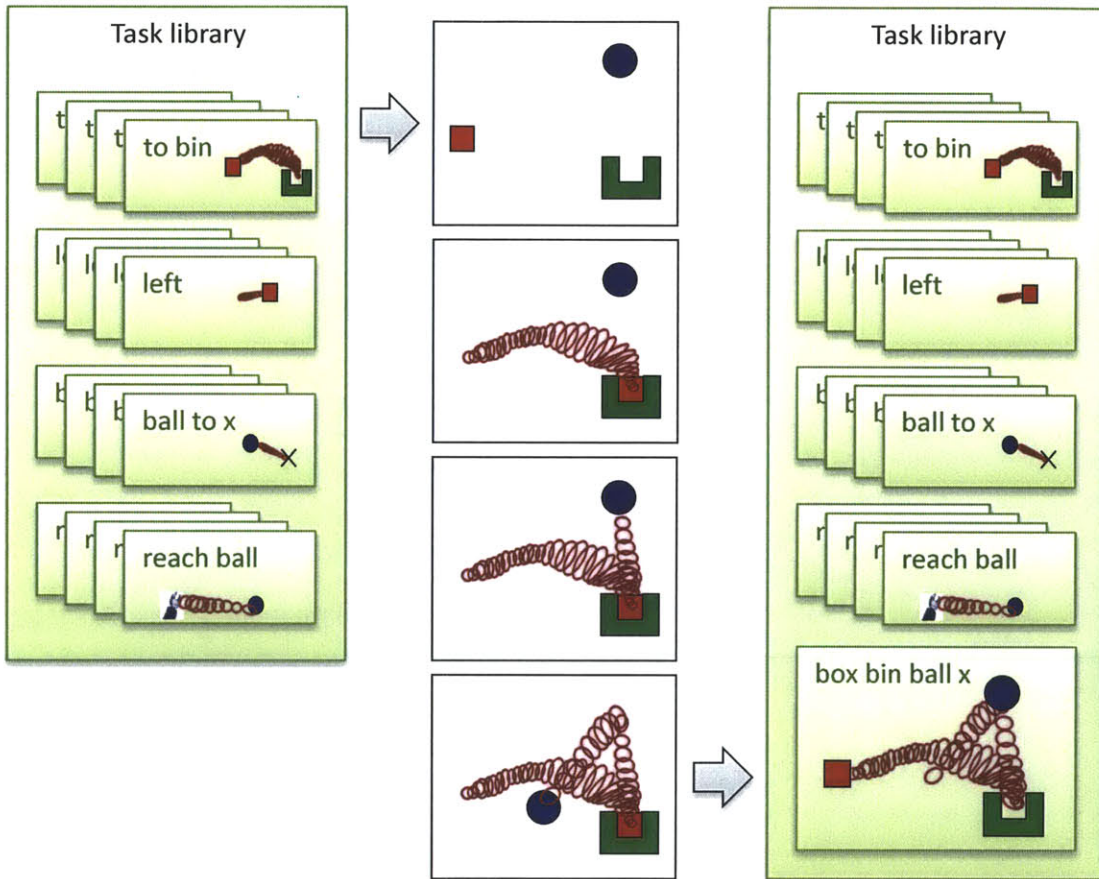


Figure 6-6: Generating the PFT for the entire plan involves concatenating the PFTs for each activity in the sequence initialized at evolving environment states

environment state is added to the set of previously stored PFTs for that library entry. The library entry's list of subtasks is directly set to the learned activity sequence  $q$ . Plan characteristics with modes *absStart* and *relEffStart* are set to those of the first subtask in  $q'$  while plan characteristics with modes *absEnd* and *relEffEnd* are set to those of the last subtask in  $q'$  (lines 10 to 14).

# Chapter 7

## Experimental Results

### 7.1 Validation of Activity Learning and Recognition

In this section, I first present my activity learning and recognition results in a two-dimensional simulated environment. Next, I test recognition performance on the Barrett Technologies Whole Arm Manipulator (WAM) robot. Finally, I demonstrate the system executing on the JPL ATHLETE and Willow Garage PR2 robots.

#### 7.1.1 Two-dimensional Variable Environment

In the simulated environment, there are four entities: a red box, a green bin, and two stationary locations marked  $x$  and  $o$ . The box and bin are positioned at varying random locations in the environment, while the  $x$  and  $o$  always mark the fixed locations  $(5, 5)$  and  $(5, 3)$ , respectively. A user can produce a motion by moving the mouse around in the region. During training, the user labels each demonstrated sequence with the name of the motion.

In the first experiment, users taught the system three different motions: “move the box to the bin,” “move the box left one unit,” and “move the box to  $x$ .” During each trial, the box and bin locations were randomly generated. Two users demonstrated 150 trials across the three motions. Thirty randomly chosen trials of each motion

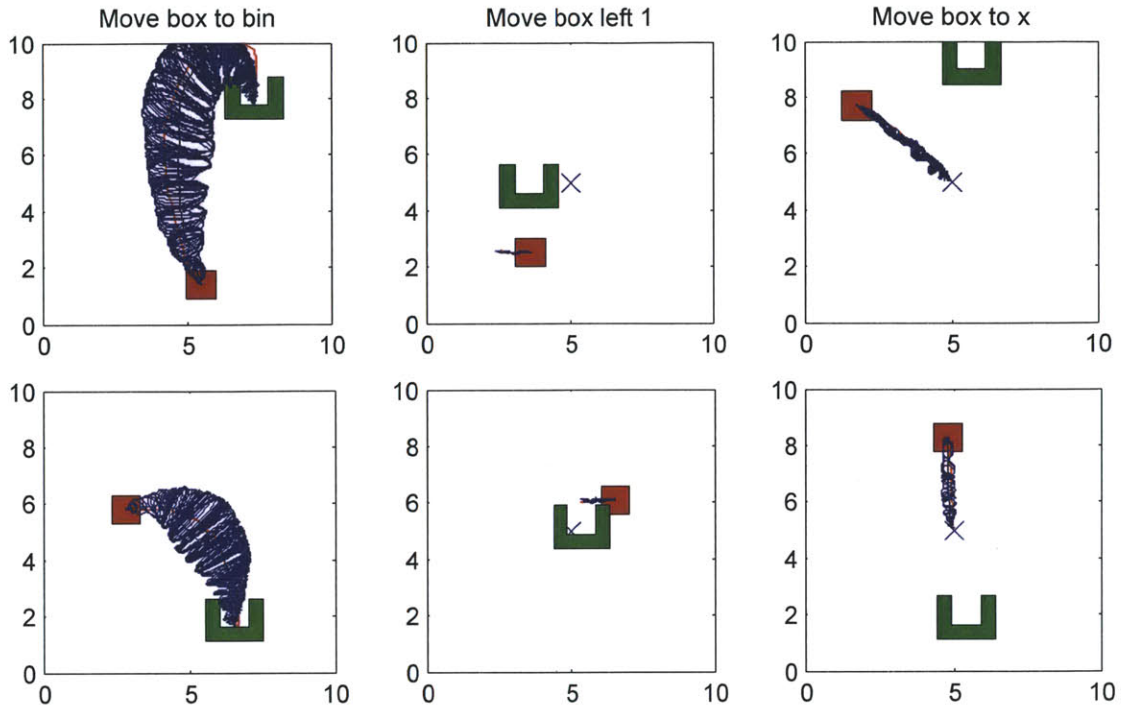


Figure 7-1: Example learned PFTs in randomly generated initial environment states.

were used for training, and the remaining 60 trials were used for testing.

Figures 7-1 and 7-2 show some examples of what the learned PFT models look like in the environment states of three randomly chosen test cases. One can see from Figure 7-2 that distinguishing among different motions is easier in some environments than others due to the relative positions of objects. For example, if the bin were located at  $x$  and the box straight above, then it is impossible to distinguish between “move to bin” and “move to  $x$ .”

I compare the PFT model to that presented by Mhlig et al. [43], which also uses dynamic time warping to temporally match demonstrated trajectories, but uses Gaussian mixture models (GMM) to describe learned motions. These GMMs are generated using Expectation Maximization with a Bayesian Information Criterion to determine the optimal number of Gaussians. Mhlig’s approach assumes prior knowledge of the type of motion; for comparison purposes, I used my algorithm for motion variable identification, and then normalized all trajectories to the appropriate start and end positions before applying the GMM.



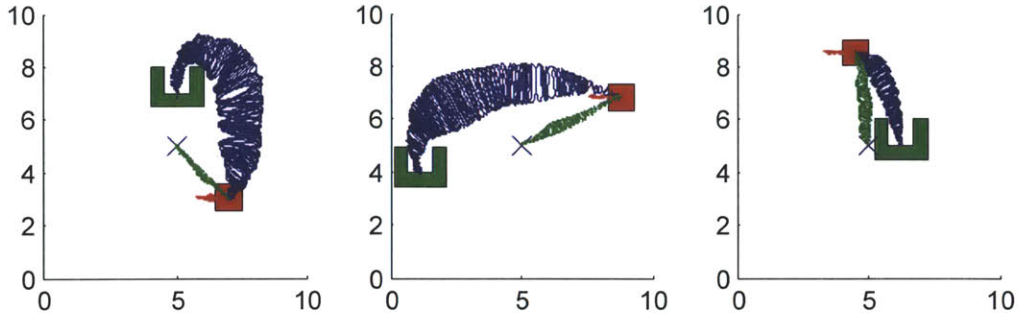


Figure 7-2: Example learned flow tubes of different activities overlaid in three different initial environment states. Blue PFTs represent “move box to bin,” red PFTs represent “move box left 1 unit,” and green PFTs represent “move box to  $x$ .”

Figure 7-3 compares the learned models on three motions in random environment states. In each case, the GMM approach automatically determines the number of Gaussians based on training data. In the “move box to bin” motion, the GMM approach determined that two Gaussians were sufficient to represent the motion. While the generated Gaussians work well for the “move box left 1” and “move box to  $x$ ” motions, I argue that the generated Gaussian mixture model for the “move box to bin” motion does not entirely capture the complexity of the motion. When compared against the user generated trajectory, the Gaussian mixture model is a poor representation in the “move box to bin” motion.

For recognition, I compare my approach to that of Martin et al. [39], which represents learned motions as tied-mixture hidden Markov models based on GMMs, and uses a buffered Viterbi algorithm for fast recognition. For comparison, I implemented their modeling approach using 3 states and 6 Gaussian mixtures, which was found to perform reasonably well.

Table 7.1 compares the results of the first experiment. The first comparison is how many test cases were correctly recognized by the end of each motion using the PFT approach ( $N_P$ ) versus the HMM approach ( $N_H$ ). My algorithm recognized 49 of the 60 test motions (82%), while the HMM-based algorithm recognized 33 (55%).

The second comparison is how long throughout a test motion did the algorithm maintain the correct classification ( $\%_P$  versus  $\%_H$ ). On average, my algorithm recognized the motion correctly 71% of the time spent during a test motion, while the

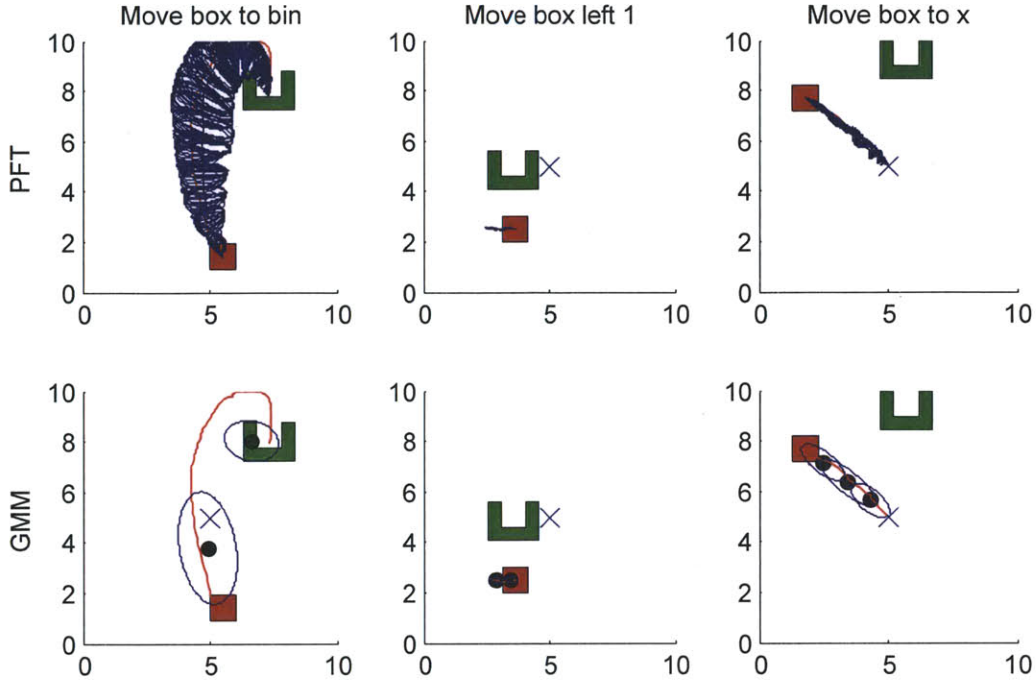


Figure 7-3: Compare learned PFT and GMM models (blue) against user generated trajectories (red) in different initial environment states. Blue ellipses represent the range of 1 standard deviation.

HMM approach spent on average 40%. This second comparison metric reflects how frequent the real-time estimates are correct throughout a test case, since as a test motion progresses, the estimated most likely motion label may change given new information at each time step.

