Inferring User Intent for Learning by Observation

Kevin R. Dixon

Thesis Committee:
Pradeep K. Khosla, Advisor
Bruce H. Krogh
Maja J. Matarić
Christiaan J.J. Paredis
Sebastian B. Thrun

Submitted to the Department of Electrical and Computer Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy at

Carnegie Mellon University
Pittsburgh, PA 15213

23 January, 2004

To my parents.

Abstract

Despite the numerous advances in human-robot interaction, most development systems still require that users have substantial knowledge of procedural-programming techniques as well as the specific robot system at hand. For the vast majority of the population, this effectively precludes the use of robots in most cases. If robots are to make headway into everyday situations, then users must be able to program robots in a more natural and intuitive manner. This dissertation explores a method of programming robots to automate motor tasks by inferring the intent of users based on demonstrations of a task. In order to understand such a system, we decompose it into simpler components: modeling user subgoal selection and the response of users to different conditions.

We have developed a learning algorithm that constructs a statistical model of user subgoal selection based on previous observations. After deriving the algorithm, we provide theoretical guarantees about the model. To validate the theoretical underpinnings of the algorithm, we isolate the performance of modeling user subgoal selection by removing extraneous factors such as sensor noise and environment considerations. To this end, we present experimental results in predicting the waypoints of manipulator-robot programs. We show that the algorithm produces submillimeter prediction errors on real-world data.

We hypothesize about the response of users to different conditions with a model of sequenced linear dynamical systems. We first develop the concept that a single dynamical system can represent a simple trajectory using a closed-form least-squares procedure. Since our approach is based on the least-squares principle, it is simple to combine multiple demonstrations, giving the system a better generalization of "what the user would have done" in novel conditions. To represent more complicated trajectories, we segment it and represent each segment by a single dynamical system.

These algorithms form the core of a mobile-robot system that learns motor skills by observing users demonstrating a task. From these observations, the system extracts task subgoals and automatically associates them with objects in the environment, so that as the objects move, the subgoals are updated accordingly. This system can learn from multiple demonstrations, as well as demonstrations performed in different environment configurations. In laboratory experiments, we show that the system accurately infers user intent.

Acknowledgments

This research would not have been possible without the support and encouragement of many people, to whom I owe a great debt of gratitude. I would like to thank my advisor, Pradeep Khosla, for his support over the years. He gave me the intellectual freedom to pursue my changing research interests, and his ideas have inspired this work. I have benefited immeasurably from discussions with my thesis committee: Chris Paredis, Bruce Krogh, Maja Matarić, and Sebastian Thrun. In particular, Chris Paredis helped set the high-level goals of this work and Bruce Krogh provided invaluable feedback for the controls aspects.

I also received a lot of help over the years from various people at CMU. John Dolan has been a constant source of guidance, assistance, and inspiration. He always found time to answer my endless stream of questions and his insight improved every aspect of this work. Mike Seltzer was a great sounding board for the ideas that bounced around my head. My officemates Yaron Rachlin and Arvind Seshadri have given me pointers and laughs over the years, and Debbie Scappatura always offered a helping hand.

During the course of my research, I have benefited from interacting with many people from ABB Robotics and Arc-Welding Systems in Sweden. Martin Strand, Ralph Sjöberg, and Henrik Lander provided feedback about some early ideas regarding Predictive Robot Programming. Mikael Svensson, Carl "Älgjägare" Ericsson, Magnus Sjöholm, and Karl Bergsten taught me how to arc-weld and spent countless hours answering questions about real-world robots. Per-Ulrik Eriksson provided the offline programs in Chapter 4 and showed me how offline-programming systems are actually used. Discussions with these people, and many others, helped to ensure that this work was not just a theoretical exercise.

Dan Gaugel, Dana Hilinski, George Lopez, Rich Malak, Spence Oliver, Mike Seltzer, Kenny Tateosian, Stef Tomko, and Jay Wylie all gave me help and, most importantly, reminded me that there is more to grad school than research. They have been an invaluable source of fun and laughs and their friendship alone would have made this experience worthwhile. My brother, Brian, has always been a dependable friend and skiing partner, in addition to all the advice and regrounding.

My wife, Meredith, has given me love and support throughout. For every success in research, there are innumerable false starts and failures. Meredith shared in the highs and offered constant encouragement through the lows. Her love means more than words can describe.

Finally, and most of all, I would like to thank my parents for their unwavering belief in me. They have given me unconditional love, support, and encouragement for my entire life. I cannot express my gratitude to them for selflessly providing me with every opportunity imaginable.

Contents

List of Figures

List of Tables

Notation

Symbol	Meaning
x	scalar
$\boldsymbol{x}$	vector
$\boldsymbol{C}$	matrix
$\widehat{x}$	estimate of x
$\widehat{x}^*$	maximum-likelihood estimate of x
$\mathbb{R}$	set of reals
$\mathbb{Z}$	set of integers
$\triangleq$	quantity definition
$\equiv$	equivalence
$\dot{=}$	change in notation
$:=$	algorithmic assignment
$\cong$	algebraic approximation
$\approx$	numeric approximation
$\propto$	proportionate
$\tilde{\propto}$	approximately proportionate
$\mathrm{p}(\cdot)$	probability density function (pdf)
$\mathrm{P}(\cdot)$	probability mass function (pmf)
$\Pr\{\cdot\}$	event probability
$\mathrm{E}_c\{x\}$	expected value of x under random variable c
$\sim$	drawn from distribution
$\mathcal{M}(\boldsymbol{\mathcal{A}})$	multiset over the set $\boldsymbol{\mathcal{A}}$
$\langle\boldsymbol{\mathcal{A}}\rangle$	sample mean of the (multi)set $\boldsymbol{\mathcal{A}}$
$\lvert\boldsymbol{\mathcal{A}}\rvert$	cardinality of the (multi)set $\boldsymbol{\mathcal{A}}$
$\setminus$	(multi)set difference
$\lVert\cdot\rVert_2$	Euclidean or L_2 vector norm
$\lVert\cdot\rVert_{\boldsymbol{C}}$	Mahalanobis or weighted-Euclidean vector norm
$\lVert\cdot\rVert_F$	Frobenius matrix norm
$\mathrm{tr}[\cdot]$	matrix trace
$\mathcal{N}(\boldsymbol{u}, \boldsymbol{\Sigma})$	Gaussian distribution with mean $\boldsymbol{u}$ and variance $\boldsymbol{\Sigma}$
$\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{A}}, \boldsymbol{y})$	Similarity measure defined as $\sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{A}}} \lVert\boldsymbol{x}-\boldsymbol{y}\rVert_{\boldsymbol{C}}^2$

Chapter 1

Introduction

Despite the numerous advances in human-computer interaction, most development systems still require that users convey knowledge to robots through procedural-programming techniques. For the vast majority of the population, this transfer of knowledge is limited by the programming expertise of the user. Most users have neither the experience nor the inclination to program robots to perform their tasks. In industrial settings, many companies do not have the resources to automate production; the down-time required to reprogram the facilities may interrupt production and the expense required to obtain programming expertise may be too great.

The goal of this work is to create a system that increases the knowledge transferred between users and robots. Consider the case of industrial robotic arc-welding. When a new product arrives that the factory is to produce, a skilled manual arc-welder typically practices welding the product together by hand to determine the best sequence of welds. The arc-welder then collaborates with a computer programmer to write the procedural code to create a robot program that automates the process that the arc-welder has performed by hand. The conventional design process is shown in Figure 1.1. Clearly, this process could be streamlined by creating a system that observes the arc-welder performing the task by hand and synthesizes the information to create the robot program needed to automate production. This approach is known as Learning By Observation (LBO), and the design process with an LBO system is shown in Figure 1.2. Such a system allows users to “program” automation tasks by demonstration, instead of writing software in a conventional procedural-programming language.

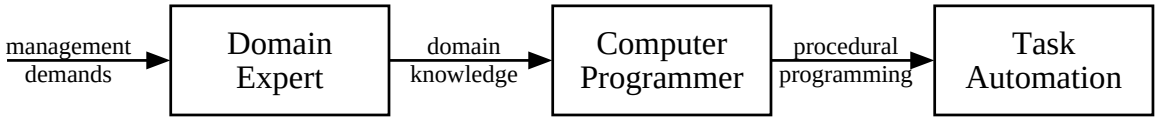


Figure 1.1: The typical industrial task-automation design loop.

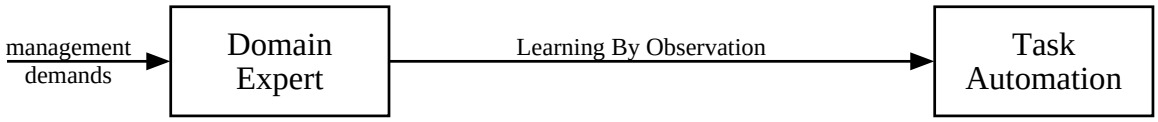


Figure 1.2: The industrial task-automation design loop with Learning By Observation.

1.1 Goals

The goal of this work is to develop a framework to automate low-level tasks by observing user demonstrations. Informally, a low-level task is one where an observable *state*[1] sequence is sufficient to extract the necessary information needed to complete the task. This information then maps onto a sequence of motor commands specifying *how* a robot should achieve the desired goal. For this reason, we call these motor-skill tasks, which form the foundation of industrial automation (Kurtz & Kuper, 1986). The vast majority of industrial robotic tasks, such as painting, welding, palletizing, foundry work, and fixturing fall into this category. A system that can automate motor-skill tasks by observing users performing tasks manually would dramatically improve the industrial-automation design process. The *goal* of a motor-skill task is a desired state, which may be a function of the environment and other factors. Furthermore, the goal can usually be decomposed into a sequence of subgoals, whose relative ordering may, or may not, be important. Consider the case of welding the joints of a rectangle. If the goal is to weld the object together, then a subgoal might be the welding of a particular corner. The position of the weld is determined by the size, location, and orientation of the rectangle, making the desired subgoal state a function of the environment. Since the goal of a motor-skill task may be modulated, the sequence of motor commands extracted from user demonstrations can be a function of sensory information.

The fundamental question in this work is:

- *"Can we design a system that automates motor-skill tasks from observations of user demonstrations?"*

To answer this question, we must first specifically describe the requirements to which the system must adhere.

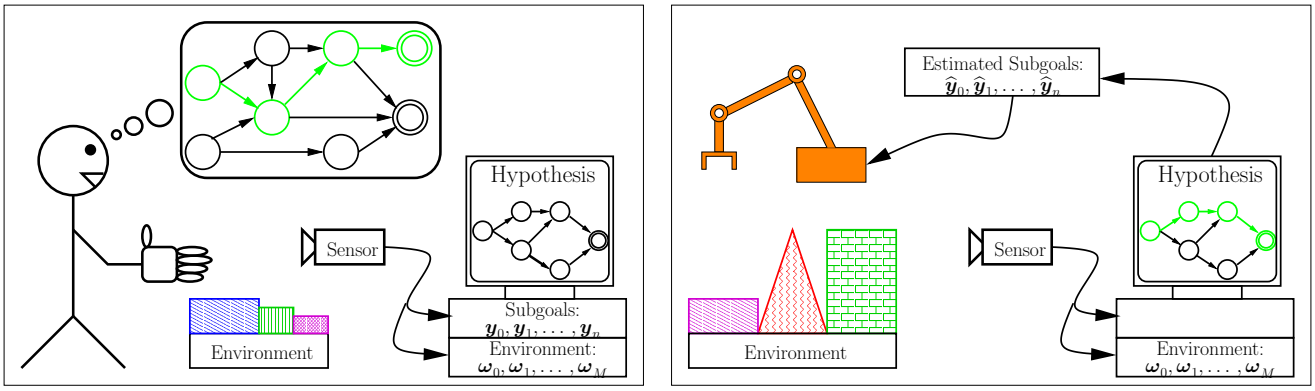
[1] By observable state, we mean those state variables that can be inferred from an observed trajectory. This definition permits temporary occlusion (unobservability) of relevant state variables during demonstration.

1.2 System Requirements

From the highest level, an LBO system "watches" users demonstrating a task. The LBO system then synthesizes that information to perform the task in the future. This implies that the LBO system has access to some actuation system capable of reproducing the necessary commands, e.g., a robot. This description naturally leads to some requirements that can be imposed.

With any system, there is a trade-off between ease of use and ease of design, and LBO is no exception. Generally speaking, placing more burden on users simplifies the design of an LBO system. Requiring users to provide detailed information about a demonstration, or integrating sophisticated instrumentation, may help to create a better-performing LBO system. However, if taken too far, this could defeat the purpose of LBO, which is simplifying task automation. Consequently, we have adopted the philosophy that the method by which the LBO system observes users should be as unobtrusive as possible. Our LBO system employs a scanning laser range finder to observe user demonstrations, which produces two-dimensional polar-coordinate readings at a predetermined horizontal plane. Such a sensor greatly simplifies the demonstration process, as we require no additional instrumentation of users or the environment, but it does increase the burden on the LBO system. For example, a laser is a line-of-sight sensor, meaning that the LBO system must deal with temporary occlusion of users. Furthermore, our learning algorithms are *unsupervised*. That is, we do not require users to provide any additional information about a task demonstration; our LBO system learns to automate tasks from observations alone, without user intervention. This reduces the burden on users to think in terms of what information our LBO system requires. To compensate for this, in an anthropomorphic sense, our learning algorithms must infer the *intent* of users from observations gleaned from task demonstrations. It is no secret that getting inside the mind of a human is never easy.

We also require that the LBO system be able to perform tasks in different environments. This implies, first of all, that the LBO system must be able to sense the state of the environment. When presented with novel conditions, the system must ultimately determine a single sequence of motor commands to perform a task. Since there may be many possibilities to complete a task, such a system is inherently biased and gives preference to some possibilities over others. Because the LBO system is emulating user actions, it is hypothesizing about what the user would have done under these conditions. This implies that the LBO system has a model of how the behavior of users changes with different conditions. To infer user intentions, the LBO system should be able to learn from multiple demonstrations of a task. Furthermore, to help the LBO system better understand how the behavior of users changes, it should be able to learn from demonstrations performed under different conditions.



(a) (b)

Figure 1.3: Building a hypothesis of user intent and performing the task under different conditions.

1.2.1 Implementation

We assume that users have a desired sequence of subgoals in mind when performing a task. In this work, a *subgoal* is a vector in the state-space of users, such as their $\{x, y\}$ position. These subgoals are subsequently modified to compensate for changes in the environment and other external factors. While subgoals are invariant for the task, the specific actions that users perform to achieve the subgoals are specific to the current operating conditions. The purpose of the LBO system is to provide the necessary sequence of motor commands specifying *how* to achieve the subgoals in the requisite order. To this end, the LBO system extracts a set of subgoals from observations of user demonstrations and develops a hypothesis about the internal user model, as shown in Figure 1.3(a). When asked to automate a task, the LBO system computes from the estimated user model the necessary sequence of actions, or estimated subgoals, that the user would have performed under those same conditions, Figure 1.3(b).

1.3 Methodology

LBO systems are inherently complex. To understand the performance of any complex system, it is necessary to decompose the system into smaller components. To this end, we attempt to isolate certain salient problems of LBO: modeling user subgoal selection and the response of users to different conditions.

1.3.1 Modeling User Subgoal Selection

As described in the following chapters, a natural model of user subgoal selection is a Continuous-Density Hidden Markov Model (CDHMM). However, one of the hallmarks of LBO is the extreme scarcity of data upon which to train the system. This is because LBO systems gather observations from human activity, and each demonstration may be of significant duration. As such, it may require hours, days, or months to obtain a sufficient number of examples for successful learning. Our CDHMM learning algorithm is designed to cope with the characteristic difficulties of LBO: real-time operation and limited training data. The hypothesis is then as follows:

Hypothesis 1.1. *CDHMMs are an effective model of user subgoal selection.*

The modeling of subgoal selection from observations of user demonstrations with a CDHMM is a mathematical abstraction of the underlying phenomenon. Testing the hypothesis that CDHMMs are an effective model of user subgoal selection, then, requires two steps. The first phase involves theoretical guarantees about the model, i.e., *assuming* that users create subgoals as a CDHMM, what provable statements can be made about the performance of the model? These statements take the form of propositions and answer the questions:

- What tasks can be automated?
- What computing resources are needed to estimate the model?
- How many demonstrations are needed to achieve a desired performance?

Even with the answers to these questions, it remains to be seen if *real-world* data, in the form of user demonstrations, are well modeled by this abstraction. To isolate the performance of the LBO system in modeling user subgoal selection, we remove extraneous factors such as sensor noise and environment considerations. We are then left with the simplified problem of inferring the user model from sequences of subgoals. An essential capability for any LBO system is *predicting* the next subgoal in a sequence, which is necessary because, if an LBO system cannot accurately predict a subsequent subgoal, then there is little hope it can generate the sequence of subgoals necessary to automate an entire task. We analyze the performance of the CDHMM in modeling user subgoal selection on real-world data with the application of Predictive Robot Programming (PRP). From a high-level perspective, in PRP, users provide subgoals directly to the LBO system, which are measured precisely and have negligible sensor noise. Even then, the subgoals must be considered noise-corrupted due to the poor repeatability and low precision of humans. The system estimates a model that describes how users create subgoals, without consideration for how the environment effects performance. While users are performing a task, the PRP system predicts only the *next* subgoal, shown graphically in Figure 1.4. By analyzing the accuracy of these predictions, we characterize the ability of a CDHMM to model user subgoal selection.

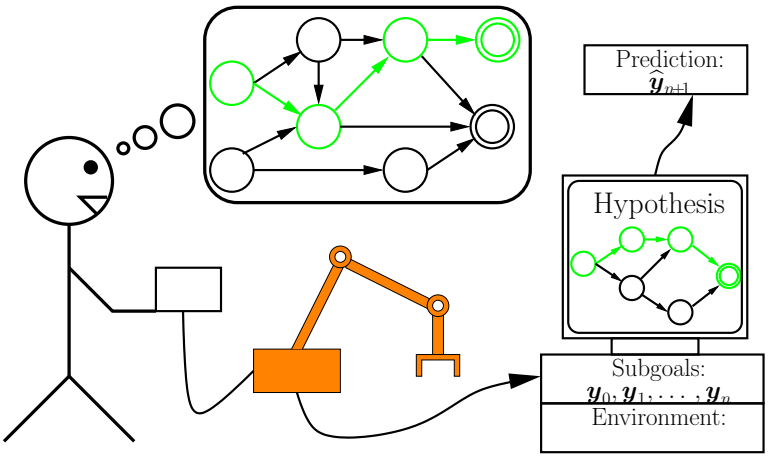


Figure 1.4: Isolating user-subgoal-selection modeling performance with Predictive Robot Programming.

1.3.2 Hypothesizing About User Actions

As mentioned previously, a subgoal is a vector in the state-space of the user. In many motor-skill tasks, the specific actions necessary to achieve a subgoal are equally as important as the subgoal itself. Consider a hypothetical LBO system learning to drive a car on a given route based on a user demonstration. The subgoals might be the locations of the intersections along the way. However, the specific actions needed to reach the destination are as important as the subgoals themselves. For example, driving in a straight line between the intersections would result in disaster for most automobiles if there are any curves in the road. An LBO system, such as this hypothetical driving example, or ours, must hypothesize about what actions the user would perform to achieve the subgoals. In other words, the LBO system must infer user *intent*. The hypothesis is then as follows:

Hypothesis 1.2. *We can form accurate hypotheses of what actions the user would perform in novel conditions based on previous demonstrations of a task.*

We create a mathematical abstraction of user intent, with a model of sequenced linear dynamical systems. We then explore provable statements that can be made about the performance of the model, which take the form of propositions and answer these questions:

- Can we optimally represent hypotheses of user actions to novel conditions?
- Under what conditions will the model succeed and fail?

1.3.3 Learning By Observation

Once again, it remains to be seen how well this mathematical abstraction models real-world data. We verify these ideas in laboratory experiments with our LBO system called "Dollop," which embodies the key features that complicate LBO systems, such as sensor noise and environmental considerations. In these experiments, a mobile robot learns motor-skill tasks by observing users with a laser. From these observations, the system automatically extracts the task subgoals. Dollop can incorporate multiple demonstrations of a task, in addition to demonstrations performed in different environment configurations. From these examples, Dollop forms a hypothesis about user intent, giving the system the ability to complete the task successfully under novel conditions. Consequently, Dollop serves as the basis for verifying the key concepts needed in an LBO system.

1.4 Contributions

The main contributions of this work are:

- A **CDHMM structure-estimation algorithm** (Dixon et al., 2004). This algorithm is designed to cope with the characteristic difficulties of LBO: scarce training data and real-time operation. The algorithm can operate in an online fashion, as observations become available, or as batch processing, on a complete data set. We provide a theoretical analysis showing the strengths and weaknesses of the algorithm.

- Design and analysis of a **Predictive Robot Programming** system (Dixon & Khosla, 2003). In addition to isolating variables to study LBO performance, the application of PRP is worthwhile in its own right to decrease manipulator-robot programming time. We show that the PRP system has a median prediction error less than 0.5% of the distance traveled during prediction on a set of data from complex, real-world robotic tasks. We also present laboratory experiments showing that the PRP system results in a significant reduction in programming time, with users completing simple robot-programming tasks over 30% faster when allowing the PRP system to compute predictions of future positions.

- Trajectory representation using **Sequenced Linear Dynamical Systems** (Dixon & Khosla, 2004b). We have derived a novel method to represent hypotheses of user actions by estimating a sequence of linear dynamical systems that describes a demonstration. We show optimality of the formulation and provide a proof of stability of the induced supervisory control law.

- Development of a computational approach to **Learning By Observation** (Dixon & Khosla, 2004a). We leverage the analyses from the previous contributions to develop an LBO system that learns motor-skill

tasks from user demonstrations. Multiple demonstrations can be incorporated to improve system performance, and demonstrations can be performed in different environment configurations. We present laboratory experiments showing that the LBO system accurately infers user intent.

1.5 Outline

This dissertation is outlined at follows:

- **Chapter 2: Background** provides a review of the relevant previous work in the related areas, such as representing motor commands, LBO, and HMMs.

- **Chapter 3: The Learning Algorithm** derives the CDHMM structure-estimation learning algorithm used throughout this work.

- **Chapter 4: Predictive Robot Programming** analyzes the ability of our system to predict waypoints in complex, real-world manipulator robot programs. We also present results showing a reduction in the time needed to program simple tasks in a laboratory setting.

- **Chapter 5: Hypothesizing About User Actions** describes the representation of hypotheses of user actions with Sequenced Linear Dynamical Systems.

- **Chapter 6: Learning By Observation** describes our motor-skill LBO system, Dollop, with experimental results in a laboratory setting.

- **Appendix A: Technical Details** to preserve continuity, we have put the proofs to most propositions, as well as lengthy derivations, in the appendix. The exception is when the proof is particularly instructive or extremely short.

Chapter 2

Background

This chapter describes the body of research related to the key ideas in this dissertation.

2.1 Introduction

As mentioned previously, we decompose Learning By Observation into two distinct components, modeling the response of the user to different conditions and user subgoal selection. In this chapter, we describe research related to each of these problems. Fundamental to modeling user response to different conditions is a method for encoding motor commands (Section 2.2). We use CDHMMs (Section 2.3) to model user subgoal selection during task demonstrations. In Section 2.4 we discuss theoretical results that provide constraints about the tractability of this approach. In Section 2.5 we place our work in the context of research related to LBO.

2.2 Encoding Motor Commands

This section describes methods by which various systems encode motor commands. Essentially, there are two broad choices regarding the strategy to represent a user demonstration. The optimal-control approach attempts to describe a user demonstration by a control law, which is a collection of coupled functions that may be stochastic in nature. The behavior-based approach describes a user demonstration by a set of predefined "primitives." By sequencing the behaviors appropriately, a system can represent user demonstrations in a symbolic manner.

2.2.1 Control Laws

The most straightforward method for encoding actions is a control law (Antsaklis & Michel, 1997), that is, a function that maps the current state of the system to a set of motor commands. The use of control laws to model biological systems goes back at least half a century (Ashby, 1952) and remains a popular approach to construct biologically plausible theories about brain behavior (Friston & Price, 2001).

Optimal Control

The field of optimal control (Stengel, 1986), where a control law is computed that minimizes some cost function, has been extremely successful for encoding motor commands. These techniques made possible automatic piloting of aircraft, cruise control in automobiles, and travel to the moon (Betts & Erb, 2003). Economists have made extensive use of optimal-control theory (Dorfman, 1969), and it has been applied to disease-therapy research (Stengel et al., 2002). A method for creating nonlinear controllers that approximately track a desired trajectory has been demonstrated for aerial vehicles (van Nieuwstadt & Murray, 1998). Ghahramani and Roweis (1999) use an iterative algorithm to approximate a demonstrated trajectory, and Ijspeert et al. (2001) describe an approach using nonlinear attractor dynamics for motor-skill learning problems. Kaiser et al. (1995) designed a neural-network system that creates a nonlinear force feedback controller based on error backpropagation (Mitchell, 1997).

The underlying mathematical model of the optimal-control formulation is the Markov Decision Process (MDP) (Boutilier et al., 1999). An MDP is a model where the current state of the system and a control law uniquely determine the cost function. Furthermore, the probability distribution of the next state of the system given the current state and the control law is conditionally independent of all other information, i.e., the Markov assumption. By making some state variables unobservable, an MDP becomes a Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998). Despite known hardness results for POMDPs (Lusena et al., 2001), these models have shown themselves extremely useful in practical applications (He & Shayman, 2000; Pineau et al., 2003).

Reinforcement Learning

While optimal-control techniques have been around since the 1950s, there has been more recent interest in applying these concepts to situations where there are unknown dynamics or extremely large state spaces. This subfield is known as Reinforcement Learning (RL) (Kaelbling et al., 1996; Sutton & Barto, 1998). The basic algorithms of RL, such as Q-learning (Watkins & Dayan, 1992) and TD(λ) (Dayan, 1992), are theoretically well founded and provide optimality results, but most approaches are heuristic and approximate in nature (Bertsekas & Tsitsiklis, 1996). This is, after all, because the goal of RL is solving intractable optimal-control problems. Despite known

divergence results when using a broad class of function approximators (Tsitsiklis & Van Roy, 1995), many researchers have successfully used neural networks in RL applications (Zhang & Dietterich, 1996). The best-known example of an RL application is the system that learned to play the game of backgammon, called TD-Gammon (Tesauro, 1995). By playing many games against itself, TD-Gammon learned to play at a level competitive with the best players in the world.

2.2.2 Behavior-Based Systems

Behaviors are variously known as skills (Kaiser et al., 1996), schemas (Arkin & Balch, 1997), primitives (Morrow & Khosla, 1995), basic operations (Friedrich et al., 1996), and atomic actions (Kuniyoshi et al., 1994). An often-implied definition of a behavior is a function that generates motor commands based on the current and previous states of the system (Matarić, 1992). For this reason, a behavior is a generalization of a control law. A typical behavior-based system contains multiple behaviors that execute in parallel with a scheduling algorithm switching between the different behaviors in response to sensory input; some systems allow multiple behaviors to issue motor commands simultaneously (Matarić, 1997). This scheduling algorithm is similar to the concept of sliding control (Slotine, 1984). As a general design principle, a behavior should be modular, permitting its reuse in different applications. Furthermore, behaviors are typically designed to achieve some goal. Common examples of these include "avoid stationary objects," "wall following," "pick up object," and "navigate to landmark" (Arkin, 1998). As such, complex *external* behavior emerges from the complex interplay of simple *internal* behaviors. Given these building blocks, the primary difficulty in creating behavior-based systems is designing the scheduling algorithm that sequences the behaviors to achieve a high-level goal. These scheduling algorithms can be quite complex, as Fagg et al. (1994) write: "However, despite some of the recently reported successes of reactive or behavior-based approaches, hiding behind most successes is a graduate student who spends many hours carefully designing, testing, and redesigning the set of control modules, until the desired behavior is achieved...." Consequently, there has been a substantial amount of research in trying to *learn* the scheduling algorithm (Maes & Brooks, 1990; Mahadevan & Connell, 1992) and many recent behavior-based systems use a learning procedure to tune the scheduling algorithm (Kaminka et al., 2002). There has also been recent work in using auction-based methods to design the scheduling algorithms, on both hand-crafted (Gerkey & Matarić, 2002; Zlot et al., 2002) and learned (Bererton et al., 2004) reward functions.

The scheduling algorithm must select from a discrete set of behaviors, and it is common to define a behavior in terms of symbolic pre-conditions and post-conditions. Nicolescu (2003) describes the behaviors `Localize`, `GetBox`, `Goto(Door)`, `OpenDoor`, `GoThroughDoor`, and `Goto(Home)`. For example, the behavior `OpenDoor` requires the pre-conditions `AtPlace(Door)=true` and `DoorOpened=false`. The behavior achieves the post-conditions `DoorOpened=true` and `HaveBox=false`. To achieve a high-level goal, e.g., carrying a box from a room through a door to a home position, a behavior-based system typically schedules the

behaviors in terms of predicate calculus (Russell & Norvig, 2002). While predicate-calculus problem solvers offer a clean avenue to achieve a goal, this approach ignores the uncertainty inherent in mobile robotics (Thrun, 2001).

Behavior-based robotics has had wide spread success in both research and practice. Unquestionably, the most resounding success of behavior-based robotics was the Mars rover, Sojourner (Gat et al., 1994). Behavior-based robots also dominate the robotic soccer competition, RoboCup (Lenser et al., 2002). Further examples of behavior-based systems are also given throughout the remainder of this chapter.

2.2.3 Our Approach

For the motor-skill learning tasks that we consider, we do not need the high-level representation that behavior-based systems offer. We have, therefore, formulated motor-command encoding as an optimal-control problem (Chapter 5). Essentially, we compute a sequence of control laws that minimize the error between what the LBO system does and what the user demonstrated. This gives our LBO system the ability to hypothesize what the user would do by examining the response of the control laws to novel conditions. The formulation also allows multiple demonstrations to be incorporated into the hypothesis to improve its generalization.

2.3 Hidden Markov Models

According to Rabiner and Juang (1986), a Hidden Markov Model (HMM) "is a doubly stochastic process with an underlying stochastic process that is *not* observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols." Informally, an HMM has a set of states and a random process that transitions between states at discrete time steps according to state-transition probabilities. Given the current state of the HMM, the probability of the next state is conditionally independent of all other information. That is, the current state uniquely determines the probability of subsequent states. At each discrete time step, an observation is generated according to a specific probability distribution depending only on the current state. To an observer of an HMM, the observations are visible, while the underlying state sequence is hidden. A typical HMM is defined to have a finite set of states and a finite set of possible observations (alphabet size). A common practice for converting continuous data into discrete data is known as Vector Quantization (VQ) (Rabiner & Juang, 1993). VQ and its adaptive counterpart, Learning Vector Quantization (Kohonen, 1988), attempt to put a continuous vector into a finite number of "bins." For many automation tasks, quantizing the observation vectors is simply not an option and observations will remain continuous vectors. In other words, there are an uncountable number of possible symbols that users can produce. Consequently, there is a class of HMMs that has a finite number of states and continuous-density observations (Juang et al., 1986). This type of HMM is called a Continuous-Density Hidden Markov Model (CDHMM), which forms the substrate of our LBO system. There

are also extensions of CDHMMs to include an uncountable number of states, i.e., the state variable is continuous (Thrun et al., 1999).

2.3.1 "The Three Basic Problems for HMMs"

Rabiner (1989) identifies what are typically called "The Three Basic Problems for HMMs."

Evaluation

The evaluation problem is, given an HMM and an observation sequence, how do we compute the probability that the observed sequence was produced by the model? The answer to this question is solved efficiently using an algorithm, similar to Dynamic Programming (DP), called the "forward procedure."

Decoding

The decoding problem is, given an HMM and an observation sequence, how do we determine the optimal sequence of HMM states that explains the observations? Not specifying the term "optimal" makes this question somewhat ambiguous. In most applications, the optimality criterion is taken as maximum likelihood and then the decoding problem can be computed efficiently using a DP algorithm called the "Viterbi algorithm."

Training

The training problem is, given an HMM topology and an observation sequence, how do we tune the parameters of the HMM to maximize the likelihood of the observation sequence? Taken at face value, this is a hopeless problem, since there are results suggesting that globally optimal training of HMMs is not possible in polynomial time (Abe & Warmuth, 1992). However, it is possible to arrive at a locally optimal solution using an iterative Expectation-Maximization approach (Bilmes, 1997). When applied to HMMs, this is called the Baum-Welch or forward-backward algorithm (Rabiner, 1989).

2.3.2 The Question Not Asked

Rabiner (1989) notes that "by far the most difficult" of the Three Basic Questions is the training problem. Conspicuously absent from those questions is the *much* more difficult problem of how to estimate the *structure* of an HMM from an observation sequence, in addition to finding the optimal parameters. Once again, taken at face

value, this is an inherently ill-posed problem, since an infinite number of HMMs can produce any particular observation sequence. In Section 2.4 we describe some research in estimating the structure of various graph-based models.