An example of how estimation likelihoods may change for the two approaches throughout the same test cases is shown in Figure 7-4. The PFT approach starts

Table 7.1: Comparison of recognition approaches using PFTs ( $P$ ) and HMMs ( $H$ ) in variable environments.  $N$  is the number tests out of 20 that was correctly recognized by the end of each motion.  $\%$  is the mean percent of real-time execution that the correct motion was recognized.  $t$  is the average computation time at each real-time instance (in seconds).

	$N_P$	$N_H$	$\%_P$	$\%_H$	$t_P$	$t_H$
To bin	<b>20</b>	12	<b>81.1</b>	55.3	<b>0.008</b>	0.047
Left	<b>13</b>	6	<b>56.0</b>	28.0	<b>0.004</b>	<b>0.004</b>
To $x$	<b>16</b>	15	<b>74.4</b>	36.3	<b>0.005</b>	0.006

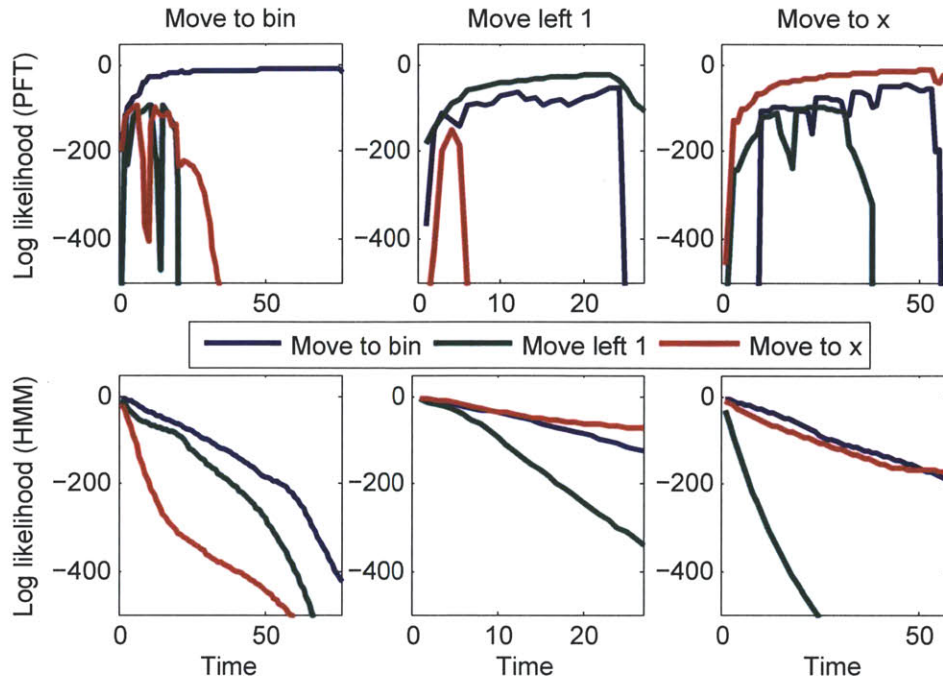


Figure 7-4: Example log likelihoods over time for the same test cases using the PFT approach (top row) and the HMM approach (bottom row)

each motion with low likelihood, while the HMM approach starts each motion with equally high likelihood. Throughout the motion, the HMM approach updates the log likelihoods smoothly, but is often unable to distinguish among the motions. The PFT approach often produces higher frequency likelihood changes as a result of the spatio-temporal relationships among the motions, but it is often able to appropriately distinguish among them through time. For example, comparing the “move to x” test case in Figure 7-4, the HMM approach has difficulty distinguishing it from the “move to bin” motion. The PFT approach, however, quickly concludes that the label “move to x” is most likely, but “move to bin” also has a small log likelihood throughout. During the first part of execution, “move left 1” is also a low probability contender, but after the user moves beyond 1 unit, the likelihood of the motion being “move left 1” quickly drops to zero, which is a behavior that makes sense intuitively.

Finally, the third metric comparison in Table 7.1 is how long, on average, the algorithm takes to compute new likelihoods at each time step in a test motion ( $t_P$  versus  $t_H$ ). Both approaches have fairly short average computation times considering

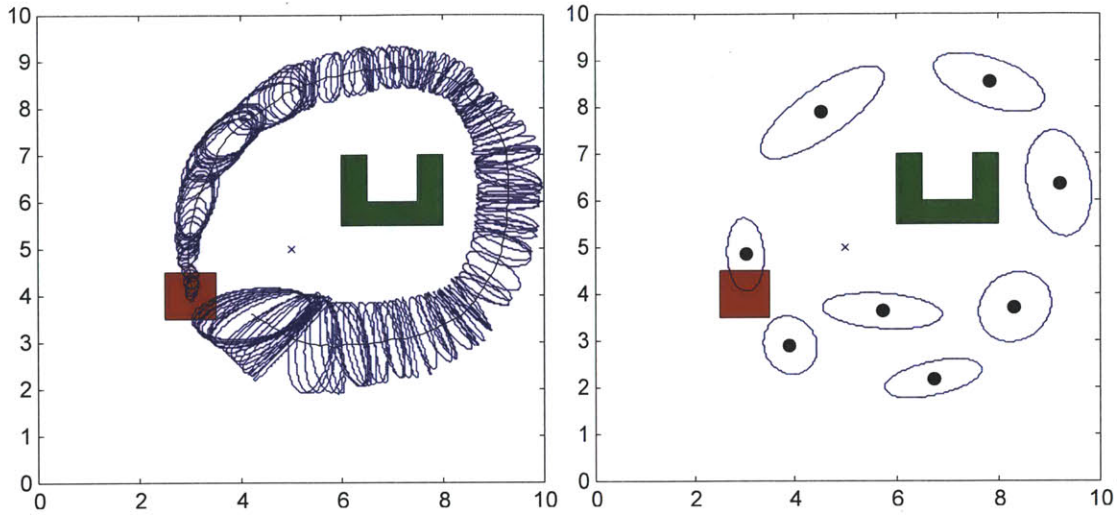


Figure 7-5: Example output of an “encircle bin clockwise with box” motion. Left: the PFT model. Right: the GMM model.

human reaction time is generally longer than 0.1 second.

One can see that the PFT approach is able to perform real-time recognition reasonably well even when the environment state varies among the different training and test trials. This is an important advantage in learning and recognizing manipulation tasks because the state of the environment often changes with manipulation, and it is desirable to reduce the number of redundant training demonstrations an operator must perform in any given environment setup.

### 7.1.2 Two-dimensional Static Environment

When comparing the PFT model to the GMM model, I found that the GMM representation suffers particular drawbacks in cases where temporal ordering of a motion is important, since it is computed based on spatial coordinates alone. For instance, GMMs cannot distinguish a clockwise circular motion (such as winding a cable) from a counter-clockwise motion (unwinding), since they occupy the same spatial region, as shown in Figure 7-5’s comparison of the GMM representation with our probabilistic flow tube representation. Lack of temporal ordering in the GMM model poses a limitation during online recognition, as I will present below.

Since the HMM/GMM-based technique is not designed to handle variations in the

Table 7.2: Comparison of recognition approaches using PFTs ( $P$ ) and HMMs ( $H$ ). There were 100 cross-validation test cases for each motion.

	$N_P$	$N_H$	$\%_P$	$\%_H$	$t_P$	$t_H$
To bin	<b>95</b>	94	65.6	<b>67.9</b>	<b>0.020</b>	0.064
Left	94	<b>100</b>	81.6	<b>95.6</b>	<b>0.007</b>	0.009
To $x$	96	<b>98</b>	79.3	<b>85.1</b>	<b>0.009</b>	0.010
CW	<b>100</b>	72	<b>83.2</b>	65.2	<b>0.023</b>	0.038
CCW	<b>99</b>	72	<b>94.3</b>	80.7	<b>0.022</b>	0.031
$x,o$	<b>96</b>	56	<b>79.0</b>	52.8	<b>0.034</b>	0.370
$o,x$	<b>99</b>	57	<b>86.7</b>	45.4	<b>0.035</b>	0.386

environment state, I chose a particular environment state and generated 25 trials for each of 7 motions in order to make a more fair comparison. The environment state and the 7 motions are the same ones shown in Figure 5-1. I used 5-fold cross-validation using 5 trials for training and 20 for testing on each motion. Only 5 trials are needed for training because the motions are fairly similar in the static environment.

Table 7.2 summarizes the comparative results. The HMM-based approach performed slightly more favorably for the goal directed motions “move to bin,” “move left 1,” and “move to  $x$ .” It had more trouble with motions where directionality plays an important role, and it performed poorly on motions that are not Markov in nature, such as the anchor loops. To the Markovian model, the loop, say around  $x$ , looks the same locally in the “anchor around  $x$  then  $o$ ” motion as it does in the “anchor around  $o$  then  $x$ ” motion. The PFT representation performs especially well for these more complex motions. Averaging over all seven motions, the HMM approach achieved a 78% recognition rate. My algorithm achieved an overall 97% recognition rate while using less computation time on all of these motions. If we look at only the non-goal directed motions, the HMM approach achieved a 64% recognition rate while my PFT approach averaged 99%.

### 7.1.3 Hardware Validation of Real-time Recognition

I also demonstrated my motion recognition capability on a Barrett Whole Arm Manipulator (WAM) robot to illustrate the performance on a real-world platform. A user trained the robot by physically moving it through each of 5 motions in gravity

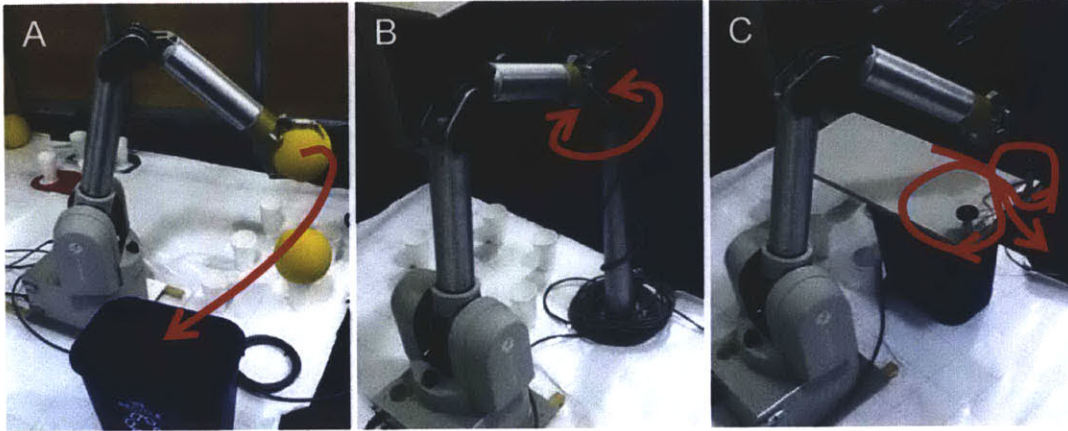


Figure 7-6: WAM robot setups for the five motions: “move ball to bin”(A), “wind cable” (B), “unwind cable” (B), “anchor rope left then right” (C), and “anchor rope right then left” (C)

Table 7.3: Results on WAM robot using PFT approach. There were 10 cross-validation test cases for each motion.

	$N_P$	$\%P$	$t_P$
To bin	10	93.5	0.024
Wind	10	92.7	0.013
Unwind	10	96.4	0.011
Anchor LR	10	84.1	0.032
Anchor RL	10	86.2	0.033

compensation mode 10 times. The motions include: “move ball to bin,” “wind cable,” “unwind cable,” “anchor rope left then right,” and “anchor rope right then left,” the setups of which are shown in Figure 7-6. Each trial for a motion started with the same environment state and was recentered in post-processing to all have the same starting location. Training trajectories can be found in Figure 7-7. Five of the 10 trials for each motion were used for testing for each of two cross-validations.

Table 7.3 shows that my algorithm successfully recognized all 50 cross-validation test trials correctly by the end of each motion, and spent on average 91% of each test trial classifying the motion correctly. It seems that the higher dimensionality of the state space is quite favorable for recognition as it allows motions to diverge more.

In this demonstration, the environment state did not change for the different demonstrations, so only a few number of user demonstrations were required to be

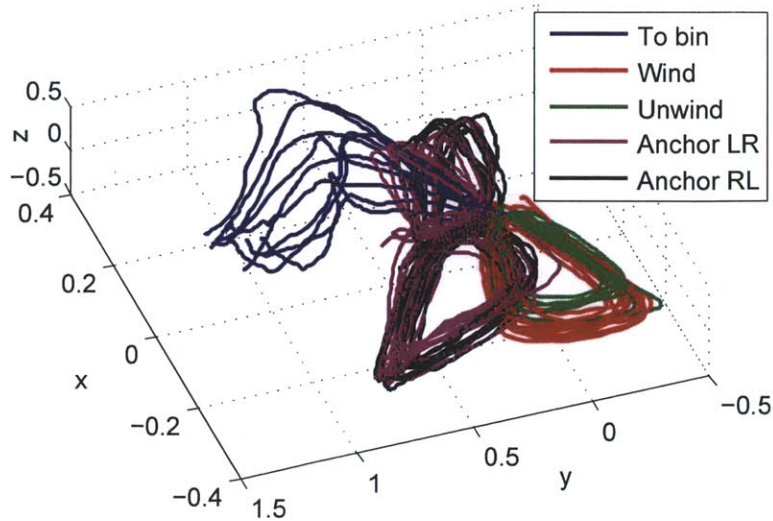


Figure 7-7: WAM end effector trajectories

able to learn generalized PFTs of each motion. An interesting future study could determine the optimal number of user demonstrations needed for a certain degree of variation in the environment state.