2.4 Structure Estimation

Most of the research in structure estimation for HMMs can be traced back to Gold (1967). In the seminal paper, Gold (1967) writes, "In finite identification, the learner is to stop the presentation of information at some finite time when it thinks it has received enough, and state the identity of the unknown object. This is not possible unless there is some finite time at which the information distinguishes the unknown object. That is, no other object satisfies the information."

2.4.1 Structure Learning in Deterministic Finite-State Automata

Initial research in structure estimation focused on learning with Finite-State Automata (FSAs); Ron (1995) gives a comprehensive survey. The most fundamental form of structure estimation, a deterministic FSA, was shown by Angluin (1978) to be NP-complete even when given both positive and negative observation sequences, i.e., examples that the target FSA could have and could not have generated. From this foundation, it follows that the more sophisticated structure-estimation problems are intractable as well. Kearns and Valiant (1994) showed that deterministic FSA structure estimation from an observation sequence is at least as hard as factoring, regardless of the choice of representation used in the learning algorithm. This implies that structure estimation in deterministic FSAs is as difficult as breaking many current cryptographic protocols. However, when the learning algorithm is provided access to an oracle that can provide answers to queries, then the deterministic FSA structure-estimation problem becomes tractable (Angluin, 1987). Rivest and Shapire (1994) derive a polynomial-time algorithm for exactly learning the structure of an FSA when the learner is allowed to experiment and given access to an oracle. When deterministic automata become stochastic, either with probabilistic transitions between states or noise-corrupted observations, the results become worse. Rudich (1985) gives an algorithm for estimating the structure of a Markov chain in the limit of infinite observations, but the computational complexity is intractable. Gillman and Sipser (1994) show that even when given access to an oracle, an ergodic HMM learning algorithm requires an exponential number of queries for exact learning.

2.4.2 Structure Learning in Hidden Markov Models

In the machine-learning field there is a substantial amount of research on structure estimation in HMMs. Due to the inherently ill-posed nature of the structure-estimation problem, some researchers have focused on special subclasses of HMMs to deliver better results. Ron et al. (1998) have derived a state-merging algorithm that constructs correct, in the Probably Approximately Correct sense, discrete-symbol Probabilistic Finite Automata (PFAs). The running time is polynomial in the number of observations and alphabet size, provided the states of the target PFA are *distinguishable* and *acyclic*. There is a similar state-merging algorithm that estimates general PFA topologies in the limit of infinite training data (Carrasco & Oncina, 1999). The theoretical results of these works rely on enumerable alphabet sizes, whereas LBO requires multivariate real-valued vectors to describe human actions. Several researchers have also pursued more heuristic approaches to HMM structure estimation. For example, Stolcke and Omohundro (1994a) incorporated a prior topology distribution favoring simple models and performed a best-first state-merging to induce the structure of an HMM from observations. Brand (1999) used an entropy-based prior to cause parameter extinction in ergodic HMMs. But such heuristics make it difficult to provide performance guarantees. Singh et al. (2002) used early-stopping techniques to determine the phonetic units for a speech recognizer. The process of determining the atomic units of speech has much in common with representing the "primitives" needed to complete a set of tasks. However, the large cross-validation sets found in the speech-recognition domain, required for early-stopping, are not available in LBO.

There has also been recent interest in estimating the structure of HMMs in the statistics and information-theory fields, where it is called order estimation (Ephraim & Merhav, 2002). Results from these fields deal with bounding the asymptotic likelihood of under- or over-estimating the number of states (Merhav et al., 1989) and there have been generalizations to handle continuous observations (Rydén, 1995). While theoretically appealing, these approaches rely on the existence of a globally optimal maximum-likelihood estimation procedure for HMMs, which is known to be intractable (Abe & Warmuth, 1992). There has been more recent work on restricting the classes of HMMs in order to derive viable results (Gassiat & Boucheron, 2003). However, these results are focused on asymptotic performance and not computational complexity. Even the tractable approaches specify termination only in a finite number of steps (Ephraim & Merhav, 2002) and, consequently, these guarantees are not appropriate for real-time use in an LBO system.

We have created a CDHMM learning algorithm motivated by the characteristic difficulties of LBO (Chapter 3). First, the algorithm must be able to operate in real time, so its computational complexity must be low and it cannot rely on asymptotic performance. Furthermore, the algorithm cannot expect large amounts of training data, such as the large corpora of data in the speech-recognition domain. Rather, our learning algorithm is designed to identify the relatively short-sequence similarities found in LBO. We have, therefore, focused our attention on a low-complexity algorithm that produces the "right" observation at the "right" time, rather than asymptotic sequence guarantees such as those described by Ephraim and Merhav (2002).

2.5 Learning By Observation

We use the term Learning By Observation (LBO) to describe the umbrella concept of a computer-based system learning from observations obtained from user demonstrations of a task. The goal of an LBO system is to synthesize the information in order to automate the task. LBO is variously known as Programming By Demonstration, Teaching From Example, Learning By Experience, or some permutation thereof. From our perspective, the most important distinction occurring in LBO systems is whether a particular system *imitates* or interprets *intent* from user demonstrations.

2.5.1 Imitation

Imitation is an enormously well-studied phenomenon in psychology and biology (Zentall & B.G. Galef, 1988). This work is concerned with computational aspects of imitation and the application to robotics. We, therefore, restrict our attention to what it means for a robotic system to "imitate" a user (Kawamura et al., 2000). Schaal (1999) describes imitation in terms of a control policy that maps the current state of the system to a set of motor commands. Many approaches have attempted to compute this mapping directly, using look-up tables or function-approximation techniques. In fact, this method forms the basis of industrial robotics (Craig, 1989), where users supply robots with the precise sequence of motor commands to perform and there is no learning involved. This requires that the workspace be extremely well calibrated and structured, since industrial robots cannot adapt their response to unspecified conditions.

The most straightforward implementation of LBO is one where the actions of the teacher are repeated directly (Gaussier et al., 1998). Asada and Asari (1988) created a system that learned to imitate the forces exerted by users during a demonstration. Hayes and Demiris (1994) developed an LBO system that allowed the transfer of knowledge between robots using direct imitation of the actions of the teacher-robot. Schaal (1997) extends this concept by incorporating RL techniques to refine the action sequence learned from users. Many researchers have developed LBO systems that attempt to reproduce a predefined sequence of features, such as contact points or higher-level concepts. Chen and Zelinsky (2003) use a symbolic configuration-space description to represent a simple assembly task. Suboptimal human demonstrations were ameliorated by "filtering" the observations and by incorporating extensive domain knowledge with heuristic methods. Friedrich et al. (1996) developed a system that decomposed a task, demonstrated by users, into a sequence of predefined primitives. The commands were executed in sequence from a pre- and post-condition theorem-proving model (Fikes & Nilsson, 1971). This representation restricts its use to well-structured, static environments that do not require sensory feedback. Kaiser and Dillman (1996) further extended this work by learning the primitives involved in a user demonstration. Ijspeert et al. (2002) have developed a computational method for movement imitation that can incorporate multiple demonstrations of a task to improve performance. This system can also modulate the learned trajectory based on objects in the

environment. Kang and Ikeuchi (1995) decompose camera images into a set of primitives to imitate a human demonstration of a grasping task. Yang et al. (1994) have developed a system that learns to perform simple telerobotic tasks from a user demonstration. Hugues and Drogoul (2002) learn to imitate the slalom skills of users from demonstrations by incorporating extensive environmental information.

2.5.2 Interpretation

Some LBO systems go further than imitation strategies and attempt to interpret the *intent* of users from demonstrations. Informally, user intent is the set of invariants occurring in the demonstrations (Billard et al., 2003; Alissandrakis et al., 2002). The concept of inferring user intent is relatively new and few LBO systems have incorporated this idea into the system during the design process. Iba et al. (2003) have developed a system that attempts to simplify the robot-programming process by reducing the number of instructions that must be provided by users by inferring their intent. Intille and Bobick (1999) created a system to infer the goals of American football players from visual-scene analysis. Skubic and Volz (2000) construct a Finite-State Machine (FSM) based on the contact information of objects in the environment from user demonstrations. Multiple demonstrations can be incorporated into the FSM to derive a better generalization of the correct sequence of contacts. Similarly, Nicolescu and Matarić (2001) created a system that decomposes a demonstration into a symbolic set of robotic behaviors. Using dynamic programming and human feedback, an FSM behavior network is constructed from multiple examples of a task. Intentions are inferred by allowing users to give the system feedback from a predefined vocabulary.

In unstructured environments, LBO systems will be presented with unfamiliar situations that may cause their performance to decline, unless properly designed. The obvious problem with imitation-based approaches is that users cannot possibly demonstrate the correct action for every possible situation that the LBO system may encounter. However, most LBO systems only incorporate user models in *ad hoc* situations to improve upon previously poor system performance. Consider the case of two imitation-based systems (Pomerleau, 1991; Bentivegna & Atkeson, 1999) that were forced to use a more general approach due to the uncertainties of the real world (Pomerleau, 1996; Bentivegna et al., 2003). Pomerleau (1991) developed an LBO system that learned to imitate the vehicle-steering skill of human drivers using a neural network that mapped camera images to steering commands. However, he noted that, “since the person steers the vehicle down the center of the road, the network will never be presented with situations where it must recover from misalignment errors. When driving for itself, the network may occasionally stray from the road center, so it must be prepared to recover from steering the vehicle back to the middle of the road (Pomerleau, 1996).” To achieve acceptable performance, Pomerleau (1996) modified camera images to emulate driver recovery using extensive knowledge of human-driving ability, essentially inferring the intent of users. These improvements allowed the system to drive successfully in unstructured environments. Bentivegna and Atkeson (1999) designed an LBO system that learns to roll a ball through a maze while avoiding holes from user demonstrations. This imitation-based approach encountered problems similar to

the vehicle-driving system; that is, it is not possible for users to demonstrate the correct action in every location of the maze, and during normal operation users only roll the ball through a small subset of possible positions. The initial solution imitated user actions from a nearest-neighbor rule. That is, for a given ball location and velocity, lookup the action performed by users in the closest conditions, in the Euclidean sense. In the hole-filled maze, however, similar Euclidean positions may require vastly different actions, e.g., to avoid a hole. The proposed solution was to allow the LBO system to generate a large number of self-simulated demonstrations, without user intervention, which greatly improved the performance of the LBO system (Bentivegna et al., 2003). This system is still imitative but uses simulation to give itself an essentially limitless number of examples to imitate.

Our approach to LBO incorporates a CDHMM to model user subgoal selection from demonstrations of a task (Chapter 3). Our LBO system infers user intent by forming hypotheses of user actions to different conditions (Chapter 5) and updating the subgoals of the LBO system according to "what the user would have done" (Chapter 6).

2.5.3 HMM-based LBO Systems

Several researchers have also used HMMs to describe human actions. Typically, these systems create an individual HMM for each action that users may perform during a given task. The topologies of these constituent HMMs are determined *a priori* according to some predefined structure, such as left-right models (Hannaford & Lee, 1991) and the HMMs are typically optimized by the Baum-Welch algorithm using a labeled demonstration. To encode higher-level knowledge, these models are sometimes connected together in a task-specific manner, creating a sort of "grammar" that constrains the set of possible actions (Hovland et al., 1996). This grammatical concept has recently been extended to recognize unknown actions (Iba et al., 2003). HMM-based LBO systems have also been used in the telerobotic manipulation domain for analysis of force/torque actions (Hannaford & Lee, 1991), flipping eggs (Pook & Ballard, 1992), consistency analysis of human actions (Tso & Liu, 1997), learning new skills (Yang et al., 1997), and real-time operator assistance (Hundtofte et al., 2002; Li & Okamura, 2003). In addition to their applications in speech recognition (Rabiner, 1989), HMMs have also been used extensively in recognizing handwriting of various languages (Bahlman & Burkhardt, 2001; Solis et al., 2002), and finding genes in DNA sequences (Krogh et al., 1994).

As mentioned previously, we employ a CDHMM to model user subgoal selection. Other HMM-based LBO systems have relied on knowing the structure of the target task in advance. However, our approach differs from previous work in that we do not know the set of tasks that users may perform *a priori*. To cope with this issue, we have developed a learning algorithm that estimates the *structure*, or topology, of a CDHMM based on user observations. After estimating the structure of the model, any of the well-known fixed-topology algorithms, such as the Baum-Welch algorithm, can be applied to optimize the model parameters.

2.6 Some Fundamental Definitions from Linear Algebra

Several propositions in this work rely on fundamental results from linear algebra (Strang, 1993; Golub & Van Loan, 1996), and we give the definitions here. A positive-definite (PD) matrix, $\boldsymbol{C}$, is defined as any matrix that satisfies the inequality $\boldsymbol{x}^\mathsf{T}\boldsymbol{C}\boldsymbol{x} > 0$, for any $\boldsymbol{x} \neq \boldsymbol{0}$. Similarly, a positive-semidefinite (PSD) matrix, $\boldsymbol{C}$, is defined as any matrix that satisfies the inequality $\boldsymbol{x}^\mathsf{T}\boldsymbol{C}\boldsymbol{x} \geq 0$, for any $\boldsymbol{x} \neq \boldsymbol{0}$. The trace of a square matrix, $\boldsymbol{C}$, is the sum of its diagonal elements and is written as

$$\mathrm{tr}[\boldsymbol{C}] \triangleq \sum_i \boldsymbol{C}_{i,i}.$$

The number of linearly independent rows (or columns) of a matrix $\boldsymbol{C}$ is called the rank. An orthonormal matrix has orthogonal unit basis vectors as its columns and, consequently, its transpose is its inverse. The Singular-Value Decomposition (SVD) of a real matrix $\boldsymbol{C}$ is defined as

$$\boldsymbol{C} \equiv \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\mathsf{T},$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal and $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_r^2, 0, \ldots, 0)$, where r is the rank of $\boldsymbol{C}$. The singular values are sorted in descending order as $\sigma_1^2 \geq \cdots \geq \sigma_r^2 > 0$. It can be shown that the SVD always exists for any matrix (Golub & Van Loan, 1996). Let $\boldsymbol{C}^\mathsf{R}$ be the right pseudo-inverse of $\boldsymbol{C}$, which is defined in terms of the SVD,

$$\boldsymbol{C}^\mathsf{R} \triangleq \boldsymbol{V}\mathrm{diag}(1/\sigma_1^2, \ldots, 1/\sigma_r^2, 0, \ldots, 0)\boldsymbol{U}^\mathsf{T}.$$

If the matrix $\boldsymbol{C}$ has full row rank, then $\boldsymbol{C}^\mathsf{R} \equiv \boldsymbol{C}^\mathsf{T}\left(\boldsymbol{C}\boldsymbol{C}^\mathsf{T}\right)^{-1}$. If the matrix $\boldsymbol{C}$ has full row rank and is square (hence it has full column rank as well), then the right pseudo-inverse becomes the inverse $\boldsymbol{C}^\mathsf{R} \equiv \boldsymbol{C}^{-1}$.

For vectors, the Euclidean distance, or L_2 norm, is defined as $\|\boldsymbol{x}\|_2^2 \triangleq \boldsymbol{x}^\mathsf{T}\boldsymbol{x}$. The Frobenius norm of a matrix, $\boldsymbol{C} = [\boldsymbol{c}_1 \cdots \boldsymbol{c}_N]$, is defined as

$$\begin{aligned}\|\boldsymbol{C}\|_F^2 &\triangleq \sum_i \sum_j \boldsymbol{C}_{i,j}^2 \\ &= \sum_j \|\boldsymbol{c}_j\|_2^2.\end{aligned}$$

For the Frobenius norm, it can be shown that, if $\boldsymbol{V}$ is an orthonormal matrix, then $\|\boldsymbol{C}\|_F = \|\boldsymbol{C}\boldsymbol{V}\|_F$. The Mahalanobis distance (Duda et al., 2001), sometimes called the weighted-Euclidean norm, is defined as

$$\|\boldsymbol{x} - \boldsymbol{y}\|_{\boldsymbol{C}}^2 \triangleq (\boldsymbol{x} - \boldsymbol{y})^\mathsf{T}\boldsymbol{C}(\boldsymbol{x} - \boldsymbol{y}),$$

where $\boldsymbol{C}$ is a symmetric PD matrix, sometimes called a precision matrix. It is easy to show that all the norms defined above are metrics, are positive definite and symmetric, and obey the triangle inequality.

2.7 Summary

We have formulated the encoding of motor commands using optimal control, in lieu of a behavior-based approach, since motor-skill learning problems do not need the high-level representation offered by behavior-based systems. Previous theoretical research into HMM structure estimation indicates that any approach will either rely on infinite data, severely restrict the class of possible models, or be heuristic in nature. To build a realizable LBO system, we cannot rely on infinite (or very large amounts of) training data, and restricting the class of models implicitly restricts the set of tasks that can be automated. As a result, our structure-estimation approach will be heuristic in nature, using a similarity-based state-merging algorithm. Finally, to be able to cope with the unpredictable operating conditions that the LBO system will encounter, the system must interpret user *intent* instead of merely imitating user actions.

Chapter 3

The Learning Algorithm

We derive a new learning algorithm that estimates the structure of Continuous-Density Hidden Markov Models (CDHMMs) based on observation sequences from a target CDHMM. The algorithm uses a best-first state-merging scheme to produce an estimated CDHMM whose underlying graph is irreducible, having a locally minimal number of nodes. The worst-case running time is quadratic in the number of observations. We then derive a bound showing that the estimate of the target CDHMM improves as additional observations are incorporated. We also show how the estimated CDHMM can be used to predict future observations, and we give a causal statistic that indicates how accurate a prediction is likely to be.

3.1 Introduction

Previous theoretical research in estimating the structure of HMMs from observation sequences (Section 2.4) indicates that the problem is inherently intractable; any approach must rely on infinite data, severely restrict the class of possible HMMs, or be heuristic in nature. After reviewing the literature in Section 2.4, we did not find a structure-estimation algorithm that was appropriate for a motor-skill LBO application; consequently, we have derived a CDHMM structure-estimation learning algorithm motivated by the characteristic difficulties of LBO. We begin by describing how our system models the user (Section 3.2) and motivate the learning algorithm from a high-level perspective (Section 3.3). The remainder of the chapter then formalizes the learning algorithm, beginning by precisely defining the terms and structures needed (Section 3.4). We define node similarity and show that the best-first merging algorithm only produces graphs with a locally minimal number of nodes (Section 3.5). We then show that the worst-case running time of the learning algorithm is quadratic in the number of observations (Section 3.6). The topological structure embodied in the graph is converted to a CDHMM using a simple one-shot

Figure 3.1: Evolution of the user model from a deterministic to a doubly stochastic system.

estimation (Section 3.7). Next we derive a bound showing that, as more tasks are incorporated into the CDHMM, the probability of generating the "correct" observation increases asymptotically (Section 3.8). We also compare the behavior of our learning algorithm to maximum *a posteriori* structure estimation (Section 3.9). We then show how the CDHMM computes predictions of future observations (Section 3.10) and describe a parameter that the PRP system uses to indicate its prediction confidence (Section 3.11).

3.2 Modeling the User

From the perspective of users, they select the task from their repertoire and begin performing it. Various configurations of the workspace may cause users to perform the same task in different ways. For example, consider the welding of a door frame. The goal of the task, and the intention of users, is somewhat independent of the order of specific actions taken to complete the task. Users may be able to achieve the goal by welding the corners in any order. In this sense, if the workspace is modified, the intention remains the same; the goal of the task is irrespective of the location, orientation, and size of the door frame in the workspace. With these assumptions, a reasonable user model is a Finite-State Machine (FSM). This type of model outputs a deterministic symbol at each discrete time step based on the current state. All branching is deterministic, based on inputs to the FSM. However, for a particular task there may be many possible ways for users to achieve the desired goal, and it is difficult to instrument fully any realistic working environment. It is unreasonable to require users to explain the exact reasons for choosing a particular action during each step of a complex task. Furthermore, the behavior of users will be modified by, e.g., obstacles in the workspace. Consequently, there will always be hidden, or latent, causes for user behavior. This means that transitions between states in the user model appear stochastic in nature. With these assumptions, the FSM user model becomes a Markov chain. Due to the poor repeatability and low precision of humans, observations can be considered noise-corrupted. In laboratory experiments, we have found the one-sigma Cartesian error to be approximately 2 centimeters for users moving a robot to a point in space with no external references. In most automation tasks, this error is too large to ignore. Therefore, the model of a user repertoire is a type of doubly stochastic random process, or Hidden Markov Model (HMM). The evolution of the user model is shown in Figure 3.1.

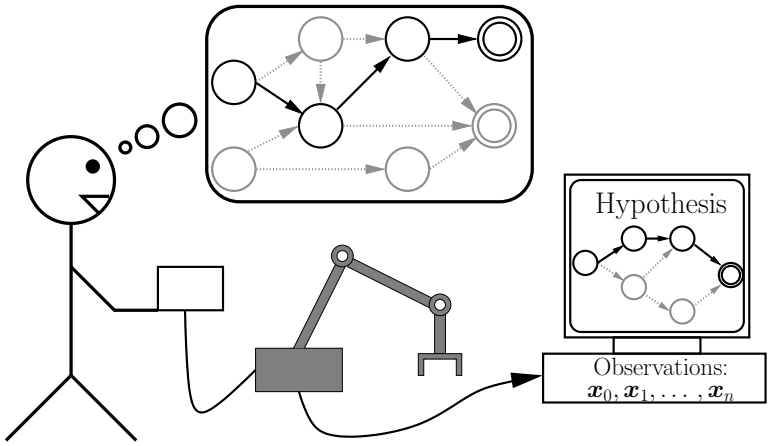


Figure 3.2: Conceptual modeling of users.

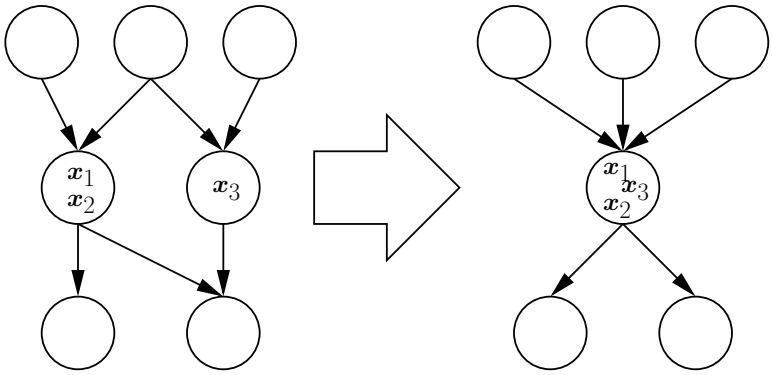
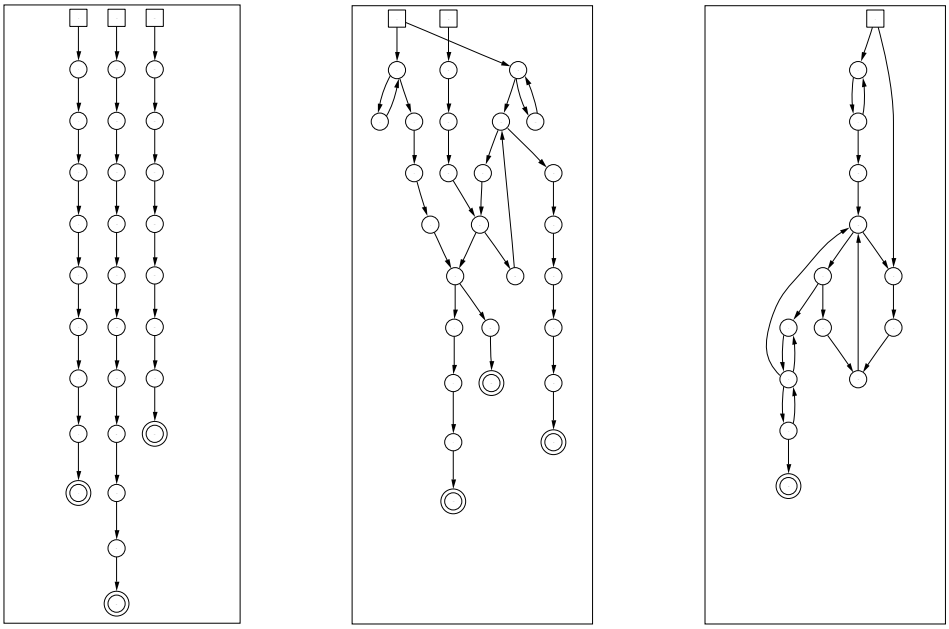


Figure 3.3: Hypothetical merging of two nodes in a graph.

3.3 Learning Algorithm Overview

We have created a learning algorithm that is designed to cope with the characteristic difficulties of LBO: sparse training data, real-time operation, and a wide range of target tasks. We assume that the system has no *a priori* knowledge of what tasks users may demonstrate. Therefore, we must induce the structure of the CDHMM, as well as parameters for the model, from observations alone (Figure 3.2). We consider users as generating each task according to a random walk through an unknown *target* CDHMM. The learning algorithm begins by assigning one observation to each node in a graph and encoding temporal information by edge connectivity, as in Figure 3.4(a). The algorithm then searches for the most *similar* nodes in the graph and, if these nodes are sufficiently similar, then the nodes are *merged*. That is, a new node is created containing the observations from the old nodes, and the edges of the old nodes are reconnected to the new node, as in Figure 3.3. This merging process repeats until



(a) (b) (c)

Figure 3.4: The maximal-node graph is constructed from three demonstrations of a task in Figure 3.4(a). Results of state merging using strict and loose similarity are shown in Figure 3.4(b) and Figure 3.4(c), respectively.

no similar nodes remain, as in Figure 3.4. At this point, the graph contains the *estimated* topology of the target CDHMM, i.e., the node connectivity, as well as the edge counts and the observations assigned to each node. The learning algorithm produces *general* directed graphs that may be cyclic, self-looping, or acyclic. In principle, any of the well-known fixed-topology HMM estimation procedures, such as the Baum-Welch algorithm (Rabiner, 1989), could be used to optimize the model parameters from the structure embodied by the graph. However, we use a simple one-shot procedure to determine the parameters of the CDHMM. For example, the edge counts can be quickly converted to transition probabilities by simple division. The observation-generation probability density functions can be estimated by fitting a parametric model to the observations assigned to each node, e.g., the mean and covariance of a Gaussian. We use one-shot estimation, in lieu of an iterative Expectation-Maximization (EM) approach (Dempster et al., 1977), for computational expediency and because we typically lack the data required for complete EM re-estimation procedures.

3.4 Definitions

To estimate the structure of the CDHMM, we use a type of directed multigraph called an *observation graph* and given by the 7-tuple $G_{\boldsymbol{\mathcal{X}}} = (V, V^0, E, \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{V}}, f, g)$ where

- V is the set of nodes;
- V^0 is the set of initial nodes;
- E is the set of directed edges;
- $\boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^d$ is the set of observations of dimension d;
- $\boldsymbol{\mathcal{V}} : V \to \mathcal{M}(\boldsymbol{\mathcal{X}})$ is the multiset of observations assigned to each node;
- $f : E \to \mathbb{Z}_{\geq 0}$ is the edge-count function;
- $g : V^0 \to \mathbb{Z}_{\geq 0}$ is the initial-count function.

The depth of an observation graph is determined by a breadth-first search from the set of initial nodes, V^0. Since the graph may be cyclic, a node may exist at multiple depths. A CDHMM is given by the quintuple $\boldsymbol{\lambda} = (\mathcal{Q}, \boldsymbol{\mathcal{X}}, a, b, \pi)$ where

- $\mathcal{Q}$ is a finite set of states;
- $\boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^d$ is the observation set of dimension d;
- $a : \mathcal{Q} \times \mathcal{Q} \to [0, 1]$ is the stationary state-transition probability mass function (pmf);
- $b : \boldsymbol{\mathcal{X}} \times \mathcal{Q} \to [0, \infty)$ is the stationary observation pdf;
- $\pi : \mathcal{Q} \to [0, 1]$ is the stationary initial-state pmf;

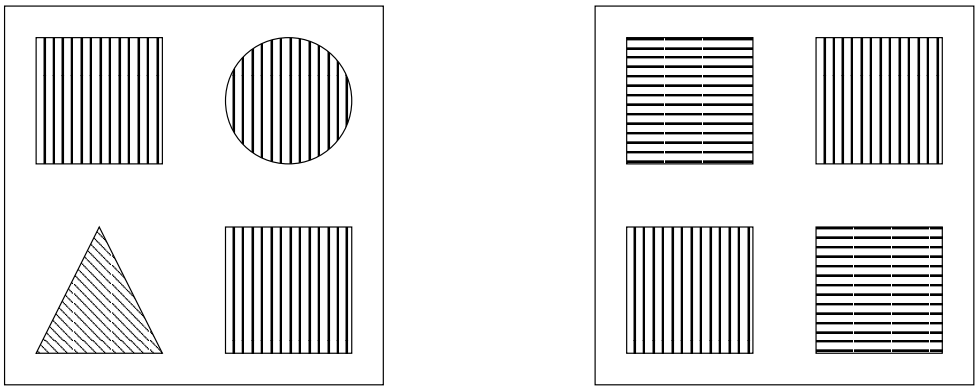
In this work, a *task* is defined as a finite sequence of observations.

3.5 Learning Algorithm Derivation

Central to the learning algorithm is the definition of similarity. We extend the use of the Mahalanobis distance to compute the similarity between observations assigned to different nodes in the graph. Our measure (Morgan, 2000) of similarity is defined as

$$\begin{aligned} \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \boldsymbol{y}) &\triangleq \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \|\boldsymbol{x} - \boldsymbol{y}\|_{\boldsymbol{C}}^2 \\ &\dot{=} \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} (\boldsymbol{x} - \boldsymbol{y})^{\mathsf{T}} \boldsymbol{C} (\boldsymbol{x} - \boldsymbol{y}), \end{aligned} \tag{3.1}$$

where the symmetric PD precision matrix $\boldsymbol{C}$ provides a notion of the *a priori* expectation of node variance. We also define $\mu_{\boldsymbol{C}}(\emptyset, \boldsymbol{y}) = 0$.



(a) (b)

Figure 3.5: The concept of similarity is a continuum.

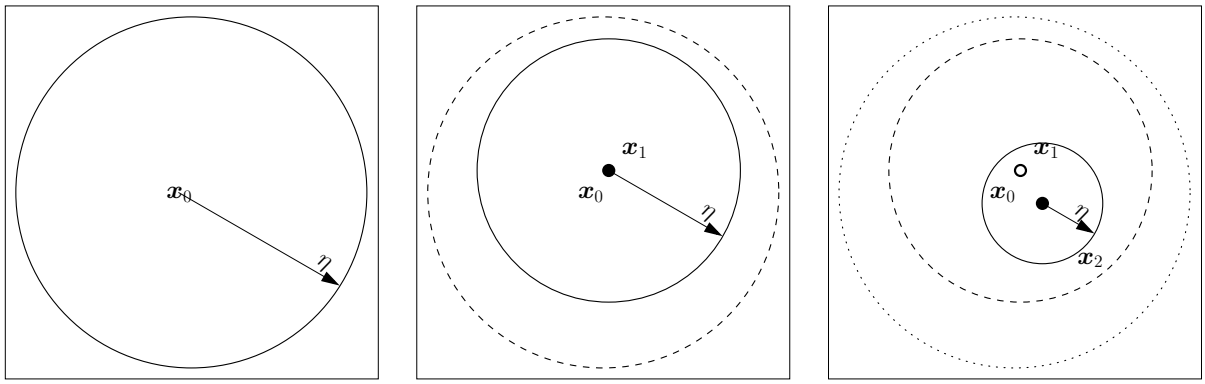
Lemma 3.1. *$\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \boldsymbol{y})$, defined in Equation 3.1, is a measure on the multiset $\mathcal{M}(\boldsymbol{\mathcal{X}})$ and is minimized when $\boldsymbol{y}$ is the sample mean of $\boldsymbol{\mathcal{V}}_{v_i}$.*

The proof is given in Section A.1.1.

Consider the objects shown in Figure 3.5, there are several possible groupings of similar objects. Furthermore, there are "strict" and "loose" definitions of similarity, as well as values in between; the notion of similarity is a continuum. It is natural to apply a stricter definition when objects are not as distinct. This concept is related to the "Universality of the Principles of Categorization" from cognitive psychology. Rosch et al. (1976) write, "On the most general level, categories form so as to be maximally differentiable from each other. This is accomplished by categories which have maximum cue validity – i.e., categories that have the most attributes common to members of the category and the least attributes shared with members of other categories." With this in mind, there seem to be two degrees of freedom in determining similarity: the expected variation of the objects and an absolute "threshold." If there is not much variation in the objects, such as in Figure 3.5(b), then the discrimination between objects must increase to identify similarity, and the converse also applies. In our learning algorithm, two nodes are considered similar if

$$\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j} \rangle) \quad \leq \quad \epsilon,$$

where $\epsilon \geq 0$ provides a continuous definition of similarity. Small values of ϵ imply a strict definition of similarity, while large values imply a loose definition. Intuitively, if a node v_i can be found with the capacity to add all the observations from node v_j, then the two nodes will be merged. This process repeats until no similar nodes exist.