#### 7.1.4 Hardware Demonstration of Autonomous Execution

To exhibit the ability of my learning algorithm in generating reasonable humanlike motions that can be used for autonomous execution, I performed demonstrations on two different robotic platforms. My colleagues and I have previously reported a proof-of-concept demonstration [20, 19] on the All-Terrain Hex-Limbed Extra-Terrestrial Explorer (ATHLETE) robot [41] through a collaboration with Caltech’s Jet Propulsion Laboratory. This robot is designed to transport habitats and other large objects on the moon. Its six wheeled limbs can be used for driving, walking, or object manipulation with certain tool attachments. We demonstrated autonomous execution on a “move box to platform” task as shown in Figure 7-8.

We provided five teleoperated demonstrations of a construction motion that picked up a large box and placed it on top of a platform, as shown in Figure 7-9. In each demonstration, the box and platform were moved to slightly different locations. Because ATHLETE had no onboard sensing, we manually provided the positions of the

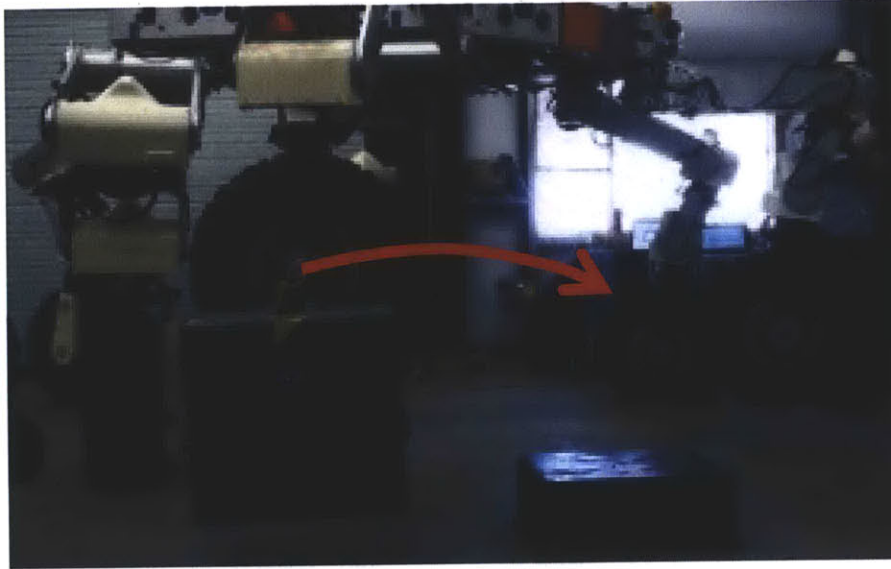


Figure 7-8: ATHLETE move box to platform task

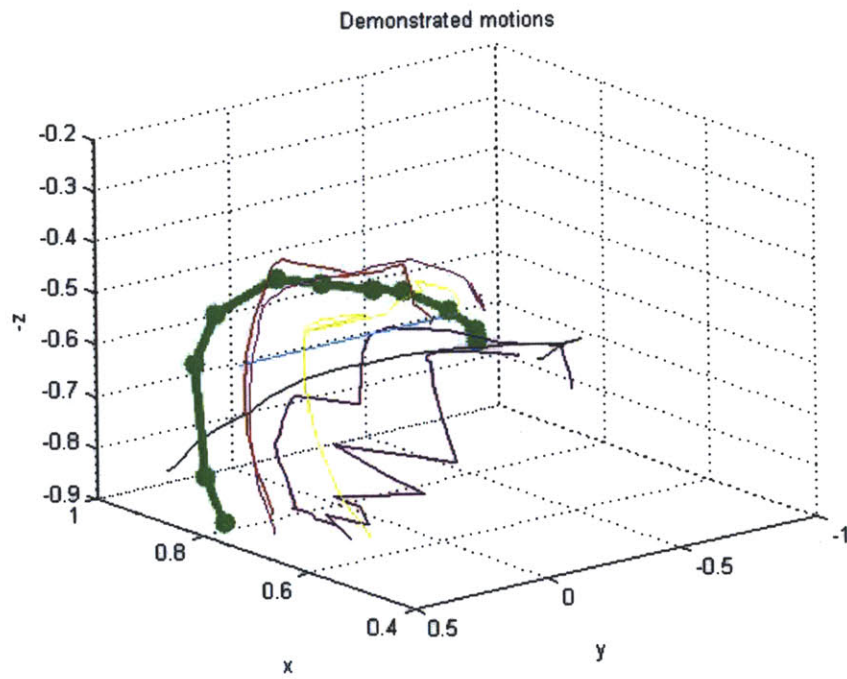


Figure 7-9: Five teleoperated demonstrations of the “ATHLETE move box to platform” task. Autonomous execution of the task is shown in thick green.



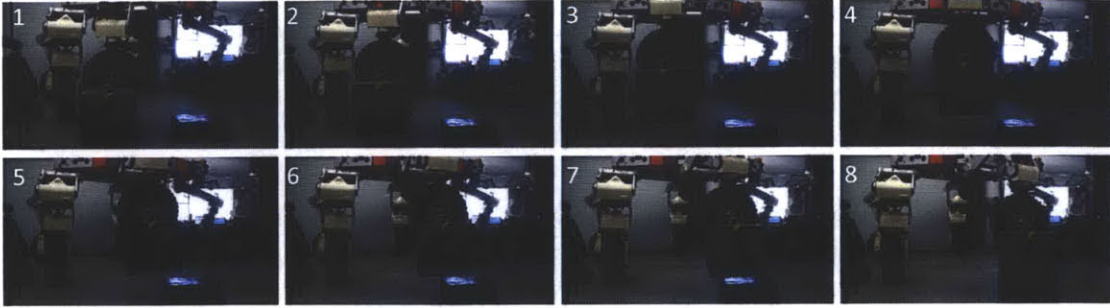


Figure 7-10: Autonomous execution of ATHLETE move box to platform task

objects. We then moved the objects to previously unseen locations and used my learning algorithm to generate a trajectory for autonomous execution. The algorithm successfully identified the relevant motion variables for this task, and the resulting autonomously executed motion (the nominal trajectory of the learned probabilistic flow tube) shown in green in Figure 7-9 appropriately resembled the demonstrations and successfully accomplished the task. Note that the original five user demonstrations had a large amount of noise. The resulting learned trajectory was smoother due to the averaging during the learning process and down sampling to generate waypoints to send to the controller. Figure 7-10 shows ATHLETE autonomously executing the learned task.

I also demonstrated my learning algorithm on the PR2 robot developed by Willow Garage, as part of the Learning from Demonstration (LfD) Challenge at the 2011 AAAI conference. Prior to the conference, users performed teaching demonstrations and preliminary testing through the Bosch remote lab facility [47, 46]. The system used the PR2’s onboard sensing and the Robot Operating System (ROS)’s object recognition software to record the environment states throughout the demonstrations. In the physical setup shown in Figure 7-11, the PR2 faced a table where certain objects are placed, and all motions were taught kinesthetically by manually moving the right arm of the robot.

Users taught the robot and tested the learning algorithm on several different motions, including: “move left,” “move up and over,” “go home,” “pour into,” “pour done,” “reach,” “put on table,” “shake,” “stir.” Five demonstrations of each motion

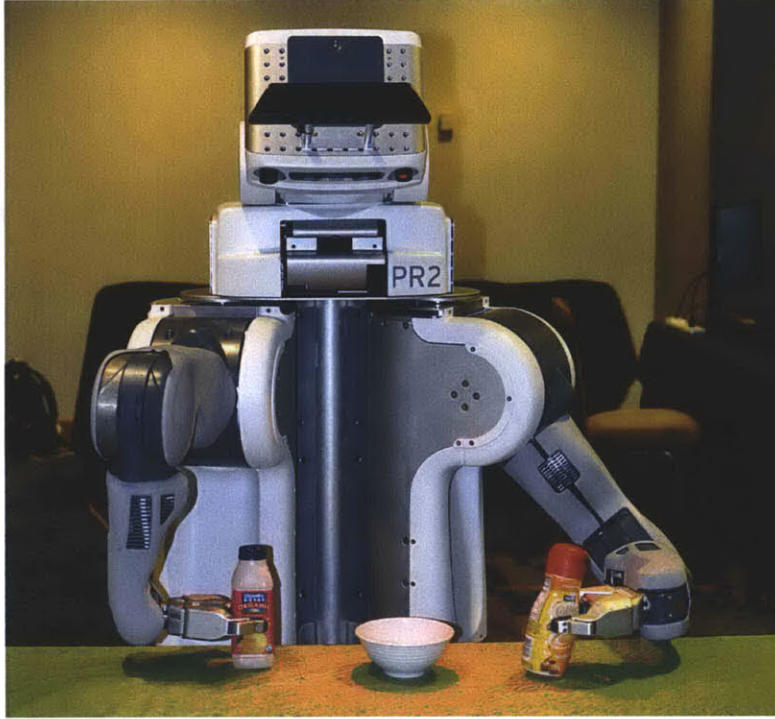


Figure 7-11: PR2 and potential task setup

were provided. Some additional helper commands including “close grip,” “open grip,” and “detect objects,” were handled separately through ROS and were not learned. During execution, learned objects for the “pour into” and “reach” motions could be optionally replaced by user-specified objects. Users could also optionally instruct the robot to use its left arm to complete the task, which was accomplished by inverting the sign of the  $y$  position and  $q_w, q_y$  quaternion values during execution.

Figure 7-12 shows the behaviors of the learned motions compared with the original user demonstrations. During autonomous execution in these example outputs, the environment contains only one object (an odwalla bottle). For each demonstration, the object and robot end effector (measured at the PR2 wrist roll link) are set to begin with the same arbitrary poses. The computation time for offline PFT learning for each motion ranged between 0.9 seconds and 1.3 seconds for user inputs sampled to about 100 data points each.

The “pour into,” “pour done,” and “reach” motions were demonstrated with an object in the environment. The algorithm learned that at the end of the “pour into”

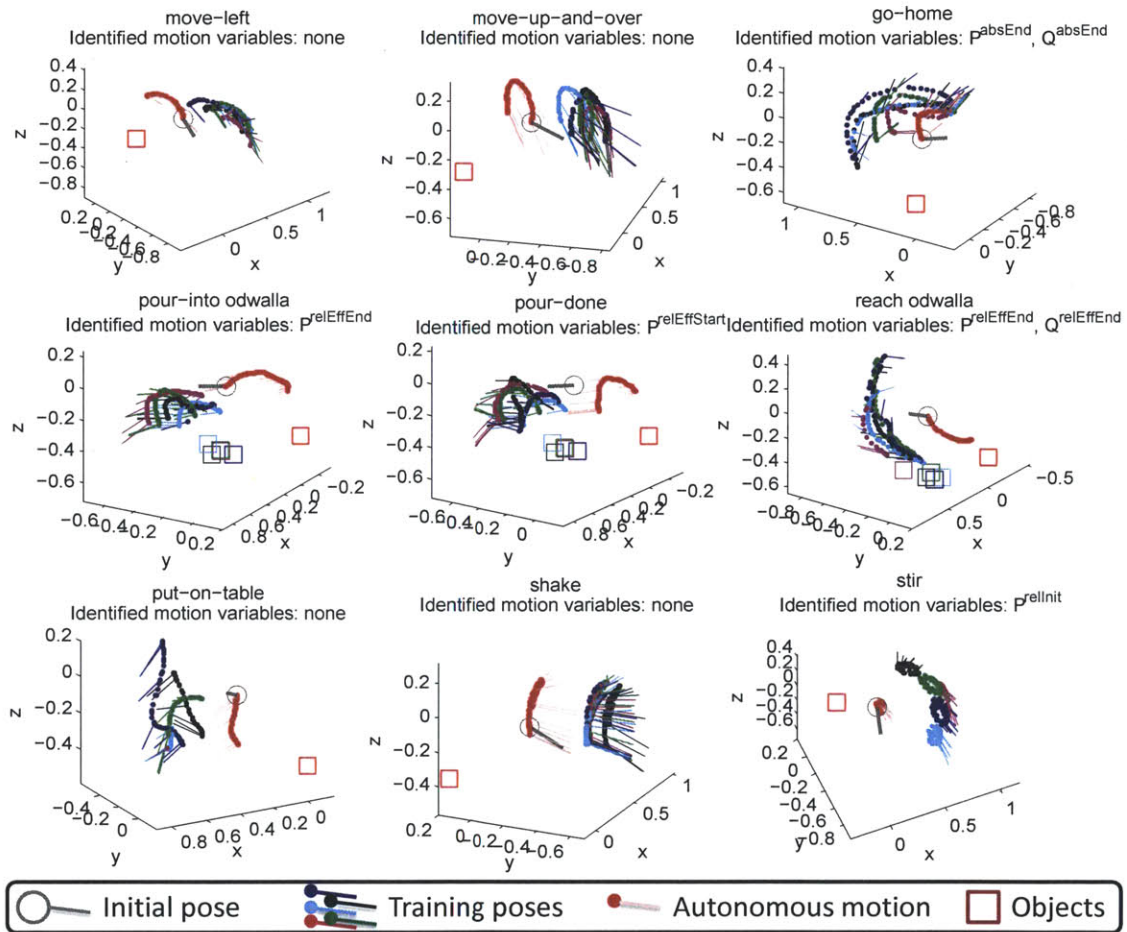


Figure 7-12: Results of learning from 5 demonstrations of each motion.

motion, the robot end effector is always a certain distance away from the object, and consequently in the beginning of the “pour done” motion, the effector always starts relative to the object it was just pouring into. In the “pour done” example in Figure 7-12, although the effector’s initial pose is not over the object, it is immediately moved to the appropriate position before executing the motion. The “shake” motion was a quick up-down-up-down activity inspired by the movement used to shake an orange juice bottle. The “stir” motion involved making five complete circular movements. Interestingly, this resulted in the algorithm learning that the end position of the “stir” motion should be relative to the initial position. The learned motion turned out to be more of a small vibration around a point. After investigating the issue further, it appeared that the user demonstrations generally all consisted of circular motions, but each was on a different plane, so that taking geometric means became less representative of the original motions.