(a) (b) (c)

Figure 3.6: Illustration of adding the two-dimensional vectors $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, and $\boldsymbol{x}_2$ to node v_i. The solid circle represents the region within which a future vector must lie to ensure that $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle) \leq \epsilon$. Consequently, the circle has radius $\eta = \epsilon - \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle)$. The dashed circle is the image of the previous region. The centroid of the circles shifts according to the sample mean, $\langle \boldsymbol{\mathcal{V}}_{v_i} \rangle$, of the observations already assigned to the node.

The nodes in the maximal graph in Figure 3.4(a) are merged using a strict and a loose definition of similarity, in Figure 3.4(b) and Figure 3.4(c), respectively. The resulting graphs are different, since the definition of similarity was different. In Section 4.3.2, we show how modifying the definition of similarity effects the prediction performance of the PRP system. With respect to computing the similarity between observation sequences, Equation 3.1 is *memoryless* in that the measure does not consider the similarity of ancestor or descendant observations. Consequently, it is possible for two sequences to be similar for a single observation while being dissimilar for all others. While recursive similarity may be appropriate in some domains (Ron et al., 1998), memoryless similarity seems more appropriate for identifying similarities common in LBO (cf. Figure 4.1). In a sense, the parameter ϵ behaves like an initial node "capacity," or a region within which future observations must lie. Let $\boldsymbol{x}_0$ be the first observation assigned to node v_i. Any future observations assigned to node v_i must lie inside a hyperellipse of radius ϵ, whose axes are defined by the eigendecomposition of $\boldsymbol{C}$, and centered about $\boldsymbol{x}_0$. Due to the non-negative nature of Equation 3.1, as more observations are added to node v_i, the capacity for the node to accept more observations decreases, shown graphically in Figure 3.6. In the following lemma we show that the acceptable region necessarily becomes smaller as observations are assigned to a node and that the subsequent regions are completely contained within the initial hyperellipse. This behavior is central in deriving an algorithm that produces irreducible observation graphs.

Lemma 3.2. *For any multiset $\mathcal{B}$ and for all submultisets $\mathcal{A} \subseteq \mathcal{B}$ and for all supermultisets $\mathcal{C} \supseteq \mathcal{B}$,*

$$\mu_{\boldsymbol{C}}(\mathcal{A}, \langle \mathcal{A} \rangle) \leq \mu_{\boldsymbol{C}}(\mathcal{B}, \langle \mathcal{B} \rangle) \leq \mu_{\boldsymbol{C}}(\mathcal{C}, \langle \mathcal{C} \rangle).$$

The proof is given in Section A.1.2.

Corollary 3.2.1. *For any $\tau \geq 0$ and multisets $\widetilde{\mathcal{A}}$ and $\widetilde{\mathcal{B}}$ such that $\tau < \mu_{\boldsymbol{C}}(\widetilde{\mathcal{A}} \cup \widetilde{\mathcal{B}}, \langle \widetilde{\mathcal{A}} \cup \widetilde{\mathcal{B}} \rangle)$ implies that for all supermultisets $\mathcal{A} \supseteq \widetilde{\mathcal{A}}$ and $\mathcal{B} \supseteq \widetilde{\mathcal{B}}$, $\tau < \mu_{\boldsymbol{C}}(\mathcal{A} \cup \mathcal{B}, \langle \mathcal{A} \cup \mathcal{B} \rangle)$.*

Proof. By direct extension of Lemma 3.2. □

For a state-merging approach such as ours, we want to define what it means for a graph to be irreducible, or *compact*. Intuitively, a compact graph is one where all similar nodes are merged but all dissimilar nodes are left unmerged. The following definition states this in precise terms.

Definition 3.1. *For some $\epsilon \geq 0$, an observation graph is* ϵ-compact *with respect to $\mu_{\boldsymbol{C}} : \mathcal{M}(\boldsymbol{\mathcal{X}}) \to [0, \infty)$, if for all nodes $v_i \neq v_j$ at the same depth*

- $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle) \leq \epsilon$,
- $\epsilon < \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j} \rangle)$.

While the definition of ϵ-compact implies a sort of irreducible graph, it does not necessarily imply a *globally* minimal number of nodes. It is entirely possible that merging nodes in a different order would result in fewer nodes. From this perspective, ϵ-compactness implies a type of *locally* minimal number of nodes. If we are striving for ϵ-compact graphs, then, by Corollary 3.2.1, we can merge node v_i and node v_j such that

$$\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j} \rangle) \quad \leq \quad \epsilon$$

and be assured that the observation graph will remain ϵ-compact. Indeed, this same test can be applied to an individual observation to implement an online scheme. This insight leads to Algorithm `Learn-Structure` in Figure 3.7.

Theorem 3.3. `Learn-Structure` *produces only ϵ-compact observation graphs.*

Proof. The algorithm satisfies the first predicate of Definition 3.1 ($\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle) \leq \epsilon$), since it will not add an observation, $\boldsymbol{x}_n$, to a node, v_i, unless $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \{\boldsymbol{x}_n\}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \{\boldsymbol{x}_n\} \rangle) \leq \epsilon$. Consider the second predicate of Definition 3.1 ($\epsilon < \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j} \rangle)$). Take any two nodes and, without loss of generality, assume

Algorithm `Learn-Structure`
$\boldsymbol{X} = \{\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots, \boldsymbol{X}^M\}$ is the multiset of all tasks.
$\epsilon \geq 0$ is the similarity threshold.

```
1: V := ∅, E := ∅
2: G_X := (V, V^0, E, X, V, f, g)
3: for all X^i ∈ {X^0, X^1, ..., X^M}
4:    G_X := assimilate-task (G_X, X^i, ε) (Figure 3.8)
5: end for all
```

Figure 3.7: The structure-estimation learning algorithm.

node v_i was created before node v_j. There must exist an observation $\boldsymbol{x}_{v_j} \in \boldsymbol{\mathcal{V}}_{v_j}$ and a submultiset $\widetilde{\boldsymbol{\mathcal{V}}_{v_i}} \subseteq \boldsymbol{\mathcal{V}}_{v_i}$ such that

$$\epsilon \;<\; \mu_{\boldsymbol{C}}(\widetilde{\boldsymbol{\mathcal{V}}_{v_i}} \cup \{\boldsymbol{x}_{v_j}\}, \langle \widetilde{\boldsymbol{\mathcal{V}}_{v_i}} \cup \{\boldsymbol{x}_{v_j}\}\rangle).$$

Otherwise, node v_j would never have been created. Loosely speaking, the observation $\boldsymbol{x}_{v_j}$ could not "fit" into any existing node (`assimilate-task` Line 7). From Corollary 3.2.1,

$$\begin{aligned} \epsilon \;&<\; \mu_{\boldsymbol{C}}(\widetilde{\boldsymbol{\mathcal{V}}_{v_i}} \cup \{\boldsymbol{x}_{v_j}\}, \langle \widetilde{\boldsymbol{\mathcal{V}}_{v_i}} \cup \{\boldsymbol{x}_{v_j}\}\rangle) \\ &\Rightarrow \\ \epsilon \;&<\; \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}\rangle) \end{aligned}$$

and therefore the algorithm produces only ϵ-compact observation graphs. □

3.6 Learning Algorithm Running Time

Lemma 3.4. *The sufficient statistics of $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i}\rangle)$ are*

$$\xi_i = |\boldsymbol{\mathcal{V}}_{v_i}|; \qquad \kappa_i = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x}; \qquad \boldsymbol{\sigma}_i = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x}.$$

The update rule for computing the union of two multisets ($\boldsymbol{\mathcal{V}}_{v_k} = \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}$) is

$$\xi_k = \xi_i + \xi_j; \qquad \kappa_k = \kappa_i + \kappa_j; \qquad \boldsymbol{\sigma}_k = \boldsymbol{\sigma}_i + \boldsymbol{\sigma}_j.$$

Function `assimilate-task`
$G_{\boldsymbol{\mathcal{X}}} = (V, V^0, E, \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{V}}, f, g)$ is the observation graph.
$\boldsymbol{X}^i = \{\boldsymbol{x}_0^i, \boldsymbol{x}_1^i, \ldots, \boldsymbol{x}_{N_i}^i\}$ is the task to assimilate.
$\epsilon \geq 0$ is the similarity threshold.

```
 1: for all x_n ∈ X^i = {x_0^i, x_1^i, ..., x_{N_i}^i}
 2:   ε_min := min_{v_i ∈ V} μ_C(V_{v_i} ∪ {x_n}, ⟨V_{v_i} ∪ {x_n}⟩)
 3:   if ε_min ≤ ε then
 4:     v_new := arg min_{v_i ∈ V} μ_C(V_{v_i} ∪ {x_n}, ⟨V_{v_i} ∪ {x_n}⟩)
 5:     V_{v_new} := V_{v_new} ∪ {x_n}
 6:   end if
 7:   else if ε_min > ε then
 8:     create empty node v_new
 9:     V := V ∪ {v_new}
10:     V_{v_new} := {x_n}
11:     g_{v_new} := 0
12:     if n > 0 then
13:       E := E ⋃ {e_{prev→new}}
14:       f_{e_{prev→new}} := 0
15:     end if
16:   end else if
17:   if n > 0 then
18:     f_{e_{prev→new}} := f_{e_{prev→new}} + 1
19:   end if
20:   else if n = 0 then
21:     V^0 := V^0 ∪ {v_new}
22:     g_{v_new} := g_{v_new} + 1
23:   end else if
24:   v_prev := v_new
25: end for all
26: return (V, V^0, E, X, V, f, g)
```

Figure 3.8: Assimilating tasks into an observation graph.

Using sufficient statistics, Equation 3.1 can be computed as

$$\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle) \quad \equiv \quad \kappa_i - \frac{1}{\xi_i} \boldsymbol{\sigma}_i{}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{\sigma}_i.$$

The proof is given in Section A.1.3. The significance of Lemma 3.4 is that the function $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j} \rangle)$ can be computed in constant time, irrespective of the number of observations assigned to either node. This is useful in providing a bound on the computational complexity of the algorithm.

Corollary 3.4.1. *The computational complexity of computing $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j} \rangle)$ is $O(d^2)$, where d is the dimension of an observation.*

The proof is given in Section A.1.4.

Though not required by the algorithm, we simplify the running-time analysis by assuming that each task is of length N.

Theorem 3.5. *The worst-case computational complexity of* `Learn-Structure` *to assimilate M tasks of length N is $O(M^2N^2d^2)$, where d is the dimension of an observation.*

Proof. We make two reasonable assumptions, namely:

- Appending a value to a multiset is constant-time ($\mathcal{A} := \mathcal{A} \cup \{\boldsymbol{x}\}$).
- Accessing any node is constant-time ($v_i \in V$).

With these assumptions, there is one line of non-constant cost inside the locus of `assimilate-task` , finding the minimum similarity measure (Line `2`). This line forms an arithmetic series on the number of nodes in the observation graph. From Corollary 3.4.1, the update rule for the union of two multisets is independent of their cardinalities and has cost $O(d^2)$. Since this cost is embedded in an arithmetic series on the number of nodes, the worst-case running-time occurs when the number of nodes is maximized, i.e., when no nodes are merged. Therefore, the worst-case computational complexity for assimilating M tasks of length N is

$$O(\sum_{m=1}^{M} \sum_{n=1}^{N} mnd^2) \quad \in \quad O(M^2N^2d^2),$$

which is quadratic in the number of observations. □

3.7 Estimating CDHMM Parameters

In this section we present a simple one-shot procedure for estimating CDHMM parameters from an observation graph. We use one-shot estimation, in lieu of an EM-type approach, for computational expediency and because we typically lack the data required for complete EM re-estimation procedures. The EM algorithm is a two-step process. First, infer the values of hidden variables from the data set given parameter values, which is the Expectation (E) step. Next, use the estimated values of the hidden variables to compute the ML estimate of all parameters, which is the Maximization (M) step. This two-step process repeats until convergence in the likelihood of the data set (Boyles, 1983). Since the EM algorithm maximizes data likelihood, it will attempt to assign parameter values that cause the data likelihood to tend to infinity, particularly when data are scarce (Redner & Walker, 1984). For HMM training, this corresponds to states producing only the observed data, and having no generalization ability. However, there are well-known modifications for this problem, such as constraining the set of possible parameter values (Hathaway, 1986), but this can, in practice, be difficult to implement. The function `graph-to-HMM` in Figure 3.9 simply computes the maximum-likelihood CDHMM parameters from the information contained in the observation graph, which is a single "Maximization" step from the Baum-Welch algorithm, which is a special case of the EM algorithm. For example, the probability of transitioning from state q_i to state q_j is the number of times the observation graph recorded a transition from node v_i to node v_j, divided by the number of observations assigned to node v_i, written as $a_{j|i} = f_{e_{i\to j}}/|\boldsymbol{\mathcal{V}}_{v_i}|$. The initial-state probabilities are computed similarly. We do not assume any distribution for the observation-generation pdfs, $b_i(\boldsymbol{x})$, beyond requiring that the mean of the pdf be the sample mean of the observations assigned to the node, $\mathrm{E}_{\boldsymbol{x}}\{\boldsymbol{x}|q_i, \boldsymbol{\lambda}\} = \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle$. In practice, however, we typically fit a Gaussian distribution to the observations assigned to each node, which is a $O(d^2)$ computational-complexity procedure.

Lemma 3.6. `graph-to-HMM` *converts a $|V|$-node observation graph to a CDHMM with a running time of $O(|V|^2 + |V|b)$, where $O(b)$ is the cost of evaluating the parameters of the observation-generation pdf.*

Proof. There are two lines that dominate the function `graph-to-HMM` , creating the observation-generation pdf (Line `6`) and creating the state-transition pmf (Line `9`). For this analysis, we define the cost of Line `6` as $O(b)$. Because Line `6` executes once for each state, the total running time of Line `6` is $O(|V|b)$. Creating the state-transition pmf entry (Line `9`) is constant, but executes once for each state and loops over all states, giving a running time of $O(|V|^2)$. Since these lines execute in sequence, `graph-to-HMM` converts a $|V|$-node observation graph to a CDHMM with a running time of $O(|V|^2 + |V|b)$. □

Function `graph-to-HMM`
$G_{\boldsymbol{\mathcal{X}}} = (V, V^0, E, \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{V}}, f, g)$ is the observation graph.
M is the number of sequences assimilated into $G_{\boldsymbol{\mathcal{X}}}$.

```
 1: Q := ∅
 2: λ = (Q, X, a, b, π)
 3: for all v_i ∈ V
 4:    create new CDHMM state q_i
 5:    Q := Q ∪ {q_i}
 6:    create b_i from V_{v_i} with E_x{x|q_i, λ} = ⟨V_{v_i}⟩
 7:    π_i := g_{v_i}/M
 8:    for all v_j ∈ V
 9:       a_{j|i} := f_{e_{i→j}}/|V_{v_i}|
10:    end for all
11: end for all
12: return (Q, X, a, b, π)
```

Figure 3.9: Converting an observation graph to a CDHMM.

Algorithm `Learn-HMM`
$\boldsymbol{X} = \{\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots, \boldsymbol{X}^M\}$ is the multiset of all tasks.
$\epsilon \geq 0$ is a similarity threshold.

```
1: V := ∅, E := ∅
2: G_X := (V, V^0, E, X, V, f, g)
3: for all X^i ∈ {X^0, X^1, ..., X^M}
4:    G_X := assimilate-task (G_X, X^i, ε) (Figure 3.8)
5: end for all
6: λ := graph-to-HMM (G_X, M) (Figure 3.9)
```

Figure 3.10: The complete CDHMM-learning algorithm.

3.8 Bounding State Error

Assume that the observation sequences are actually generated by an unknown *target* CDHMM in an independent identically distributed (*iid*) manner. We derive a bound showing that, as more tasks are incorporated, states in the target CDHMM are estimated with asymptotically increasing accuracy. It is typical to provide HMM bounds regarding an observation *sequence* (Ephraim & Merhav, 2002), whereas we provide a bound on individual observations. This appears to be the strongest statement we can make, since our definition of similarity, Equation 3.1, is memoryless. First, we precisely define the idea that an estimated CDHMM will produce "close" to the correct observation at the correct time. Let c_n be the random variable denoting the current state of the CDHMM at time n.