In the motions where the object is determined to be not relevant, the autonomous execution correctly ignores the object completely. In the pouring and reaching motions, users taught the motions using an object different from the odwalla bottle. During execution, users can specify which object to use if different from those in the demonstrations. In this way, learned motions using one object may be extended to other similar objects.

During the AAI LfD Challenge, my teammate and I enabled the PR2 robot to autonomously execute a scenario that involved many of the motions shown in Figure 7-12. The initial setup resembled that of Figure 7-11, except both bottles were on the right side. First, the right arm was commanded to reach for the coffee-mate bottle, then move it up and over to the left. The left arm then reached for the coffee-mate, poured it into the bowl, and placed it back down. Next, the right arm reached for the odwalla bottle, gave it a shake, and poured it into the bowl. Each time we performed this demonstration scenario, we arbitrarily put the objects in new locations.

We were limited in the kinds of tasks and number of objects we could use by the physical abilities of the robot. The PR2 vision system had a very narrow field of

view, so all motions had to be performed in an area of about one square foot on the table. This limited the number of objects that could feasibly fit in the visible range. Another frequent issue was that the object recognition would become confused when the robot end effector was holding an object because it would detect its own effector and object as one large unidentifiable object. Despite these issues, we were able to successfully demonstrate the aforementioned scenario and other motions.

## 7.2 Validation of Plan Learning

This section demonstrates the results of learning complex plans from user demonstration, first in the two-dimensional simulated environment, and then on the Barrett Technologies Whole Arm Manipulator robot.

### 7.2.1 Two-dimensional Environment Tests

In the two-dimensional world, a blue ball was added to the environment in addition to the red box and green bin objects to allow for more complex activity sequences. A user demonstrated plan learning on two plans in the two-dimensional environment: “move box to bin and ball to x” and “move box to x, then bin.” The first plan is composed of the activity sequence {“reach box”, “move box to bin”, “reach ball”, “move ball to x”, “done with ball”}. The second plan is composed of the activity sequence {“reach box”, “move box to x”, “move box to bin”, “done with box”}.

I first compare different approaches to auto-segmentation. In the approach discussed in Section 6.3, the system typically determines segmentation points in non-keyframe trials by incorporating the geometric information provided by motion variables. For motions without identifiable motion variables, the algorithm uses an optimization on recognition likelihoods approach as a backup option. Here, I compare the performance of auto-segmentation using entirely the motion variable approach versus using entirely the recognition optimization approach. Both are compared to ground truth segmentations that the user provides.

For each plan, a user provided 50 demonstration trajectories, of which 3 were

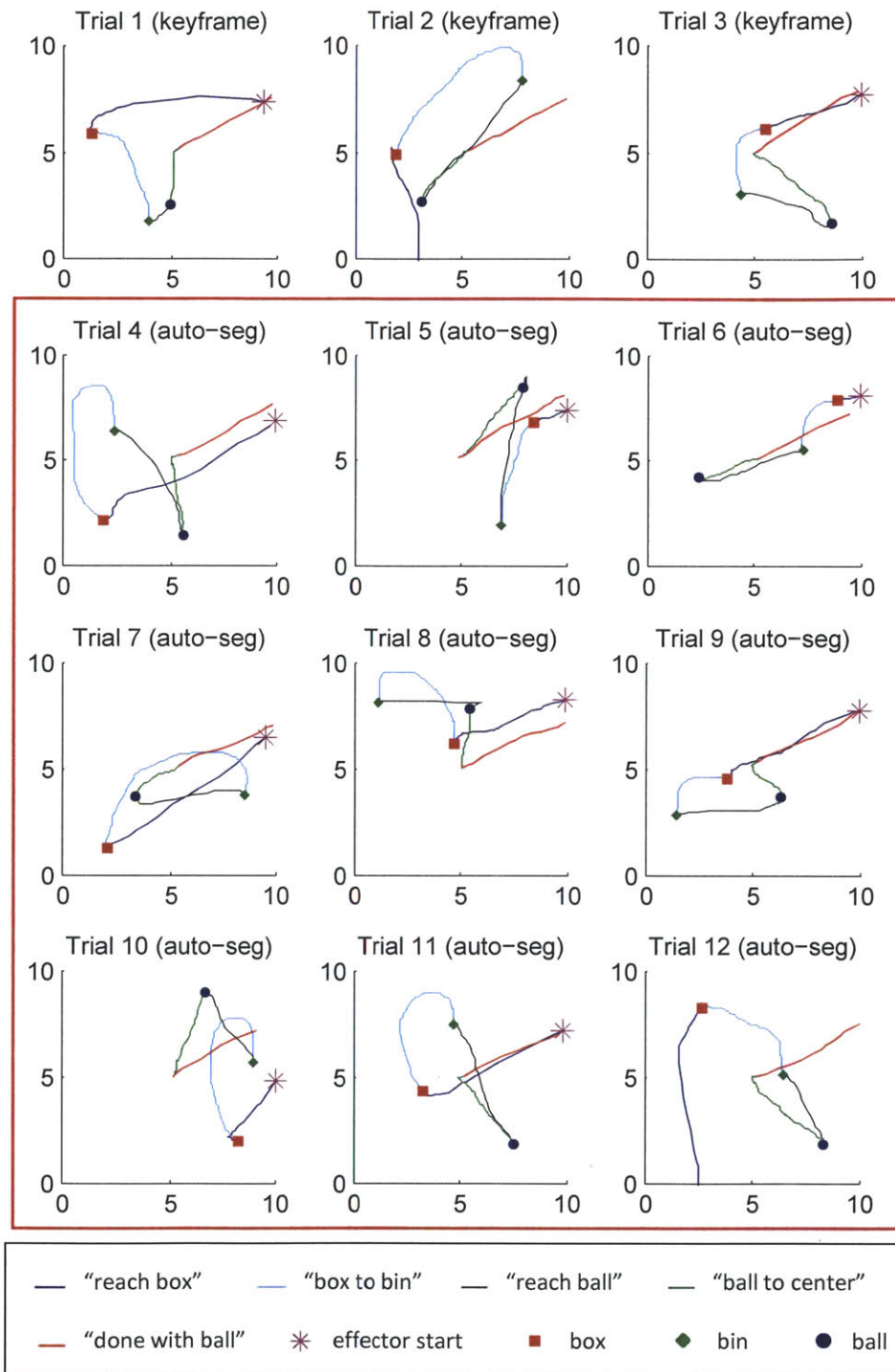


Figure 7-13: Result of auto segmentation on non-keyframe trials (boxed) of "box to bin, ball to center" plan, from 3 user provided keyframe trials using motion variable inference approach. Compare with Figure 7-14.

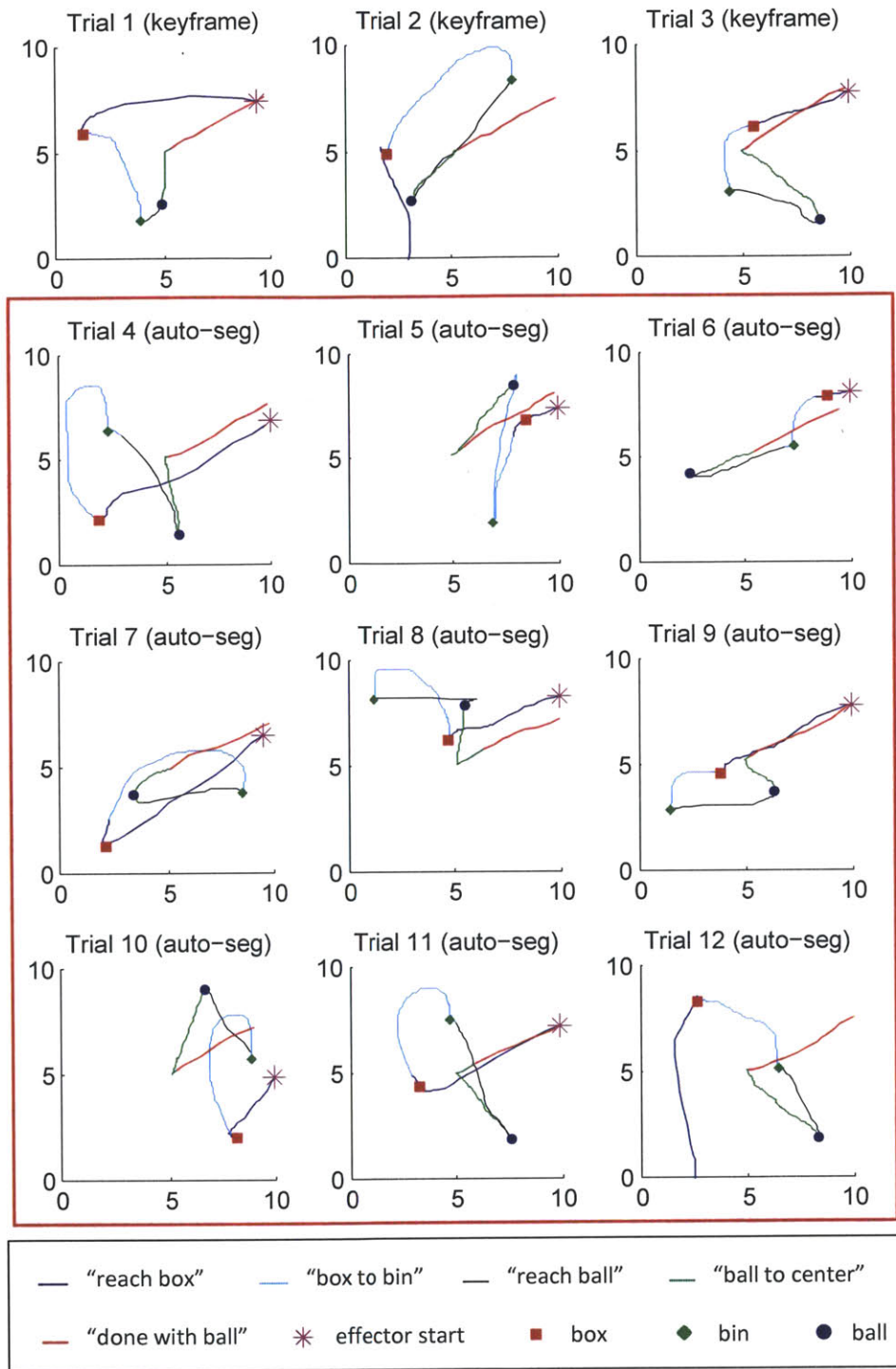


Figure 7-14: Result of auto segmentation on non-keyframe trials (boxed) of "box to bin, ball to center" plan, from 3 user provided keyframe trials using recognition optimization approach. Compare with Figure 7-13.

Table 7.4: Comparing results of auto-segmentation using motion variable approach and recognition optimization approach on two plans (47 trials each)

	$\mu_t$ (sec)	$\sigma_t$	$\mu_{error}$	$\sigma_{error}$
“Box bin ball x” (recog. optimization)	25.6	17.0	0.1023	0.0753
“Box bin ball x” (motion variable)	<b>0.0018</b>	0.0011	<b>0.0604</b>	0.0283
“Box x bin” (recog. optimization)	24.6	9.4	0.2569	0.1322
“Box x bin” (motion variable)	<b>0.0013</b>	0.0003	<b>0.0513</b>	0.0322

assumed to be pre-segmented keyframe trials. Figure 7-13 shows the results of auto-segmentation using the motion variable inference approach on nine of the first twelve trials (where the first three are keyframe trials) of the “box to bin, ball to x” plan (“x” is also referred to as the “center”). The robot effector start position is marked with an asterisk, while the initial environment states of the objects are also noted in each trial. The trajectory colors indicate the portions of each trajectory that belongs to each activity label after segmentation. Trials 1 through 3 were the keyframe trials provided by the user while trials 4 through 12 are auto-segmented by our algorithm. Note that the auto-segmentation is nearly perfect in every trial.

I compare these auto-segmentation results with those obtained from recognition optimization in Figure 7-14. The first three trials are the same user provided keyframe trials. One can see that the recognition optimization approach also performs well in most trials while making some mistakes in a few trials. For example, the algorithm failed to identify the “move ball to center” activity in trial 5, instead considering it a part of the “move box to bin” activity. This is possibly because the “move box to bin” activity has a very wide flow tube allowing much room for deviation while the “move ball to center” motion has a very narrow flow tube allowing for very little deviation based on the training trajectories, causing the recognizer to consider it more likely that the trajectory is executing the “move box to bin” activity in a roundabout way, rather than the “move ball to center” activity since it deviates slightly from the learned flow tube. Additionally, I note that the recognition optimization approach slightly overestimates the “move ball to center” activity in trials 8 and 11.

Table 7.4 compares the motion variable inference auto-segmentation approach with the recognition optimization auto-segmentation approach in terms of computa-



tion time and auto-segmentation accuracy. These measurements are taken from all 47 test trials based on the 3 user provided keyframe trials for each of the “move box to bin and ball to x” and “move box to x, then bin” plans. The mean computation time  $\mu_t$  and corresponding standard deviation  $\sigma_t$  are given in seconds, and reflect the amount of time spent performing auto-segmentation on a single trial, using a MATLAB implementation on an Intel Core i7 processor. The auto-segmentation accuracy is reflected in the error rate  $\mu_{error}$ , which is the fraction of time steps in the trial that has an erroneous activity label due to segmentation error, and its corresponding standard deviation  $\sigma_{error}$ .

The motion variable inference approach outperforms the recognition optimization approach in both computation time and segmentation accuracy. Since motion variable inference is computed in one shot whereas recognition optimization is computed iteratively until convergence, the former is orders of magnitudes faster than the latter (on the order of a thousandths of a second versus over twenty seconds). The motion variable inference approach also has better segmentation accuracy. As shown in Table 7.4, in the “box to bin, ball to x” plan, on average 94% of time steps were correctly segmented using the motion variable inference approach whereas 90% of time steps were correctly segmented using the recognition optimization approach. In the “box to x, then bin” plan, the segmentation accuracies are 95% versus 74% in favor of the motion variable inference approach. Therefore, it seems reasonable to use the motion variable inference approach for auto-segmentation whenever possible, and use the recognition optimization approach as a backup for motions without identifiable motion variables.

Next, the approach generates plan PFTs in new environment states using our training trials. Figure 7-15 displays the results of generating PFTs for the “box to bin and ball to center” plan in nine new environment states depicted by the locations of the objects and effector. The pink trajectory describes the generated nominal trajectory of the PFT, and the covariances are shown in light blue. Prior to learning this plan, the task library already contained “reach for box” and “move box to bin” activities. However, the other activities (“reach ball”, “move ball to x”, and “done

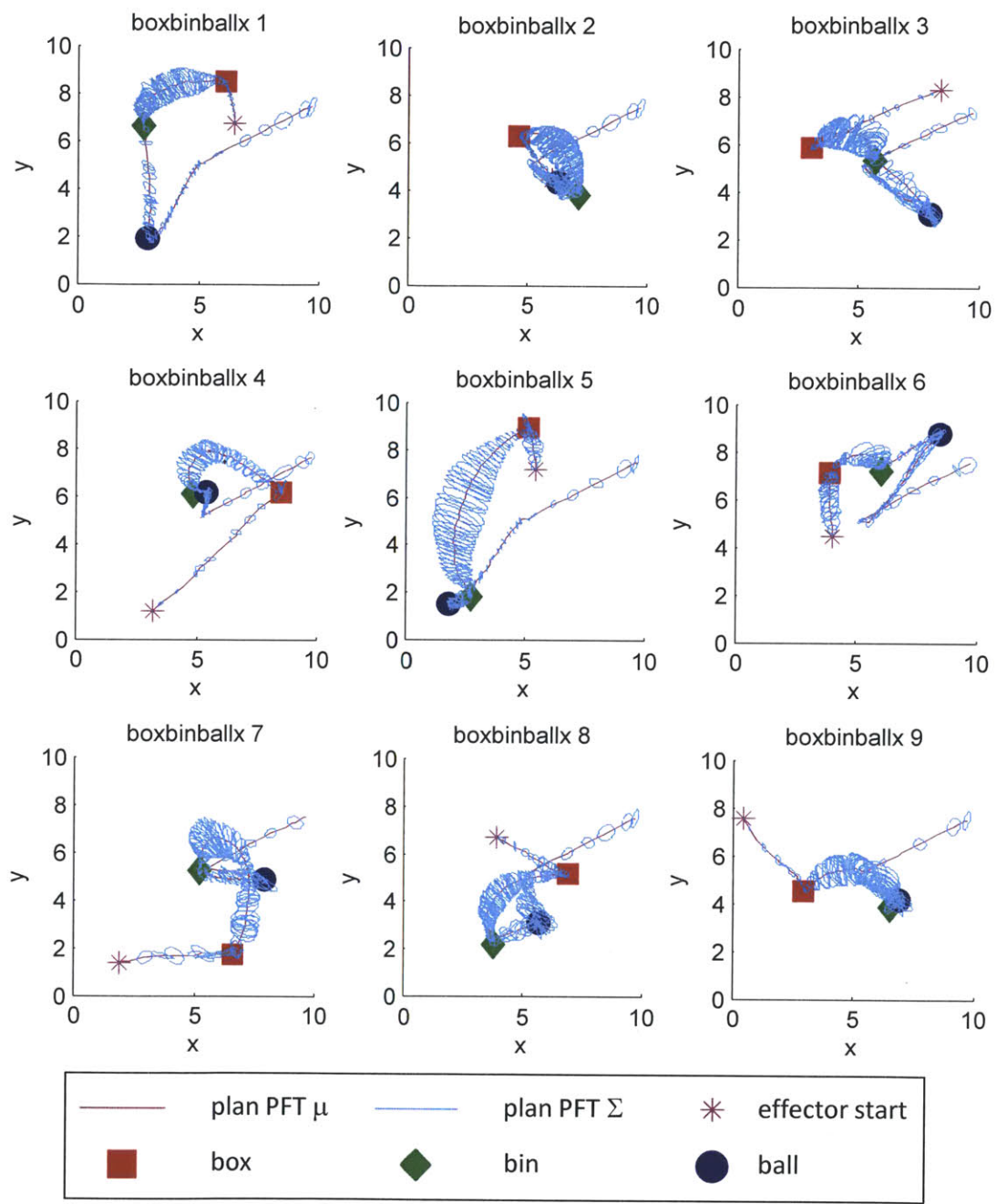


Figure 7-15: Example generated plan PFTs for “box to bin and ball to center” plan from different initial environment states.

with ball”) were identified as “unknown” activities and learned on the fly. In each new environment state, the plan learner was able to generate a reasonable trajectory that achieves the task. I note that the training demonstrations generally ended toward the upper right of the environment in order to reach a “save” button in our user interface, hence the generated “done with ball” activities also reflect this behavior.

### 7.2.2 Hardware Demonstration of Plan Learning

As shown in Figure 7-16, the hardware setup consists of a 7-degree of freedom Whole Arm Manipulator (WAM) robot developed by Barrett Technologies complete with a 3-fingered Barrett Hand [69], a rolling cart, and four colored (blue, red, pink, and green) blocks. Each item is labeled with fiducial tags that can be detected by a ground based vision system. The user interface also incorporates the Sphinx voice commanding system [28] to actuate the WAM hand. The software to run the hardware, sensing, and user interfaces are created through ROS (Robot Operating System), an open source and reusable set of robot software libraries developed by Willow Garage [54]. I also developed a ROS node to interface with the learning code in MATLAB. A complete description of the hardware system capabilities is presented by Levine [37].

All hardware control and sensing feedback is handled through a simulation environment built on the OpenRAVE platform [15], an open source 3D simulation and collision detection environment, as shown in Figure 7-17. During the experiments, the simulator is run in hardware mirroring mode, in which the simulator models the state of the world as well as possible given the limitations of the sensing system. The WAM arm hardware controller is equipped to send proprioceptive sensory information as joint angles to the simulator, which tracks the kinematics of the arm. The vision sensing system is composed of several standard webcams that track and filter the fiducial markers on each object in the environment and relays the information to the simulator. The vision software, developed by Santana [59], applies several layers of additional data filtering to handle noise and data intermittence during object tracking.

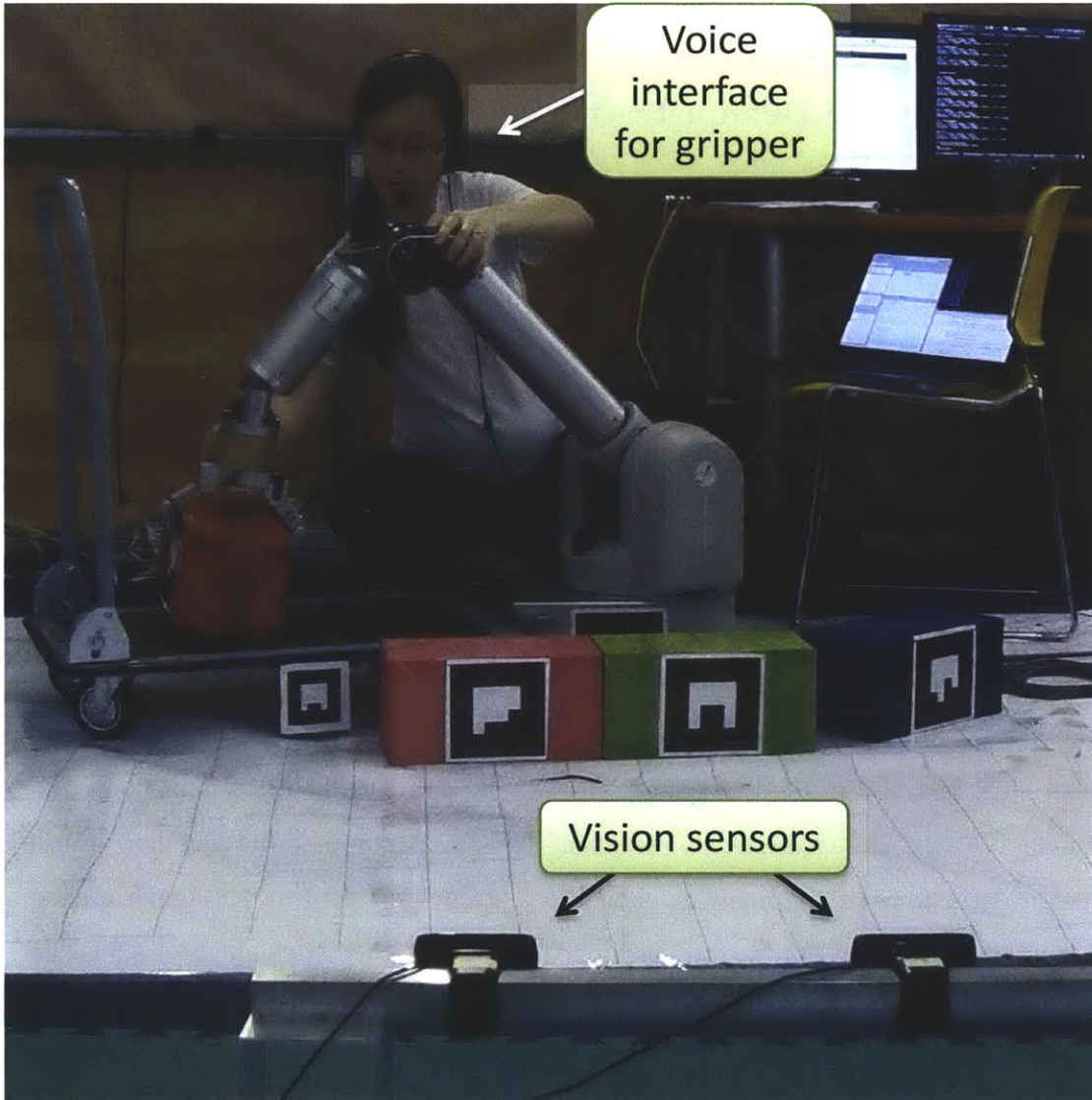


Figure 7-16: Hardware environment setup

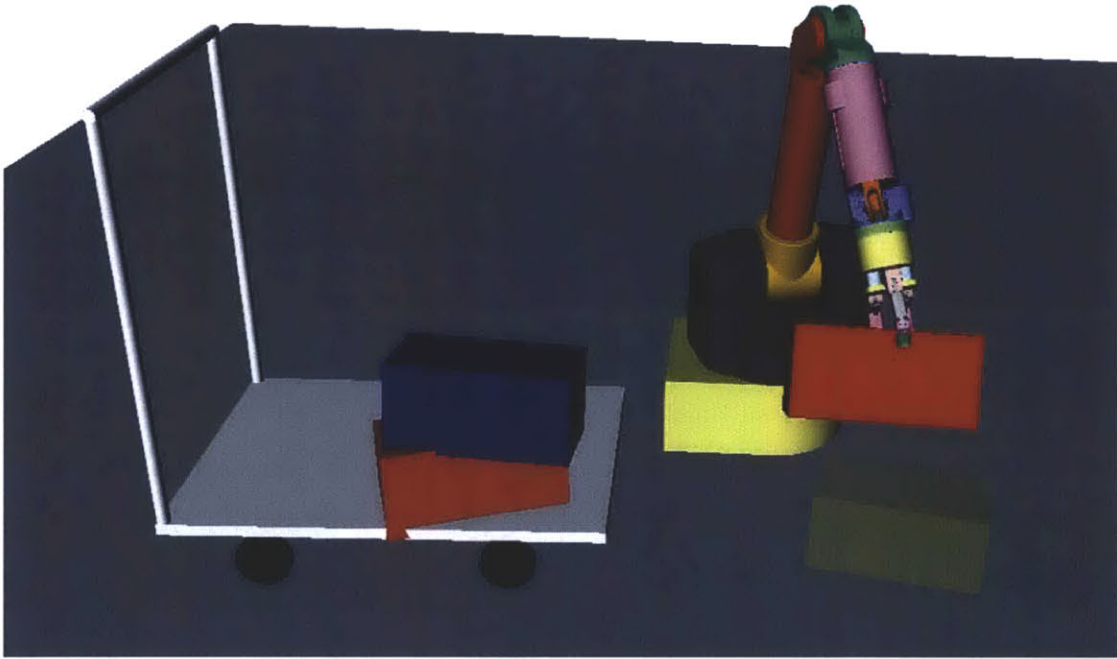


Figure 7-17: Simulation environment

A user performed kinesthetic teaching to demonstrate motion trajectories by putting the WAM in gravity compensation mode and physically manipulating the arm through desired motions. Along the way, the user could command the hand to open or close through the voice interface.