Definition 3.2. *A CDHMM, $\boldsymbol{\lambda}_1$, γ-represents a state in another CDHMM, $q_2 \in \boldsymbol{\lambda}_2$, if there exists a state, $q_1 \in \boldsymbol{\lambda}_1$, at the same depth n such that*

$$\|\mathrm{E}_{\boldsymbol{x}}\{\boldsymbol{x}|c_n=q_1,\boldsymbol{\lambda}_1\}-\mathrm{E}_{\boldsymbol{x}}\{\boldsymbol{x}|c_n=q_2,\boldsymbol{\lambda}_2\}\|_{\boldsymbol{C}}<\gamma.$$

If $\boldsymbol{\lambda}_1$ γ-represents state q_2 in $\boldsymbol{\lambda}_2$ then we write $\boldsymbol{\lambda}_1 \xrightarrow{\gamma} q_2 \in \boldsymbol{\lambda}_2$. This defines what it means for different CDHMMs to generate observations within γ of each other at the correct time. Essentially, γ is a radius of acceptable error. Intuitively, it is easier to estimate a state in the target CDHMM if it generates observations more frequently. Furthermore, if a state emits observations with high variance, then it will be more difficult to distinguish between different states. This intuition is formalized in the following theorem.

Theorem 3.7. *Let $\widehat{\boldsymbol{\lambda}}$ be a CDHMM estimated from* `Learn-HMM` *with $\epsilon \geq 0$ and M* iid *tasks of length at least n, generated by some target CDHMM, $\boldsymbol{\lambda}^*$. If the state $q_* \in \boldsymbol{\lambda}^*$ generates observations with finite first and second moments, then*

$$\Pr\left\{\widehat{\boldsymbol{\lambda}} \xrightarrow{\gamma} q_* \in \boldsymbol{\lambda}^*\right\} \;>\; \left(1-(1-p_*)^M\right)\left(1-\frac{\mathrm{tr}[\boldsymbol{C}\mathrm{Var}(\boldsymbol{x}|q_*,\boldsymbol{\lambda}^*)]}{(\gamma-\sqrt{\epsilon})^2}\right),$$

where $\gamma > \sqrt{\epsilon}$, $p_* = \mathrm{P}(c_n=q_*|\boldsymbol{\lambda}^*) > 0$, and $\mathrm{Var}(\boldsymbol{x}|q_*,\boldsymbol{\lambda}^*)$ is the observation-generation variance of state $q_* \in \boldsymbol{\lambda}^*$.

Proof. Let m be the random variable summing the number of times $c_n = q_*$ over each of the M tasks. The probability that state q_*, with prior probability p_*, generated *at least* one observation at time n in M *iid* tasks is

$$\begin{aligned}\Pr\{m>0|p_*,M\} &= 1-\Pr\{m=0|p_*,M\}\\ &= 1-(1-p_*)^M.\end{aligned}$$

Let $\boldsymbol{x} \sim \mathrm{p}(\boldsymbol{x}|q_*, \boldsymbol{\lambda}^*)$, where $\mathrm{p}(\boldsymbol{x}|q_*, \boldsymbol{\lambda}^*)$ has finite mean $\boldsymbol{y}^*$ and finite variance $\boldsymbol{\Sigma}^*$. By the triangle inequality and reflexive property,

$$\|\boldsymbol{y}^* - \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle\|_{\boldsymbol{C}} \quad \leq \quad \|\boldsymbol{x} - \boldsymbol{y}^*\|_{\boldsymbol{C}} + \|\boldsymbol{x} - \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle\|_{\boldsymbol{C}} \,.$$

From Theorem 3.3, $\|\boldsymbol{x} - \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle\|_{\boldsymbol{C}} \leq \sqrt{\epsilon}$, for any $\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}$. Since $\mathrm{E}_{\boldsymbol{x}}\left\{\boldsymbol{x}|q_i, \widehat{\boldsymbol{\lambda}}\right\} = \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle$ (`graph-to-HMM` Line 6), the state $q_* \in \boldsymbol{\lambda}^*$ will be γ-represented by the estimated CDHMM if q_* generates at least one observation such that

$$\|\boldsymbol{x} - \boldsymbol{y}^*\|_{\boldsymbol{C}} \quad \leq \quad \gamma - \sqrt{\epsilon}.$$

By assumption $\gamma > \sqrt{\epsilon}$, and from the Multivariate Chebyshev's Inequality (Section A.1.6),

$$\Pr\left\{\|\boldsymbol{x} - \boldsymbol{y}^*\|_{\boldsymbol{C}} < \gamma - \sqrt{\epsilon}\right\} \quad > \quad 1 - \frac{\mathrm{tr}[\boldsymbol{C}\boldsymbol{\Sigma}^*]}{(\gamma - \sqrt{\epsilon})^2}.$$

Multiplying the prior by this conditional probability yields

$$\Pr\left\{\widehat{\boldsymbol{\lambda}} \xrightarrow{\gamma} q_* \in \boldsymbol{\lambda}^*\right\} \quad > \quad (1 - (1-p_*)^M)\left(1 - \frac{\mathrm{tr}[\boldsymbol{C}\boldsymbol{\Sigma}^*]}{(\gamma - \sqrt{\epsilon})^2}\right),$$

which completes the claim. □

Perhaps a picture (Figure 3.11) can best illustrate the mechanics of Theorem 3.7. In the multidimensional case, the circles are actually hyperellipses determined by the eigendecomposition of the precision matrix $\boldsymbol{C}$. For clarity in this example, we will refer to the hyperellipses as circles. The analysis begins by constructing several circles. One circle is centered at the mean of the observations from target state q_*. The radius, γ, is determined by the precision matrix $\boldsymbol{C}$ and the variance of observations, $\mathrm{Var}(\boldsymbol{x}|q_*, \boldsymbol{\lambda}^*)$. Another circle has unknown origin, $\langle \boldsymbol{\mathcal{V}}_{v_i} \rangle$, determined through a sort of worst-case analysis. From Theorem 3.3, all observations assigned to node v_i lie within a circle of radius $\sqrt{\epsilon}$. Provided that $\gamma > \sqrt{\epsilon}$, it is possible that, even in the worst case, an observation within a radius of $\gamma - \sqrt{\epsilon}$ from the origin $\mathrm{E}_{\boldsymbol{x}}\{\boldsymbol{x}|q_*, \boldsymbol{\lambda}^*\}$ can belong to node v_i within a circle of radius γ. Because of this, we bound the probability that the state q_* emits observations inside the circle of radius $\gamma - \sqrt{\epsilon}$. This bound does not assume an underlying form on the observation-generation distributions for the target CDHMM, $\mathrm{p}(\boldsymbol{x}|q_*, \boldsymbol{\lambda}^*)$, and is proportional to the eigenvalues of the variance of the distribution. In more concrete terms,

$$\Pr\left\{\widehat{\boldsymbol{\lambda}} \xrightarrow{\gamma} q_* \in \boldsymbol{\lambda}^*\right\} \quad > \quad \underbrace{\left(1 - (1-p_*)^M\right)}_{(\star)} \underbrace{\left(1 - \frac{\mathrm{tr}[\boldsymbol{C}\mathrm{Var}(\boldsymbol{x}|q_*, \boldsymbol{\lambda}^*)]}{(\gamma - \sqrt{\epsilon})^2}\right)}_{(\diamond)}.$$

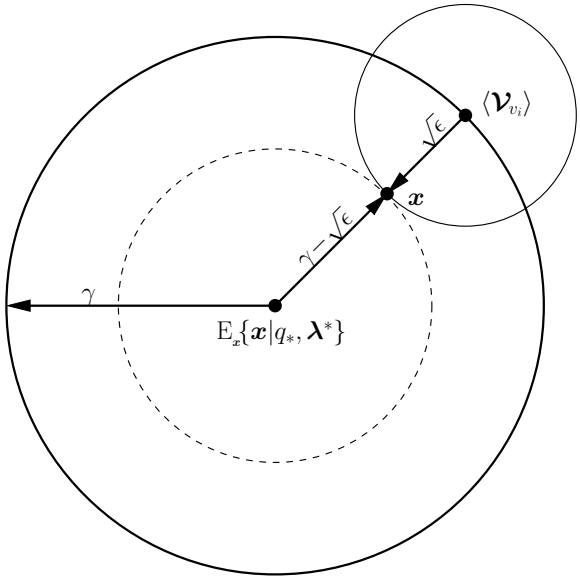


Figure 3.11: Two-dimensional illustration of Theorem 3.7.

The term $(\diamond)$ is in many ways an "inherent" error rate, which decreases as the

- observation variance decreases;
- similarity radius (ϵ) decreases;
- acceptable error (γ) increases.

The term $(\star)$ increases asymptotically as the number of tasks M increases at a rate determined by the prior probability p_* that the state q_* generates an observation. This implies that the bound in Theorem 3.7 asymptotes to the inherent error rate of the target CDHMM, as in Figure 3.12, and the estimated CDHMM probably generates the correct observation at the correct time.

We would like to derive the converse bound that each state in the estimated CDHMM is γ-represented by a state in the target CDHMM. Intuitively, if a state is assigned more observations, the probability increases that it is "close" to a state in the target CDHMM. In Theorem 3.8, we derive a bound showing that an estimated state is closer to a target state as it is assigned more observations.

Theorem 3.8. *Let $\widehat{\boldsymbol{\lambda}}$ be a CDHMM estimated from algorithm* `Learn-HMM` *with similarity threshold $\epsilon \geq 0$ and M* iid *tasks generated from some target CDHMM, $\boldsymbol{\lambda}^*$. Let state $q_i \in \widehat{\boldsymbol{\lambda}}$ result from node v_i in the observation graph with $|\boldsymbol{\mathcal{V}}_{v_i}|$ observations. Then*

$$\Pr\left\{\boldsymbol{\lambda}^* \xrightarrow{\gamma} q_i \in \widehat{\boldsymbol{\lambda}}\right\} \;>\; 1 - \left(\frac{\mathrm{tr}[\boldsymbol{C}\boldsymbol{\Sigma}_*]}{(\gamma - \sqrt{\epsilon})^2}\right)^{|\boldsymbol{\mathcal{V}}_{v_i}|},$$

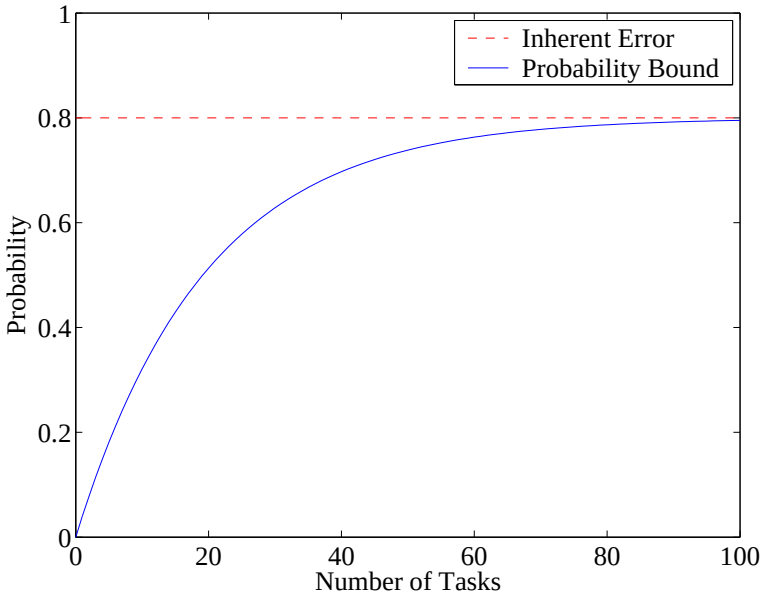


Figure 3.12: Asymptotic nature of the bound in Theorem 3.7 for a fixed inherent error rate with $p_* = 0.05$, $\gamma = 1$, *and* $\epsilon = 0.25$.

where $\boldsymbol{\Sigma}_*$ *is the maximum observation-generation variance of any state in the target CDHMM* $\boldsymbol{\lambda}^*$.

The proof is given in Section A.1.5.

3.9 Relationship to Maximum *A Posteriori* Structure Estimation

Stolcke and Omohundro (1994b) introduced the idea of using Maximum *A Posteriori* (MAP) structure estimation to derive a best-first state-merging HMM algorithm. With this scheme, the algorithm merges two HMM states with the largest increase, if any, in the posterior likelihood. In this framework, the objective is to maximize the *a posteriori* likelihood of prediction, conditioned on past observations,

$$\mathrm{p}\big(\boldsymbol{x}_n^c|\boldsymbol{X}_{0:n-1}^c,\boldsymbol{\lambda}\big)\,\mathrm{p}\big(\boldsymbol{x}_{n-1}^c|\boldsymbol{X}_{0:n-2}^c,\boldsymbol{\lambda}\big)\cdots\mathrm{p}(\boldsymbol{x}_1^c|\boldsymbol{x}_0^c,\boldsymbol{\lambda})\,\mathrm{p}(\boldsymbol{x}_0^c|\boldsymbol{\lambda}) \quad = \quad \mathrm{p}(\boldsymbol{X}_{0:n}^c|\boldsymbol{\lambda})\,.$$

Using the Bayesian formulation,

$$\begin{aligned}\boldsymbol{\lambda}^* &= \arg\max_{\boldsymbol{\lambda}} \frac{\mathrm{p}\big(\{\boldsymbol{X}^0,\boldsymbol{X}^1,\ldots,\boldsymbol{X}^M\}|\boldsymbol{\lambda}\big)\,\mathrm{p}(\boldsymbol{\lambda})}{\mathrm{p}(\{\boldsymbol{X}^0,\boldsymbol{X}^1,\ldots,\boldsymbol{X}^M\})} \\ &= \arg\max_{\boldsymbol{\lambda}} \mathrm{p}\big(\{\boldsymbol{X}^0,\boldsymbol{X}^1,\ldots,\boldsymbol{X}^M\}|\boldsymbol{\lambda}\big)\,\mathrm{p}(\boldsymbol{\lambda})\,.\end{aligned}$$

3.9.1 Conditional Prediction Likelihood

If the tasks are *iid*, then the conditional likelihood of a CDHMM, $\boldsymbol{\lambda}$, generating the tasks is

$$\begin{aligned} \mathrm{p}\big(\{\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots, \boldsymbol{X}^M\}|\boldsymbol{\lambda}\big) &= \prod_{c=1}^{M} \mathrm{p}(\boldsymbol{X}^c|\boldsymbol{\lambda}) \\ &= \prod_{c=1}^{M} \sum_{q_i \in \mathcal{Q}} \mathrm{p}(c_N{=}q_i, \boldsymbol{X}^c|\boldsymbol{\lambda}) \\ &\doteq \prod_{c=1}^{M} \sum_{q_i \in \mathcal{Q}} \alpha_N(i), \end{aligned} \tag{3.2}$$

where $\alpha_N(i)$ are the HMM "forward variables" (Rabiner, 1989). However, the computation of Equation 3.2 requires Dynamic-Programming (DP) complexity, over each task assimilated into the model. Since Equation 3.2 will be evaluated for each hypothesis CDHMM, this cost can be prohibitive in any state-merging scheme. In the speech-recognition field, it is common to avoid the DP complexity of Equation 3.2 by approximating it with the *single* best path through the HMM, known as the Viterbi Approximation,

$$\begin{aligned} \widehat{\alpha}^c_{\boldsymbol{\lambda}} &\triangleq \max_{q_0,\ldots,q_N} \mathrm{p}(\boldsymbol{x}^c_N|c_N{=}q_N, \boldsymbol{\lambda})\, \mathrm{P}(c_N{=}q_N|c_{N-1}{=}q_{N-1}, \boldsymbol{\lambda}) \cdots \mathrm{p}(\boldsymbol{x}^c_0|c_0{=}q_0, \boldsymbol{\lambda})\, \mathrm{P}(c_0{=}q_0|\boldsymbol{\lambda}) \\ &= \max_{q_0,\ldots,q_N} \mathrm{p}(\boldsymbol{X}^c, q_0, ..., q_N|\boldsymbol{\lambda}) \\ &\cong \sum_{q_0,\ldots,q_N} \mathrm{p}(\boldsymbol{X}^c, q_0, ..., q_N|\boldsymbol{\lambda}) \\ &= \mathrm{p}(\boldsymbol{X}^c|\boldsymbol{\lambda}) \\ &\doteq \sum_{q_i \in \mathcal{Q}} \alpha_N(i). \\ &\Rightarrow \\ \prod_{m=1}^{M} \widehat{\alpha}^m_{\boldsymbol{\lambda}} &\cong \mathrm{p}\big(\{\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots, \boldsymbol{X}^M\}|\boldsymbol{\lambda}\big)\,. \end{aligned}$$

In other words, the Viterbi Approximation assigns each observation to a single state in the HMM and multiplies the observation likelihood of that state by the corresponding transition probabilities. Though Stolcke and Omohundro (1994b) mentioned applying the Viterbi Approximation for computational considerations, we derive the MAP structure-estimation algorithm from a measure-theoretic point of view, which naturally leads to the Viterbi Approximation. Let the multiset of observations assigned to state q_i by the Viterbi Approximation be $\boldsymbol{\mathcal{V}}_{q_i}$ and

assume each state in the HMM has a Gaussian distribution with constant covariance,

$$\begin{aligned} b_i(\boldsymbol{x}) &\sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}), \\ \log b_i(\boldsymbol{x}) &= A - \frac{1}{2} \|\boldsymbol{x}-\boldsymbol{\mu}_i\|^2_{\boldsymbol{\Sigma}^{-1}} . \end{aligned}$$

Theorem 3.9. *Let state $q_k \in \widetilde{\boldsymbol{\lambda}}$ be the result of merging state q_i and state q_j in the CDHMM $\boldsymbol{\lambda}$, from M* iid *tasks and constant-variance Gaussian states. The log conditional-likelihood ratio is then*

$$\begin{aligned} \log \prod_{m=1}^{M} \widehat{\alpha}^m_{\widetilde{\boldsymbol{\lambda}}} - \log \prod_{m=1}^{M} \widehat{\alpha}^m_{\boldsymbol{\lambda}} &= \mu_{\boldsymbol{\Sigma}^{-1}}(\boldsymbol{\mathcal{V}}_{q_k}, \langle \boldsymbol{\mathcal{V}}_{q_k} \rangle) - \mu_{\boldsymbol{\Sigma}^{-1}}(\boldsymbol{\mathcal{V}}_{q_i}, \langle \boldsymbol{\mathcal{V}}_{q_i} \rangle) - \mu_{\boldsymbol{\Sigma}^{-1}}(\boldsymbol{\mathcal{V}}_{q_j}, \langle \boldsymbol{\mathcal{V}}_{q_j} \rangle) \\ &\quad + \sum_{q_l \in \mathcal{Q}} \left(f_{e_{l\to i}} \log \frac{f_{e_{l\to i}} + f_{e_{l\to j}}}{f_{e_{l\to i}}} + f_{e_{l\to j}} \log \frac{f_{e_{l\to i}} + f_{e_{l\to j}}}{f_{e_{l\to j}}} \right) \\ &\quad + \sum_{q_l \in \mathcal{Q}} \left(f_{e_{i\to l}} \log \frac{\xi_i}{\xi_i + \xi_j} \frac{f_{e_{i\to l}} + f_{e_{j\to l}}}{f_{e_{i\to l}}} + f_{e_{j\to l}} \log \frac{\xi_j}{\xi_i + \xi_j} \frac{f_{e_{i\to l}} + f_{e_{j\to l}}}{f_{e_{j\to l}}} \right) \end{aligned} \tag{3.3}$$

using the sufficient statistics of Equation 3.1 from Lemma 3.4.

The proof is given in Section A.1.7.

The log conditional-likelihood ratio in Equation 3.3 is very close to our similarity measure, Equation 3.1. However, Equation 3.3 contains two extra terms that reward the algorithm for merging states that cause a reduction in the number of edges in the CDHMM. This will necessarily result in merging selections different from `Learn-HMM` and will, most likely, result in more states for a given data set, which we show empirically in Section 4.4.1.

3.9.2 Model Prior

Stolcke and Omohundro (1994b) suggest the model-description length as the CDHMM prior, $\mathrm{p}(\boldsymbol{\lambda})$, which gives preference to HMMs with fewer states and transitions, implicitly implementing an "Occam factor." The derivation of the prior is given in Section A.2.1, but is summarized here as follows:

$$\mathrm{p}(\boldsymbol{\lambda}) \;\widetilde{\propto}\; |\mathcal{Q}|^{-|E|} \prod_{q_i \in \mathcal{Q}} |\mathrm{Var}(\boldsymbol{x}|q_i)\,|^{-1}. \tag{3.4}$$

Let state q_k in CDHMM $\widetilde{\boldsymbol{\lambda}}$ result from merging state q_i and state q_j in CDHMM $\boldsymbol{\lambda}$. Let $|\mathcal{Q}|$ and $|E|$ be the number of states and edges in CDHMM $\boldsymbol{\lambda}$, respectively. Let $|\widetilde{E}|$ be the number of edges in CDHMM $\widetilde{\boldsymbol{\lambda}}$, which has $|\mathcal{Q}| - 1$

states. The log prior-likelihood ratio is

$$\log \mathrm{p}\Big(\widetilde{\boldsymbol{\lambda}}\Big) - \log \mathrm{p}(\boldsymbol{\lambda}) \;\; \cong \;\; |E| \log |\mathcal{Q}| - |\widetilde{E}| \log(|\mathcal{Q}|-1) + |\mathrm{Var}(\boldsymbol{x})\,|. \tag{3.5}$$

Using the log conditional-likelihood ratio from Equation 3.3 and the prior from Equation 3.5, we can compute the log posterior-likelihood ratio

$$\log \mathrm{p}\Big(\widetilde{\boldsymbol{\lambda}}|\boldsymbol{X}\Big) - \log \mathrm{p}(\boldsymbol{\lambda}|\boldsymbol{X}) \;\; \cong \;\; \left(\log \prod_{m=1}^{M} \widehat{\alpha}_{\widetilde{\boldsymbol{\lambda}}}^{m} - \log \prod_{m=1}^{M} \widehat{\alpha}_{\boldsymbol{\lambda}}^{m}\right) + \Big(\log \mathrm{p}\Big(\widetilde{\boldsymbol{\lambda}}\Big) - \log \mathrm{p}(\boldsymbol{\lambda})\Big)\,. \tag{3.6}$$

The MAP structure-estimation algorithm finds the states q_i and q_j such that Equation 3.6 is maximized. If that value is greater than zero, then state q_i and state q_j are merged. This process repeats until there are no more state-merging candidates that increase the posterior. In Section 4.4.1, we compare the empirical performance of the MAP best-first state-merging algorithm to our CDHMM structure-estimation learning algorithm, `Learn-HMM`.

3.10 Predicting Observations

In the prediction phase, we use the CDHMM estimated by our learning algorithm to compute predictions of future observations. Optimal prediction, given a model, is a mature topic and can be found in many references (Duda et al., 2001). There are several reasonable choices for a prediction criterion, such as Maximum Likelihood (ML), Maximum *A Posteriori*, expectation, etc. In our experiments, ML estimators have performed the best. To compute a prediction, we condition the CDHMM probability distributions on observations from the current task and determine the most likely next observation,

$$\widehat{\boldsymbol{x}}_n^* \;\; = \;\; \arg\max_{\boldsymbol{x}_n} \mathrm{p}\big(\boldsymbol{x}_n|\boldsymbol{X}_{0:n-1}^c, \boldsymbol{\lambda}\big)\,,$$

where $\boldsymbol{X}_{0:n-1}^c = \{\boldsymbol{x}_0^c, \ldots, \boldsymbol{x}_{n-1}^c\}$ is the sequence of observations from the current task. In order to compute this estimator, we need many "stepping stones." The complete derivation is given in Section A.2.2, but we summarize it here. Using standard HMM notation (Rabiner, 1989), we define the "forward variables" to be the conjunctive likelihood of being in state q_j at time n while observing the current task,

$$\begin{aligned} \alpha_n(j) \;\; &\triangleq \;\; \mathrm{p}(c_n{=}q_j, \boldsymbol{X}_{0:n}^c|\boldsymbol{\lambda}) \\ &\doteq \;\; \begin{cases} b_j(\boldsymbol{x}_n) \sum\limits_{q_i \in \mathcal{Q}} a_{j|i}\alpha_{n-1}(i), & n > 0 \\ b_j(\boldsymbol{x}_0)\pi_j, & n = 0 \end{cases}. \end{aligned} \tag{3.7}$$

The probability of being in state q_j at time n, given observations from the current task up to time $n-1$, is the pmf

$$\begin{aligned} \nu_n(j) &\triangleq \mathrm{P}\big(c_n{=}q_j|\boldsymbol{X}_{0:n-1}^c, \boldsymbol{\lambda}\big) \\ &\doteq \frac{\sum\limits_{q_i\in\mathcal{Q}} a_{j|i}\alpha_{n-1}(i)}{\sum\limits_{q_k\in\mathcal{Q}} \alpha_{n-1}(k)}. \end{aligned} \tag{3.8}$$

Lemma 3.10. *The cost of computing $\nu_n(j)$ for all states $q_j \in \boldsymbol{\lambda}$ is $O(n|\mathcal{Q}|^2 b)$, where $|\mathcal{Q}|$ is the number of states in the CDHMM and $O(b)$ is the cost of evaluating an observation pdf.*

Proof. This DP complexity is due to the n-step recursion forming an arithmetic series over each state in the CDHMM. Since each step requires evaluating an observation pdf, which is of $O(b)$ complexity, the running time is $O(n|\mathcal{Q}|^2 b)$. □

With this notation, we evaluate the ML prediction as

$$\widehat{\boldsymbol{x}}_n^* \doteq \arg\max_{\boldsymbol{x}_n} \sum_{q_j\in\mathcal{Q}} b_j(\boldsymbol{x}_n)\nu_n(j). \tag{3.9}$$

It is also straightforward to derive predictions for any time in the future. Equation 3.9 involves the maximization of a linear combination of nonlinear functions. Except in degenerate cases, Equation 3.9 cannot be solved in closed form and we must resort to iterative multivariate maximization techniques, which can be relatively costly (Bertsekas, 1995). However, in practice, a locally optimal solution can be found quickly.

3.11 Prediction Confidence

Regardless of the criterion used (ML, expectation, etc.), a prediction will exist even if the current task is not consistent with the estimated CDHMM. This may result in computing inaccurate predictions. Ideally, the system would suggest only the most accurate predictions to users. However, the target observation is not known at the time of the prediction, so we indicate the *confidence*, $\phi_n \in [0, 1]$, based on information available at the time of prediction. Confidence of $\phi_n = 1$ indicates that observations from the current task fit perfectly with the model, while $\phi_n = 0$ indicates minimum certainty. To this end, we compute the divergence of the CDHMM from complete internal uncertainty while observing the current task. Let the number of states in the CDHMM be $|\mathcal{Q}| \geq 2$. In our work, we define confidence as the Kullback-Leibler divergence taken log base $|\mathcal{Q}|$ between the next-state random

variable, c_n, and the uniform distribution

$$\begin{aligned}
\phi_n &\triangleq \frac{\mathcal{D}_{\mathrm{KL}}(c_n \parallel \frac{1}{|\mathcal{Q}|})}{\log_2 |\mathcal{Q}|} \\
&= \frac{\sum\limits_{q_j \in \mathcal{Q}} \nu_n(j) \log_2(|\mathcal{Q}| \nu_n(j))}{\log_2 |\mathcal{Q}|} \\
&= \frac{\log_2 |\mathcal{Q}| \sum\limits_{q_j \in \mathcal{Q}} \nu_n(j)}{\log_2 |\mathcal{Q}|} + \frac{\sum\limits_{q_j \in \mathcal{Q}} \nu_n(j) \log_2 \nu_n(j)}{\log_2 |\mathcal{Q}|} \\
&= 1 - \frac{\sum\limits_{q_j \in \mathcal{Q}} -\nu_n(j) \log_2 \nu_n(j)}{\log_2 |\mathcal{Q}|} \\
&= 1 - \frac{H(c_n)}{\log_2 |\mathcal{Q}|},
\end{aligned} \tag{3.10}$$

where $H(\cdot)$ is entropy. Since c_n is a discrete random variable with $|\mathcal{Q}| \geq 2$ possibilities, the confidence is bounded on the closed interval $0 \leq \phi_n \leq 1$ for any state distribution and any observation sequence. Intuitively, if many states in the CDHMM are likely to produce the next observation, then the prediction confidence will be low. On the other hand, a prediction based on the contributions from few states will result in high confidence. As mentioned earlier, the ideal case is if prediction confidence correlates with prediction error with a normalized correlation coefficient of -1. On real-world data (Section 4.3.3), Equation 3.10 has a significant negative normalized correlation coefficient with prediction error, with $\rho = -0.89$.[1] Using this confidence value, we can infer which predictions will be more accurate in a causal manner.

3.12 Probability of Similarity

To make the experimental results easier to present, we compress the infinite interval, $\epsilon \in [0, \infty)$, to a bounded interval, $\delta \in (0, 1]$. There are many possibilities to determine δ, and we assume that users make positioning errors about a desired observation according to a Gaussian distribution. We then require that an observation be emitted with high probability,

$$\Pr\left\{ \|\boldsymbol{x} - \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle\|_{\boldsymbol{C}}^2 \leq \epsilon \right\} \quad > \quad 1 - \delta. \tag{3.11}$$

[1] We have been unable to derive a proof guaranteeing a negative correlation, though a proof does seem possible.

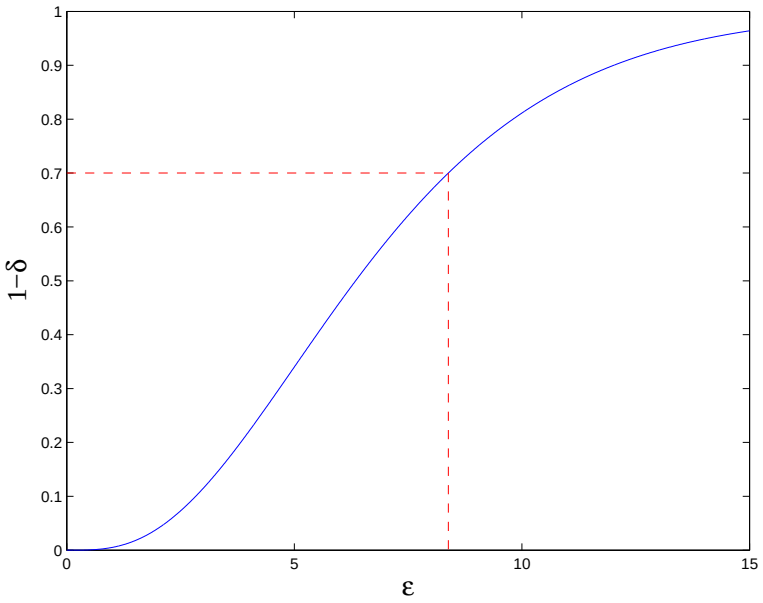


Figure 3.13: Cumulative Distribution Function of a chi-square random variable with 7 degrees of freedom, χ^2_7. Graphically, we compute Equation 3.11 by fixing a value of, e.g., $1-\delta = 0.7$ and finding the intercept on the cdf at $\epsilon \approx 8.4$.

The squared Mahalanobis distance of a Gaussian random variable is a chi-square random variable. Using this formulation, we evaluate Equation 3.11 from the cumulative distribution function (cdf) of the chi-square distribution,

$$\epsilon \in \left\{ w \left| \delta = 1 - \frac{\gamma(\frac{d}{2}, \frac{w}{2})}{\gamma(\frac{d}{2}, 0)} \right. \right\}, \tag{3.12}$$

where $\gamma(\cdot,\cdot)$ is the incomplete gamma function. Since any chi-square pdf is positive on the interval $(0,\infty)$, its cdf will be a bijection on $(0,\infty) \rightarrow (0,1]$. The solution to Equation 3.12 is shown graphically in Figure 3.13. As $\delta \rightarrow 0$, the definition of similarity becomes loose, $\epsilon \rightarrow \infty$, inducing a simpler CDHMM. As $\delta \rightarrow 1$, the definition of similarity becomes strict, $\epsilon \rightarrow 0$, inducing a more complex CDHMM. From this perspective, δ can be considered as a type of "complexity parameter." The experimental results in this work will vary the parameter δ.

3.13 Relationship to Learning By Observation

The CDHMM structure-estimation learning algorithm forms the substrate of our LBO system. As used in our system, we estimate the structure of a CDHMM that describes the *subgoals* of a demonstration. This estimated CDHMM can then be used to predict future subgoals (Chapter 4) or to complete entire tasks (Chapter 6). Because

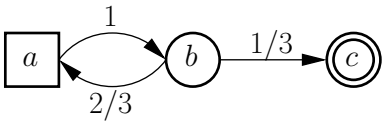


Figure 3.14: Finite-looping tasks are not well represented by our learning algorithm.

the data are sparse in LBO, the algorithm estimates the simplest CDHMM that the data suggest, using a similarity-based merging criterion, from a memoryless definition of similarity. Since the running time of the algorithm is low, worst-case quadratic, it should be able to operate in real time.