In the hardware environment, the user trained the WAM to perform a construction task involving activities {“reach red block”, “put red block on green block”, “reach pink block”, “put pink block on red block”, “done with pink block”}. During each trial, blocks were initially in different positions, including at different heights as well. All blocks were placed in accessible locations. Nine training samples of the plan were demonstrated to the system, 3 of which were assumed to be pre-segmented keyframe trials. By the nature of this plan, the signal to open or close the hand gripper naturally segments the task into activities. During learning, I assumed this information was available for only 3 of the trials while the the others were used as test cases for auto-segmentation. Trials contained on average 800 data points.

Table 7.5 summarizes the results of auto-segmentation on the construction task

Table 7.5: Results of auto-segmentation using motion variable approach on construction task (6 trials)

	$\mu_t$ (sec)	$\sigma_t$	$\mu_{error}$	$\sigma_{error}$
Red on green, pink on red	0.0032	0.0025	0.2523	0.2030

using the motion variable identification approach. The algorithm was able to achieve 75% segmentation accuracy despite many hardware and sensing issues that caused the training data to be inaccurate. Auto-segmentation computation time averaged around 3 thousandths of a second per trial.

Figure 7-18 shows the resulting auto-generated PFT trajectories in a variety of randomly generated new environment states. In each case, the effector position first reaches toward the red box, moves it in an arc toward a position above the green box, then reaches toward the pink box, and moves it to a higher position above the green box (where the red box was last placed), before moving the arm back up to an arbitrary position. Figures 7-19, 7-20, and 7-21 display the PFT trajectories with the PFT covariances for trials 2, 6, and 9, respectively, from which one can see that the range of allowable deviations is quite large for most of the plan because of large variability during training. Prior to plan learning, there were no pre-trained activities in the task library, so all activities were identified as “unknown” and learned during the plan learning process.

There were several difficulties when using the hardware system. First, vision sensors were frequently unreliable and had narrow range, causing sporadic errors in the tracked positions of the objects. Secondly, the Barrett hand frequently had circuit failures causing the hand to reset. This information would fail to be passed back to the simulator, so often the simulator would believe the arm was holding an object when in reality it was not. Thirdly, the voice sensing was noisy, forcing the user to repeat gripper commands multiple times before the correct action took place. Finally, the simulation often had slight errors in calibration causing simulated objects to be off from reality. Consequently, even when the WAM was able to pick up an object in the real world, the simulation often failed to register it picking up the object. These difficulties in the hardware and interface often occurred so frequently that successfully

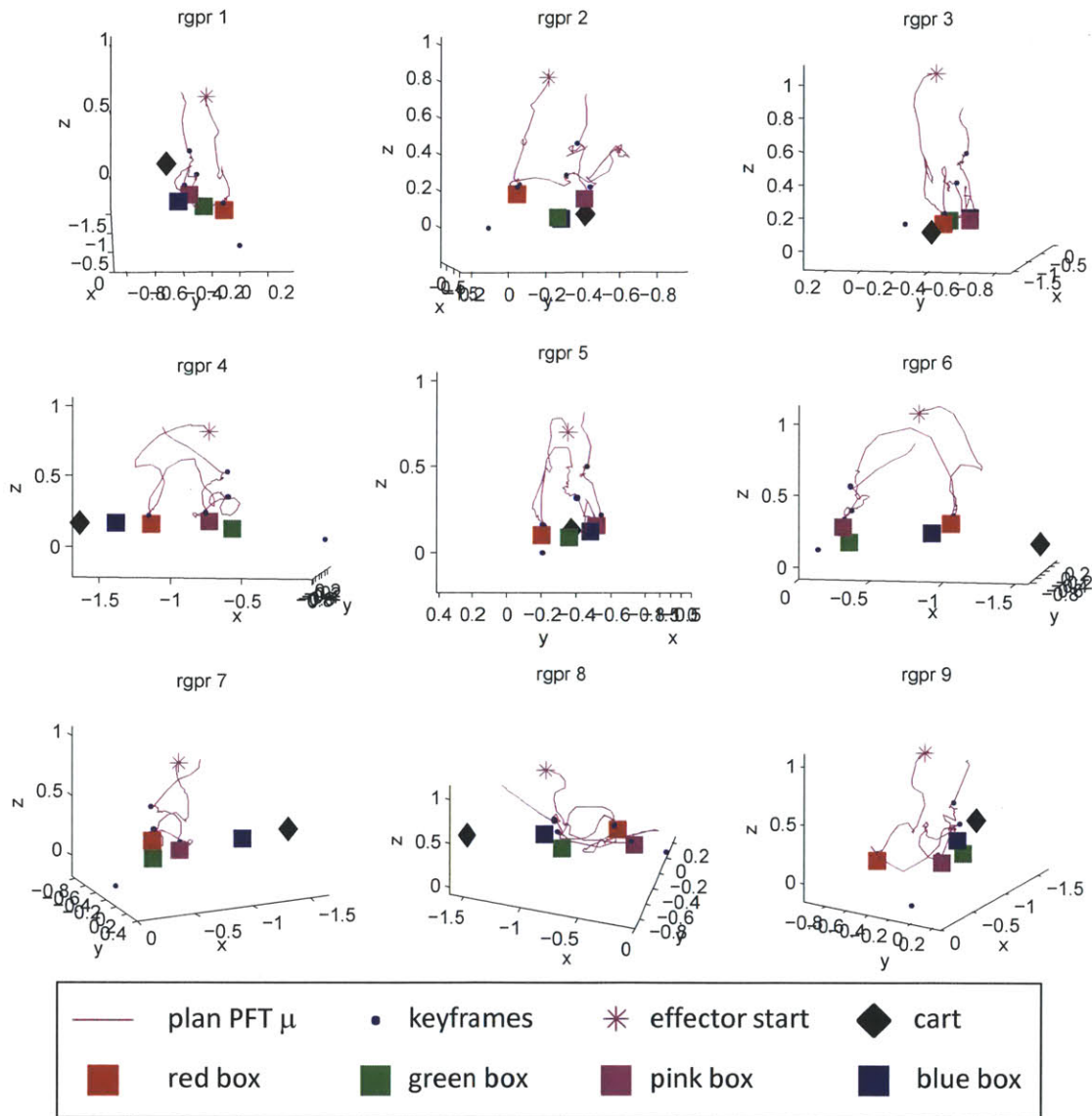


Figure 7-18: Auto-generated PFT trajectories for the “red on green, pink on red” (“rgpr” for short) task plan for various new environment states

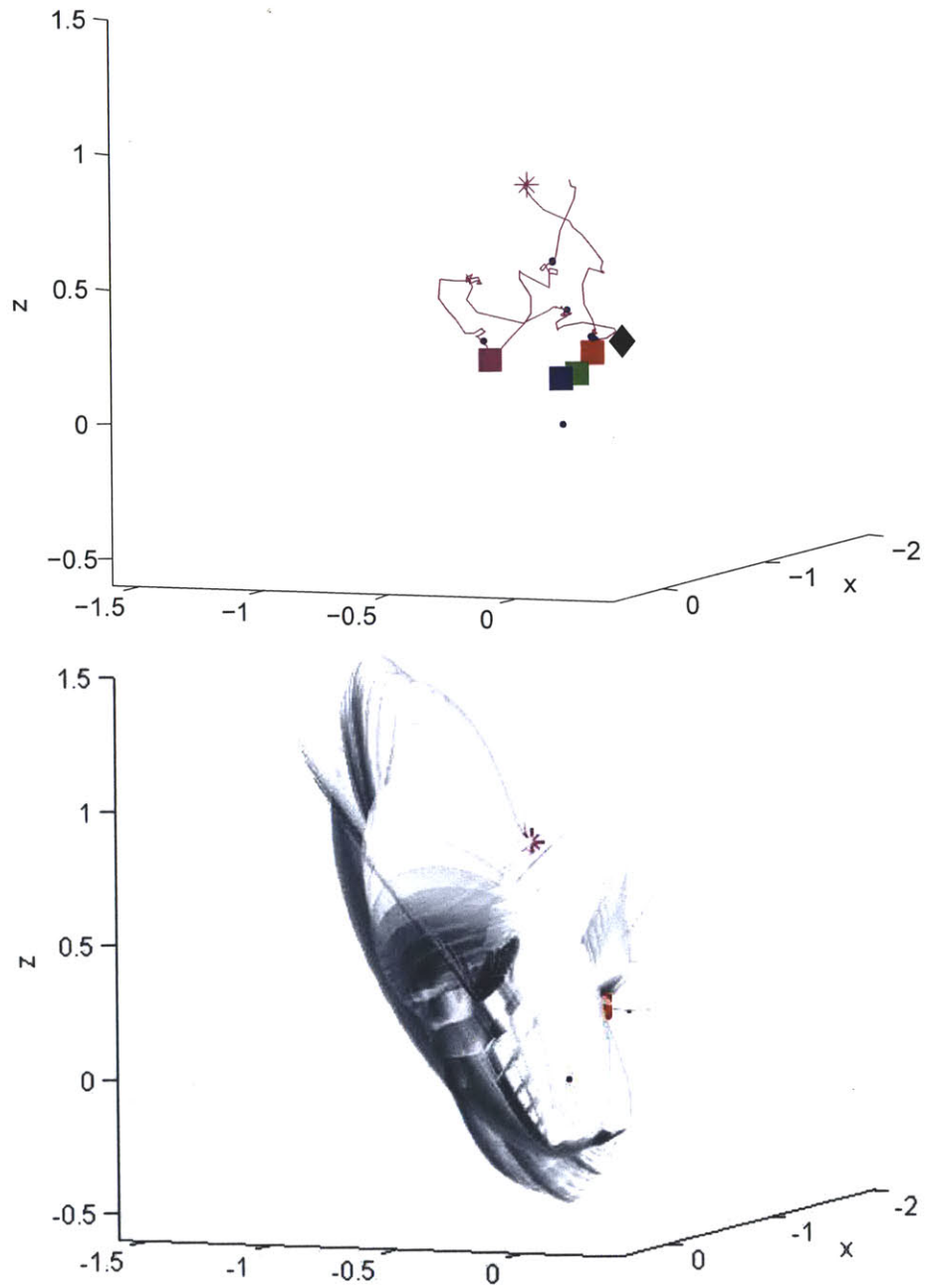


Figure 7-19: Auto-generated trajectory and PFT for the “red on green, pink on red” task plan for new environment state trial 2.



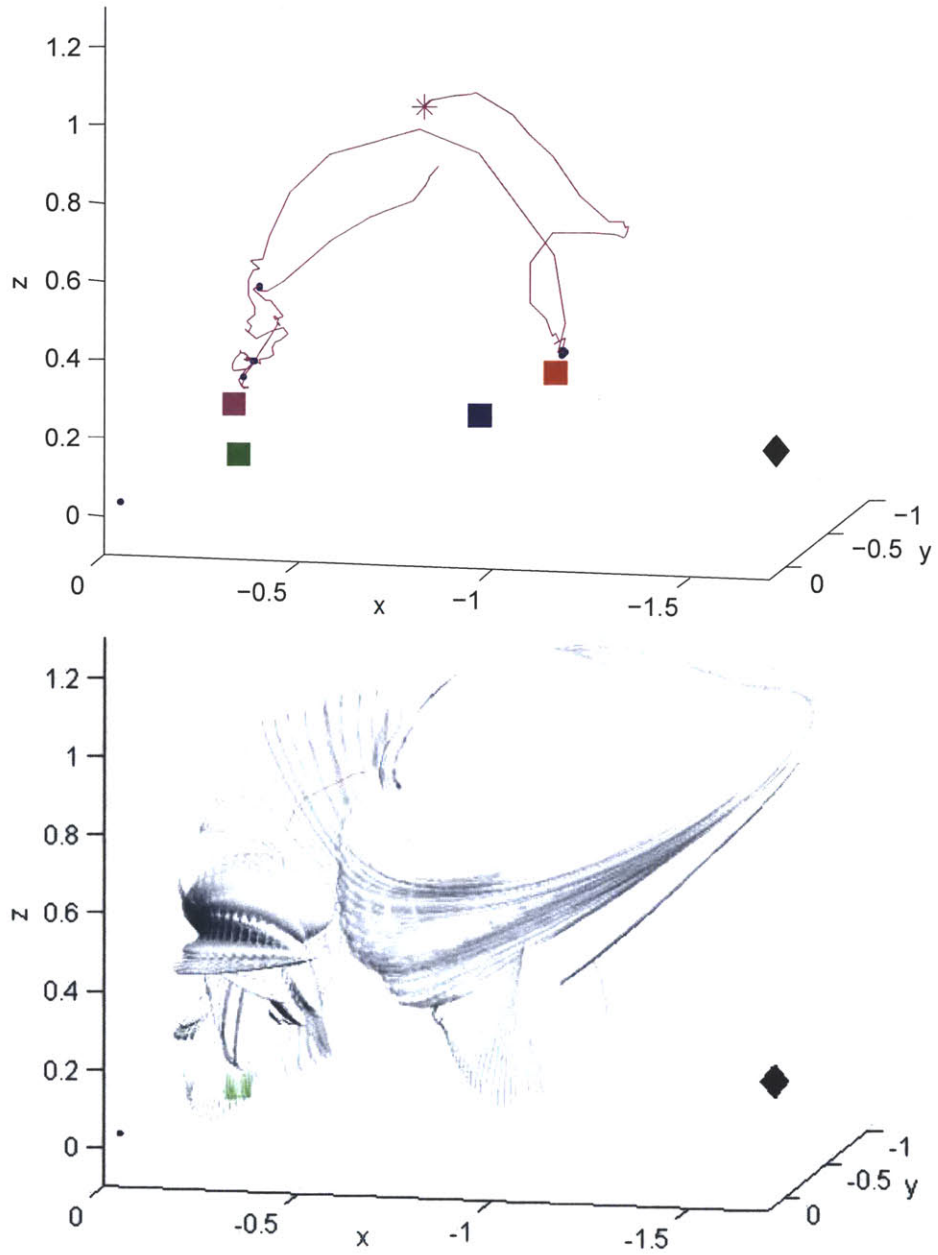


Figure 7-20: Auto-generated trajectory and PFT for the “red on green, pink on red” task plan for new environment state trial 6.

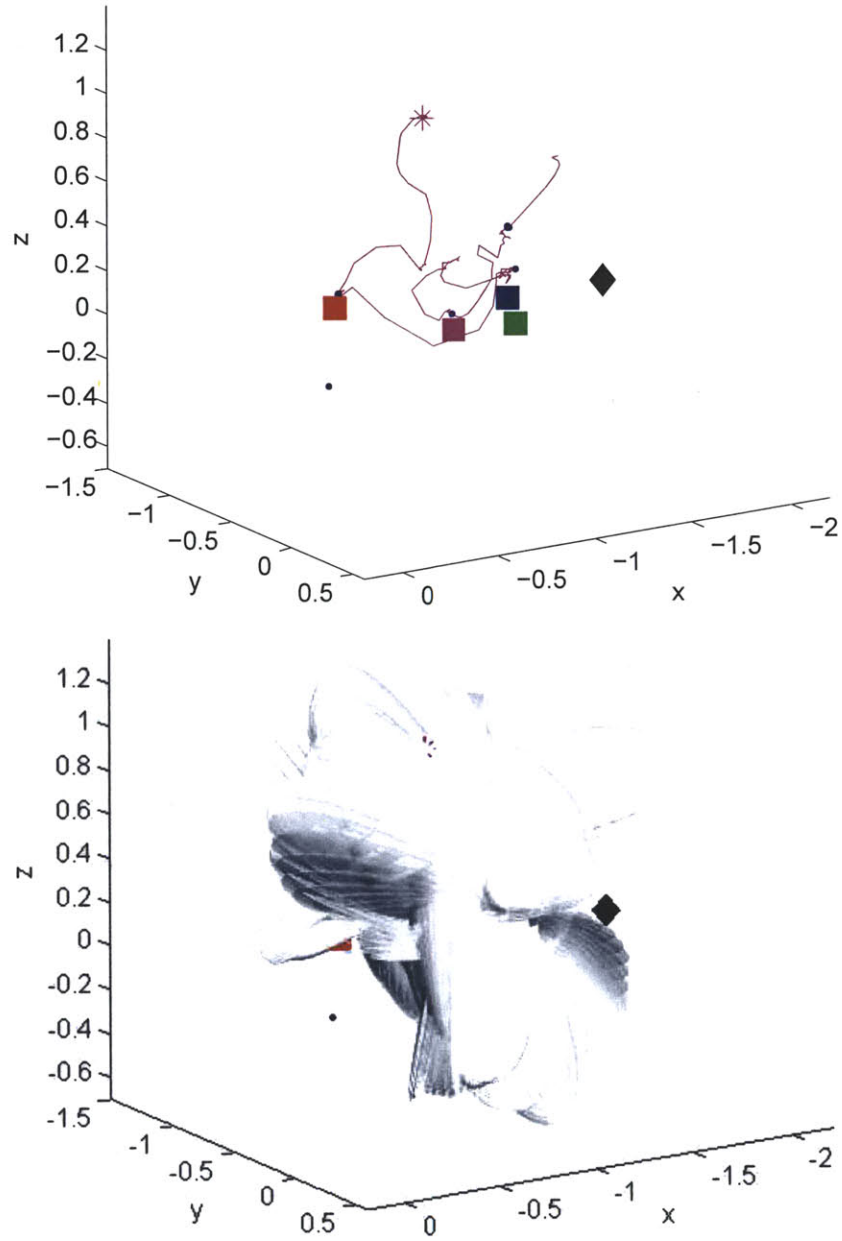


Figure 7-21: Auto-generated trajectory and PFT for the “red on green, pink on red” task plan for new environment state trial 9.

teaching a whole demonstration trial was a challenge, and demonstrated trajectories often contained erroneous movements. Given these challenges, the algorithm was still able to produce reasonable plan trajectories, which is greatly encouraging.

### 7.3 Results Summary

This chapter presented testing and demonstration results of activity learning and recognition in both variable and static two-dimensional environments. In the variable environment, my PFT approach recognized 82% of the test cases in comparison to 55% recognized by the HMM approach, which translates to the PFT approach recognizing 27% more cases than the competition. In the static environment, the PFT approach recognized 97% of the test cases in comparison to 78% recognized by the HMM approach, translating to a 19% improvement over prior art. I also presented a demonstration of real-time recognition on the Barrett Whole Arm Manipulator robot, and demonstrations of autonomous execution on the JPL ATHLETE and Willow Garage PR2 robots.

This chapter next covered test and demonstration results of plan learning. Comparing two different approaches for auto-segmentation revealed that the motion variable inference approach outperforms the recognition optimization approach by orders of magnitudes in computation time while achieving slightly better segmentation accuracy. Furthermore, autonomously generated plan PFTs behave as expected to successfully accomplish the desired tasks. I also presented a demonstration of plan learning on the Barrett Whole Arm Manipulator robot.



# Chapter 8

## Conclusions

### 8.1 Future Extensions

This section presents some ideas for expanding the capabilities of the research in this thesis.

#### 8.1.1 Compliant Execution

In the future, I expect to achieve compliant execution using PFTs. The covariance sequence of a probabilistic flow tube can be provided to a controller as a cost function during compliant execution. In narrow regions of the flow tube, a deviation from the nominal trajectory would have a high cost, whereas the same deviation in a wide region of the flow tube would have a lower cost. On some robots, this may correspond to variation in the robot stiffness as it follows the nominal trajectory.

#### 8.1.2 Obstacle Avoidance

I also foresee achieving obstacle avoidance during execution by overlaying probabilistic flow tubes on potential fields [34, 48], as illustrated in Figure 8-1. The resulting cost map would lead the robot toward the low cost nominal trajectory while guiding it away from the high cost obstacles. In this case the optimal path may not be the same as the original nominal trajectory, as nearby obstacles may push the optimal path

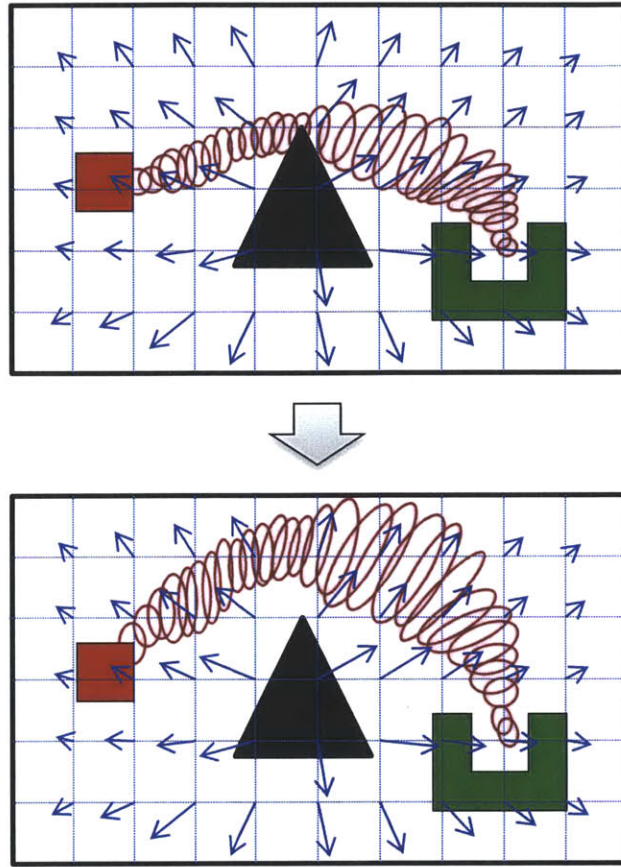


Figure 8-1: Obstacle avoidance may be achieved by overlaying a potential field that pushes away from obstacles in the environment

away from the original trajectory.

### 8.1.3 Plan Recognition

This thesis presented approaches to activity learning, activity recognition, and plan learning. A natural extension of this work is to achieve plan recognition, or detecting in real-time which activity in the plan is being executed by a human teleoperator. This involves determining the likelihood that an activity is transitioning to another. Although outside the scope of the current work, I offer some ideas on how to approach this problem in the future.

I refer to my earlier work in human intent recognition [18] for an approach to recognize which activity in a plan a user is executing. One can incorporate the

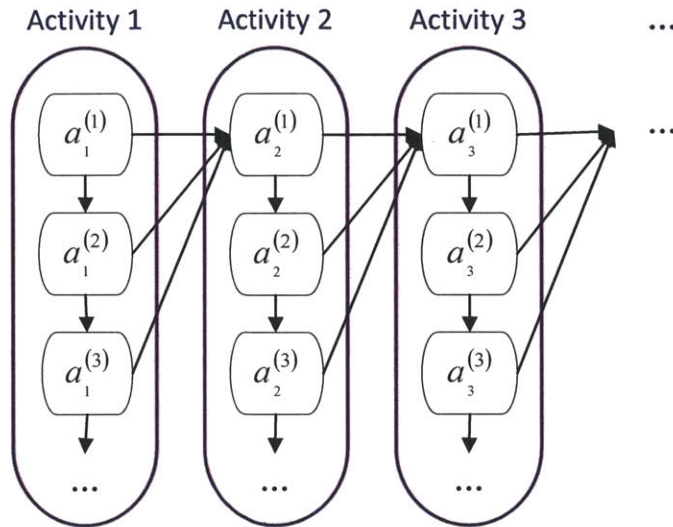


Figure 8-2: To model a sequence of activities with different durations, one can explicitly model each activity at different time steps as separate states. Each activity time slice  $a_r^{(\tau)}$  represents the  $r^{\text{th}}$  activity in the plan trajectory at time step  $\tau$  since the beginning of the activity. At each time step, an activity  $a_r^{(\tau)}$  can transition either to itself  $a_r^{(\tau+1)}$  (downward) or to the next activity  $a_{r+1}^{(\tau)}$  (rightward).

concept of duration distributions to model the range of time each activity takes. A standard hidden Markov model (HMM) can model activity states and transitions among them, but cannot capture arbitrary state duration distributions. Therefore, I propose to model a plan with a non-stationary hidden Markov model (NSHMM) based on work by Sin and Kim [63] and Rabiner [55]. The standard HMM can be expanded to explicitly model self state transitions, as shown in Figure 8-2. Each activity is still modeled as a probabilistic flow tube, from which observation likelihoods can be obtained. By considering activities at different time steps as different states, the non-stationary HMM can be treated as a standard HMM, and standard filtering techniques can be applied to achieve recognition.

Transitions from one activity to another can be based on the duration of activities. For example, if the duration of an activity is on average ten seconds and the current user execution is five seconds in, then the likelihood is high that the user is still executing the same activity and has not transitioned to the next one yet. But if the user execution is already fifteen seconds in, then the likelihood of staying in

the same activity is lower. The change in transition probabilities over time makes the problem non-stationary. The distribution of activity durations can be modeled as gamma distributions  $\Gamma(k_r, \theta_r)$  [13, 14], so that the probability of staying in an activity  $P(a_r^{(\tau+1)}|a_r^{(\tau)})$  reflects the probability that the current activity’s duration is longer than the time elapsed since the activity began, or  $\int_{\tau+1}^{\infty} \Gamma(k_r, \theta_r) du$ . The probability of the user moving to the next activity is then given by  $1 - P(a_r^{(\tau+1)}|a_r^{(\tau)})$ .

Observation likelihoods  $p_{O|\mathcal{L}}(T^{curr}|\ell)$  can be obtained by the same procedure described in Section 5.3 from the learned probabilistic flow tubes for each activity.

Now the standard HMM filtering techniques are available to be applied to the HMM to determine which partial plan a user is most likely executing, and which activity in the plan the user is most likely on. This can be especially useful when identifying subcomponents in a plan is necessary and recognizing the plan as a whole is not enough.

## 8.2 Conclusion

This thesis focuses on the problems of activity learning (identifying significant features of a primitive motion and generalizing its behavior from user demonstration trajectories), activity recognition (determining in real-time the likelihood that a user is currently executing one of several learned activities), and plan learning (generalizing the behavior of a sequence of activities from user demonstration trajectories).

I have presented an approach to learning activities from human demonstration that (1) provides flexibility during execution while robustly encoding a human’s intended motions using a novel representation called a probabilistic flow tube, and (2) automatically determines the relevant features of a motion so that they can be preserved during autonomous execution in new situations.

I have also introduced an approach to real-time motion recognition that (1) leverages temporal information to successfully model motions that may be non-Markovian, (2) provides fast real-time recognition of motions in progress by using an incremental dynamic time warping approach, and (3) employs the probabilistic flow tube rep-



resentation that enables our method to recognize learned motions despite varying environment states.

Finally, I developed an approach to learn combinations of activities represented as PFT plans, given longer user demonstration sequences, that (1) automatically determines where activities should be segmented in a sequence and (2) identifies and learns previously unseen activities embedded in a sequence.

I have validated the approach in a two-dimensional environment and demonstrated the algorithms on several different robotic platforms. The algorithms perform more favorably over prior art in the domain of robotic manipulation tasks, and are especially useful for generating robust, humanlike, temporally ordered, and trajectory oriented motions.



# Bibliography

- [1] Pieter Abbeel, Dmitri Dolgov, Andrew Y. Ng, and Sebastian Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. In *IROS*, 2008.
- [2] Baris Akgun, Maya Cakmak, Jae Wook Yoo, and Andrea Lockerd Thomaz. Trajectories and keyframes for kinesthetic teaching: a human-robot interaction perspective. In *HRI*, 2012.
- [3] Aris Alissandrakis, Chrystopher L. Nehaniv, Kerstin Dautenhahn, and Joe Saunders. An approach for programming robots by demonstration: Generalization across different initial configurations of manipulated objects. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 2005.
- [4] Robert Ambrose. Development and deployment of robonaut 2 to the international space station. In *ICRA*, 2011.
- [5] Brian Anthony. *Video based system monitoring*. PhD thesis, Massachusetts Institute of Technology, 2006.
- [6] Brenna Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(3):469–483, 2009.
- [7] Christopher G. Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*, pages 12–20, 1997.
- [8] Jaron Blackburn and Eraldo Ribeiro. Human motion recognition using isomap and dynamic time warping. In *Workshop on Human Motion*, pages 285–298, 2007.
- [9] Sylvain Calinon, Florent D’halluin, Eric Sauser, Darwin Caldwell, and Aude Billard. Learning and reproduction of gestures by imitation: An approach based on hidden markov model and gaussian mixture regression. *IEEE Robotics and Automation Magazine*, 17(2):44–54, 2010.
- [10] Matthew Carey, Eric Kurz, Joshua Matte, and Timothy Perrault. Novel EOD robot design with a dexterous gripper and intuitive teleoperation. Technical report, Worcester Polytechnic Institute, 2011.

- [11] Thomas Cederborg, Ming Li, Adrien Baranes, and Pierre-Yves Oudeyer. Incremental local online gaussian mixture regression for imitation learning of multiple tasks. In *IROS*, 2010.
- [12] Adam Coates, Pieter Abbeel, and Andrew Ng. Apprenticeship learning for helicopter control. *Communications of the ACM*, 52(7):97–105, July 2009.
- [13] Carroll Croarkin and Paul Tobias, editors. *NIST/SEMATECH e-Handbook of Statistical Methods*, chapter 1.3.6.6.11. Gamma Distribution. National Institute of Standards and Technology, July 2006.
- [14] Philip J. Davis. Gamma function and related functions. In Milton Abramowitz and Irene A. Stegun, editors, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, chapter 6. Superintendent of Documents, U.S. Government Printing Office, Washington, DC, 1972.
- [15] Rosen Diankov and James Kuffner. OpenRAVE: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Carnegie Mellon University, 2008.
- [16] M. A. Diftler, J. S. Mehling, P. A. Strawser, W. R. Doggett, and I. M. Spain. A space construction humanoid. In *IEEE-RAS International Conference on Humanoid Robotics*, pages 92–97, 2005.
- [17] Simon Dixon. An on-line time warping algorithm for tracking musical performances. In *IJCAI*, 2005.
- [18] Shuonan Dong. Unsupervised learning and recognition of physical activity plans. Master’s thesis, Massachusetts Institute of Technology, 2007.
- [19] Shuonan Dong, Patrick R. Conrad, Julie A. Shah, Brian C. Williams, David S. Mittman, Michel D. Ingham, and Vandana Verma. Compliant task execution and learning for safe mixed-initiative human-robot operations. In *AIAA Infotech*, 2011.
- [20] Shuonan Dong and Brian Williams. Motion learning in variable environments using probabilistic flow tubes. In *ICRA*, 2011.
- [21] Shuonan Dong and Brian Williams. Learning and recognition of hybrid manipulation motions in variable environments using probabilistic flow tubes. *Journal of Social Robotics*, 2012.
- [22] Fitriani. *Multiscale Dynamic Time and Space Warping*. PhD thesis, Massachusetts Institute of Technology, 2008.
- [23] Jordan Frank, Shie Mannor, and Doina Precup. Activity and gait recognition with time-delay embeddings. In *AAAI*, 2010.

- [24] Gary Gilbert and Michael Beebe. United States Department of Defense research in robotic unmanned systems for combat casualty care. Technical Report ADA526596, NATO/RTO, 2010.
- [25] Raffay Hamid, Siddhartha Maddi, Amos Johnson, Aaron Bobick, Irfan Essa, and Charles Isbell. A novel sequence representation for unsupervised analysis of human activities. *Artificial Intelligence*, 173(14):1221–1244, 2009.
- [26] Andreas Hofmann and Brian Williams. Exploiting spatial and temporal flexibility for plan execution of hybrid, under-actuated systems. In *AAAI*, 2006.
- [27] Kaijen Hsiao and Tomas Lozano-Perez. Imitation learning of whole-body grasps. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2006.
- [28] Xuedong Huang, Fileno Allewa, Hsiao wuen Hon, Mei yuh Hwang, and Ronald Rosenfeld. The SPHINX-II speech recognition system: An overview. *Computer, Speech and Language*, 7:137–148, 1992.
- [29] Anthony Jameson. *Adaptive interfaces and agents*, chapter 15, pages 305–330. L. Erlbaum Associates Inc., 2002.
- [30] Kui Jia and Dit-Yan Yeung. Human action recognition using local spatio-temporal discriminant embedding. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition CVPR 2008*, pages 1–8, 23–28 June 2008.
- [31] Kazuhiko Kawamura, Phongchai Nilas, Kazuhiko Muguruma, Julie A. Adams, and Chen Zhou. An agent-based architecture for an adaptive human-robot interface. In *36th Hawaii International Conference on System Sciences*, 2003.
- [32] Shinji Kawatsuma, Mineo Fukushima, and Takashi Okada. Emergency response by robots to fukushima-daiichi accident - summary and lessons learned. *Industrial Robot: An International Journal*, 39(5), 2012.
- [33] Zunaid Kazi, Shoupu Chen, Matthew Beitler, Daniel Chester, and Richard Foulds. Multimodal hci for robot control: Towards an intelligent robotic assistant for people with disabilities. In *AAAI*, 1995.
- [34] Bruce H. Krogh. A generalized potential field approach to obstacle avoidance control. In *International Robotics Research Conference*, Bethlehem, PA, 1984.
- [35] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, and Paul E. Wright. Convergence properties of the nelder-mead simplex method in low dimensions. *SIAM Journal of Optimization*, 9(1):112–147, 1998.
- [36] Dongheui Lee and Christian Ott. Incremental kinesthetic teaching of motion primitives using the motion refinement tube. *Autonomous Robots*, 31(2-3):115–131, 2011.

- [37] Steven Levine. Robust robotic task execution for collaborative manufacturing environments. Master's thesis, Massachusetts Institute of Technology, 2012.
- [38] Hui Li and Brian Williams. Generative planning for hybrid systems based on flow tubes. In *ICAPS*, 2008.
- [39] Rodney A. Martin, Kevin R. Wheeler, Mark B. Allan, and Vytas SunSpiral. Optimized algorithms for prediction within robotic tele-operative interfaces. Technical report, NASA/TM-2010-216417, 2010.
- [40] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 37(3):311 – 324, 2007.
- [41] David Mittman, Jeffrey Norris, Mark Powell, Recaredo Torres, and Christopher McQuin. Lessons learned from All-Terrain Hex-Limbed Extra-Terrestrial Explorer robot field test operations at Moses Lake Sand Dunes, Washington. In *AIAA SPACE*, 2008.
- [42] M. A. Moni and A. B. M. Shawkai Ali. HMM based hand gesture recognition: A review on techniques and approaches. In *IEEE ICCSIT*, 2009.
- [43] Manuel Muehlig, Michael Giengerand, Sven Hellbachand, Jochen Steil, and Christian Goerick. Task-level imitation learning using variance-based movement optimization. In *ICRA*, 2009.
- [44] C. S. Myers, L. R. Rabiner, and A. E. Rosenberg. Performance trade-offs in dynamic time warping algorithms for isolated word recognition. *Journal of the Acoustical Society of America*, 66(S1):S34–S35, 1979.
- [45] Takayuki Osa, Chrostoph Staub, and Alois Knoll. Framework of automatic robot surgery system using visual servoing. In *IROS*, 2010.
- [46] Sarah Osentoski, Victoria Manfredi, and Sridhar Mahadevan. Learning hierarchical models of activity. In *IROS*, Sendai, Japan, 2004.
- [47] Sarah Osentoski, Benjamin Pitzer, Christopher Crick, Graylin Jay, Shuonan Dong, Daniel Grollman, Halit Bener Suay, and Odest Chadwicke Jenkins. Remote robotic laboratories for learning from demonstration. *Journal of Social Robotics*, 2012.
- [48] Dae-Hyung Park, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *IEEE-RAS International Conference on Humanoid Robots*, 2008.
- [49] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *ICRA*, 2009.

- [50] Xavier Pennec. Computing the mean of geometric features – application to the mean rotation. Technical Report 3371, Institut National de Recherche en Informatique et en Automatique, 1998.
- [51] Fernando Pereda, Hector Garcia de Marina, and Juan Jimenez. Towards automatic oil spill confinement with autonomous marine surface vehicles. In *IEEE Oceans*, 2011.
- [52] Richard Alan Peters and Christina L. Campbell. Robonaut task learning through teleoperation. In *ICRA*, 2003.
- [53] Patrick Pfaff, Christian Plagemann, and Wolfram Burgard. Gaussian mixture models for probabilistic localization. In *IEEE International Conference on Robotics and Automation*, 2008.
- [54] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Robotics*, 2009.
- [55] Lawrence R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *IEEE*, 77(2), February 1989.
- [56] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
- [57] Marcia Riley and Gordon Cheng. Extracting and generalizing primitive actions from sparse demonstration. In *IEEE-RAS International Conference on Humanoid Robots*, 2011.
- [58] Stan Salvador and Philip Chan. FastDTW: Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561 – 580, 2007.
- [59] Pedro H. R. Q. Santana. Stochastic filtering for hybrid systems and its applications to aerial robotics. Master’s thesis, University of Brasilia, 2011.
- [60] Gideon Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.
- [61] Pavel Senin. Dynamic time warping algorithm review. Technical report, University of Hawaii at Manoa, 2008.
- [62] Alexander Serenko. The use of interface agents for email notification in critical incidents. *International Journal of Human-Computer Studies*, 64:1084–1098, 2006.
- [63] Bongkee Sin and Jin H. Kim. Nonstationary Hidden Markov Model. *Signal Processing*, 46(1):31–46, September 1995.

- [64] Viytas SunSpiral, Kevin R. Wheeler, Mark B. Allan, and Rodney Martin. Modeling and classifying six-dimensional trajectories for teleoperation under a time delay. In *AAAI Spring Symposium*, 2006.
- [65] Mostafa Syiam, Mostafa Abd El-Aziem, and Mohamed El-Menshawy. Adagen: Adaptive interface agent for x-ray fracture detection. *International Journal of Computing & Information Sciences*, 2(3):143–148, 2004.
- [66] Russ Tedrake. LQR-Trees: Feedback motion planning on sparse randomized trees. In *Robotics: Science and Systems*, 2009.
- [67] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.
- [68] Bill Tomlinson, Eric Baumer, Man Lok Yau, Paul Mac Alpine, Lorenzo Canales, Andrew Correa, Bryant Hornick, and Anju Sharma. Dreaming of adaptive interface agents. In *ACM Conference on Human Factors in Computing Systems*, 2007.
- [69] William Townsend and Jeffrey Guertin. Teleoperator slave - WAM design methodology. *Industrial Robot: An International Journal*, 26(3):167–177, 1999.
- [70] Sachiko Wakabayashi, Darby F. Margruder, and William Bluethmann. Test of operator endurance in the teleoperation of an anthropomorphic hand. In *SAIRAS*, 2003.
- [71] Liang Wang and David Suter. Analyzing human movements from silhouettes using manifold learning. In *IEEE International Conference on Video and Signal Based Surveillance*, 2006.
- [72] Xiaogang Wang and Xiaoou Tang. Bayesian face recognition based on gaussian mixture models. In *International Conference on Pattern Recognition*, 2004.
- [73] Zheshen Wang and Baoxin Li. Human activity encoding and recognition using low-level visual features. In *IJCAI*, 2009.
- [74] Brian Wilcox. ATHLETE: A cargo and habitat transporter for the moon. In *IEEE Aerospace conference*, 2009.
- [75] Jie Yang, Yangsheng Xu, and C. S. Chen. Human action learning via hidden Markov model. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 27(1):34–44, 1997.
- [76] Feng Zhao. *Automatic Analysis and Synthesis of Controllers for Dynamical Systems Based On Phase-Space Knowledge*. PhD thesis, Massachusetts Institute of Technology, 1992.