3.14 Summary

We have derived a new learning algorithm that uses a state-merging scheme to estimate the structure of CDHMMs based on observations from a target CDHMM. Given an ordered set of observations, the CDHMM is uniquely determined by the parameter $\epsilon \geq 0$, which is a function of the parameter $\delta \in (0, 1]$ (cf. Section 3.12). Decreasing the parameter, $\delta \to 0$, induces a simpler CDHMM, while increasing the parameter, $\delta \to 1$, induces a more complex CDHMM. We showed that the algorithm produces only CDHMMs with a locally minimal number of states, and the worst-case computational complexity is quadratic in the number of observations. We also derived a bound showing that the estimate of the target CDHMM improves as more observations are incorporated. Predictions of future observations are computed from an ML estimation procedure, Equation 3.9, by conditioning the CDHMM probability distributions on observations from the current task. In order to avoid burdening the user with inaccurate suggestions, the system suggests only predictions with high *confidence*, ϕ_n (Equation 3.10).

Despite different initial formulations, our best-first similarity-based state-merging algorithm has much in common with the best-first MAP state-merging algorithm of Stolcke and Omohundro (1994b). Our merging criterion, Equation 3.1, implicitly assumes a constant-variance Gaussian distribution for each state in the CDHMM, while, in principle, the MAP formulation allows arbitrary distributions. Furthermore, our merging criterion is *memoryless*, whereas the MAP formulation implicitly considers the similarity of ancestor and descendant observations. Our formulation lends itself to a theoretical analysis regarding computational requirements and estimation performance, which is difficult for the MAP formulation due to the application-specific distributions placed on the system.

In the introduction, we posed several questions about the ability of our algorithm to model user subgoal selection (Section 1.3.1). Specifically, the first question was:

- What tasks can be automated?

In principle, because our learning algorithm constructs CDHMMs with general underlying graphs (e.g., cyclic,

self-looping, or acyclic), any task can be represented by our algorithm. The most difficult tasks for our system to automate are those that contain finite looping. For example, assume that some task loops three times ($a \rightarrow b$) followed by a termination sequence (c), so that we have the observation sequence $\{a, b, a, b, a, b, c\}$. Because our learning algorithm estimates the structure of CDHMMs using a memoryless merging criterion, the ML prediction following "b" will always be "a," and never the termination "c," as shown in Figure 3.14. Despite this limitation, our structure-estimation algorithm still automates real-world tasks effectively, which we show experimentally in the following chapter. The second question asked in the introduction was:

- What computing resources are needed to estimate the model?

In Theorem 3.5, we showed that the worst-case running time of the algorithm was quadratic in the number of observations. This is an appealing limit but, as is the case in all *big-oh* analyses, it still remains to be seen how this translates into running-time on a real computer, which we show in the following chapter. The third question asked in the introduction was:

- How many demonstrations are needed to achieve a desired performance?

In Theorem 3.7, we derived a bound showing that, as more tasks are incorporated, states in the target CDHMM are estimated with asymptotically increasing accuracy. To solve for the number tasks needed to achieve a desired performance is simply a matter of rearranging the equations to solve for the number of tasks as a function of performance. The importance of Theorem 3.7 is that the state-estimation bound increases asymptotically, not in its numeric value. This bound was derived from Markov's inequality (Section A.1.6), which is one of the most general bounds possible on random variables and, practically speaking, the numeric value of bounds derived from Markov's inequality will be hopelessly loose, and ours is no exception. But Theorem 3.7 gives a bound on the number of demonstrations needed to achieve a desired performance.

Chapter 4

Predictive Robot Programming

In this chapter, we analyze the performance of the learning algorithm in modeling user subgoal selection with Predictive Robot Programming (PRP). This application gives the ability to test the performance of the system by removing sensor noise and environment considerations. The PRP system constructs a CDHMM model of user subgoal selection by incorporating information from previously completed tasks. With this model, the PRP system computes predictions about where users will move the robot. Users can reduce programming time by allowing the PRP system to complete the task automatically. We analyze the performance of the PRP system on two sets of data. The first set is based on data from complex, real-world robotic tasks. We show that the PRP system is able to compute predictions for about 25% *of the waypoints with a median prediction error less than* 0.5% *of the distance traveled during prediction. We also present laboratory experiments showing that the PRP system results in a significant reduction in programming time, with users completing simple robot-programming tasks over* 30% *faster when using the PRP system to compute predictions of future waypoints.*

4.1 Introduction

Programming a manipulator robot is an arduous task. A robot program typically consists of three main components: a sequence of positions through which the robot must travel, conditional branching statements, and process-specific instructions. Of these components, robot programmers usually spend the majority of their time defining the sequence of positions, called *waypoints*. While critical to the success of all robot programs, specifying waypoints is currently an overly complex and time-consuming process. Consequently, one of the main inhibitors of robotic automation is the time needed to program the manipulator.

Industrial robot programming has evolved into two mutually exclusive methods, offline and online programming, each having its advantages and disadvantages. In offline programming, users move a simulated robot to each waypoint with a simulated model of the workspace. Offline-programming packages allow users to design robot programs in simulation without bringing down production and can optimize programs according to almost any imaginable criterion. Typical optimizations involve production speed, material usage, or power consumption. To achieve the high accuracy required in many applications, the physical workspace must be well calibrated with respect to the simulated environment. Otherwise, extensive online fine-tuning will be needed, which detracts from the largest benefit of offline programming, namely, lack of production downtime. Offline systems generally require that programs be written in a sophisticated procedural-programming language. Consequently, users of these systems must be experts at the industrial process as well as at computer programming. Indeed, expertise in either of these fields is a job skill in its own right.

Despite the advantages of offline packages, online programming is more commonly used in practice. In online programming, an actual part is placed in the workspace exactly as it would be during production. Users create robot programs by moving the end-effector between waypoints with some type of control device, typically a joystick or push buttons. Even though online systems generate procedural-programming code, users can create robot programs without editing this code, and it is typically viewed as less intimidating and more intuitive than offline programming. One potential disadvantage of online systems is that production and programming cannot occur in parallel; production must be halted during reprogramming. If reprogramming cannot be completed during normal downtime, such as weekends, then cost will be incurred in the form of lost production. Therefore, the set of tasks viable for online programming is constrained by programming time. A reduction in this time would allow its use in areas previously off limits.

In this chapter, we present a novel Predictive Robot-Programming (PRP) system[1] that allows users to leverage their previous work to decrease future programming time. Specifically, this system assists users by predicting where they may move the end-effector and automatically positions the robot at the estimated waypoint. The PRP idea of automatically completing a task based on a few observations is conceptually similar to word-completion routines in word-processing programs and text-messaging in mobile phones. In that application, users are presented with a word completion based on a few keystrokes, in the hope of reducing the time needed to type a message. The domain of word completion is well defined: it is a dictionary. In PRP, there are an uncountable number of tasks that a manipulator robot may perform and no corpora of example programs. Further complicating any prediction scheme are the inherent imprecision and poor repeatability of humans and the tendency of users to perform the same task in different manners. Consider the case of welding the joints of a rectangle, where if the goal is to weld the object together, then the corners may be welded in any order. Such ambiguity complicates any prediction scheme. Our PRP system addresses this uncertainty during both modeling and prediction.

[1] Our PRP system is called Adaptive Experience Suggestion, Observation, and Prediction (AESOP) by the ABB Corporation.

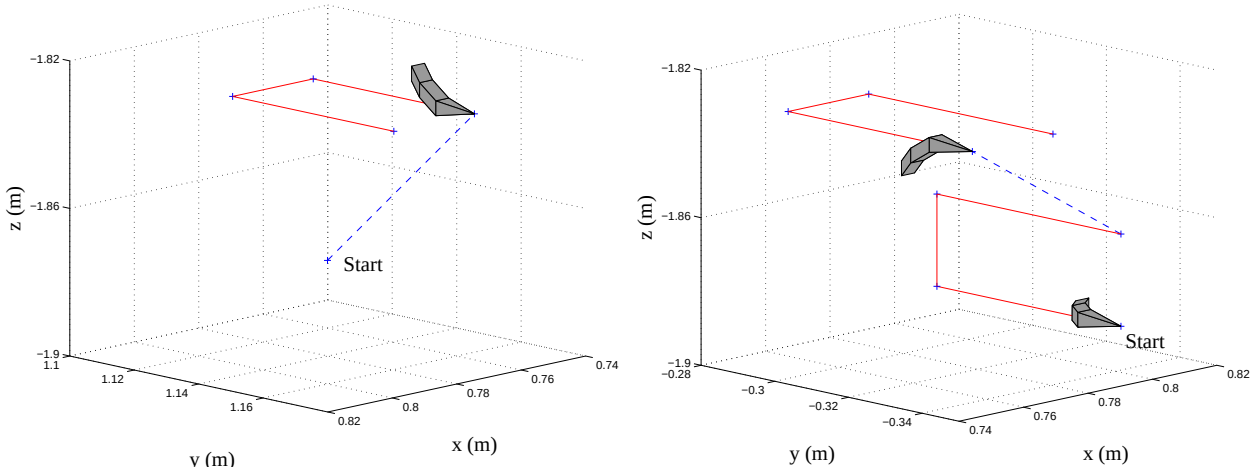


Figure 4.1: Waypoints from two subroutines in the same robot program. While they are different at the subroutine level, there are repeated subtasks occurring in translated and rotated form, such as the "U-shaped" pattern. The relative movement of the robot with respect to the end-effector during these subtasks is the same.

Manipulator robots perform a wide variety of industrial applications; as the capabilities of robots increase, they are performing more sophisticated tasks. Simple tasks can take days to program, while more complex tasks can take weeks or months. Despite their complexity, most tasks can usually be decomposed into simpler subtasks, which may be repeated throughout the program directly or in some modified form. Most robot programmers either do not recognize the similarity or create these subtasks from scratch each time due to the cumbersome nature of current programming environments. For example, in Figure 4.1 we show the waypoints from two different subroutines in an arc-welding robot program. While they are clearly different at the subroutine level, there are repeated subtasks, such as a "U-shaped" pattern, which is rotated differently in the two subroutines. However, the patterns have the same movements with respect to the end-effector. Such repeated subtasks tend to be specific to a particular robot program. That is, the similarity arises from the physical workpiece and the industrial process at hand. If a different workpiece is supplied or a different industrial process employed, then the pattern of similarity would change. While users are creating waypoints, the PRP system must be able to identify similarities, if any, to the many previously created subtasks and then suggest future waypoints. If users allow the PRP system to move the robot, then the end-effector is automatically positioned at the predicted waypoint. Compared to the time needed to move a robot manually, automatic positioning of a robot is essentially instantaneous, which reduces overall programming time. The prediction process can execute in a closed- or open-loop fashion. During closed-loop operation, the PRP system moves the robot to a single predicted position and users may fine-tune the waypoint with a conventional control device. In the open-loop mode, the PRP system automatically completes the task for users without adjustment to the waypoints. Both prediction modes have their advantages and disadvantages.

Closed-loop prediction tends to produce more accurate waypoints, because user feedback reduces error. Open-loop prediction is faster, since the PRP system does not have to wait for user intervention.

4.2 Representing Waypoints

Using our terminology, the task subgoals are the waypoints of the program. General-purpose six-degree-of-freedom (6DOF) manipulators can be described by a Cartesian-space location and orientation of the end-effector. In this work, the surjective Cartesian-space description is preferable to the bijective joint-space representation, because the Cartesian-space description forms a kinematics-free coordinate transform. With the use of this Cartesian-space end-effector description, a waypoint is given by the homogeneous coordinate transform matrix ${}^{n}\mathbf{w}_{G}$ that maps some global reference frame G to the frame of the nth waypoint. A robot program can be described by the sequence of waypoints $\mathbf{W} = \{{}^{G}\mathbf{w}_0, {}^{G}\mathbf{w}_0, \ldots, {}^{G}\mathbf{w}_N\}$ that specify the location and orientation of the end-effector at discrete intervals.

In three-dimensional space, the position of an object can be described by a location and an orientation. While the location is typically given by a rectangular $\{x, y, z\}$ description, there are many orientation representations used. Three common orientation descriptions used in practice include rotation matrices, Euler angles, and unit quaternions (Mason, 2001). Rotation matrices are a poor choice, since the predicted orientation is an average of observed orientations. In general, the average of two orthonormal matrices is not orthonormal. Other common representations for three-dimensional orientation involve Euler angles. Similar orientations may have extremely different Euler-angle representations, e.g., the Z-Y-X Euler angles $\{\pi, 0, 0\}$ and $\{-\pi, 0, 0\}$ are equivalent. The element-by-element average of these two representations is $\{0, 0, 0\}$, which describes a much different orientation.[2] These discontinuities made Euler angles somewhat unreliable in our experiments. The solution in two dimensions, using sines and cosines to ensure angle continuity, becomes a rotation matrix in the three-dimensional case. For these reasons, we use a unit quaternion to represent orientation. While quaternions also have discontinuous representations and must be of unit length, they have yielded the best performance in our experiments.

For notational convenience, we use a column vector to represent a waypoint, $\boldsymbol{x}_n = \text{vec}({}^{n}\mathbf{w}_{n+1})$. Describing orientation information with unit quaternions means that $\boldsymbol{x}_n$ is a (7×1) vector.

[2] It is typical to restrict angles to the semi-open interval $(-\pi, \pi]$. This example still applies as the value in the second Euler angle gets very close to $-\pi$.

4.2.1 Rotation and Translation Independence

A homogeneous coordinate-frame transform, ${}^a\mathbf{w}_b$, is defined by the matrix

$$ {}^a\mathbf{w}_b \triangleq \left[\begin{array}{c|c} {}^a\boldsymbol{R}_b & {}^a_a\boldsymbol{p}_b \\ \hline \mathbf{0} & 1 \end{array}\right], $$

where ${}^a\boldsymbol{R}_b$ is a (3×3) orthonormal matrix describing the rotation from frame a to frame b and the (3×1) vector ${}^c_a\boldsymbol{p}_b$ specifies the translation from frame a to frame b in coordinate frame c. Many references, e.g., Craig (1989), have derived the following properties from the definitions above:

$$ \begin{aligned} {}^c_a\boldsymbol{p}_b &= -{}^c_b\boldsymbol{p}_a, \\ {}^c_a\boldsymbol{p}_b &= {}^c_a\boldsymbol{p}_d + {}^c_d\boldsymbol{p}_b, \\ {}^c_a\boldsymbol{p}_b &= {}^c\boldsymbol{R}_d\,{}^d_a\boldsymbol{p}_b, \\ {}^a\boldsymbol{R}_b &= {}^a\boldsymbol{R}_c\,{}^c\boldsymbol{R}_b, \\ {}^a\boldsymbol{R}_b &= ({}^b\boldsymbol{R}_a)^{-1} = ({}^b\boldsymbol{R}_a)^{\mathsf{T}}, \\ {}^a\boldsymbol{R}_a &= \boldsymbol{I}. \end{aligned} $$

The product of two homogeneous coordinate transforms, called "compounding," is written

$$ \begin{aligned} {}^a\mathbf{w}_b\,{}^b\mathbf{w}_c &= \left[\begin{array}{c|c} {}^a\boldsymbol{R}_b & {}^a_a\boldsymbol{p}_b \\ \hline \mathbf{0} & 1 \end{array}\right] \left[\begin{array}{c|c} {}^b\boldsymbol{R}_c & {}^b_b\boldsymbol{p}_c \\ \hline \mathbf{0} & 1 \end{array}\right] \\ &= \left[\begin{array}{c|c} {}^a\boldsymbol{R}_b\,{}^b\boldsymbol{R}_c & {}^a\boldsymbol{R}_b\,{}^b_b\boldsymbol{p}_c + {}^a_a\boldsymbol{p}_b \\ \hline \mathbf{0} & 1 \end{array}\right] \\ &= \left[\begin{array}{c|c} {}^a\boldsymbol{R}_c & {}^a_b\boldsymbol{p}_c + {}^a_a\boldsymbol{p}_b \\ \hline \mathbf{0} & 1 \end{array}\right] \\ &= \left[\begin{array}{c|c} {}^a\boldsymbol{R}_c & {}^a_a\boldsymbol{p}_c \\ \hline \mathbf{0} & 1 \end{array}\right] \\ &\doteq {}^a\mathbf{w}_c. \end{aligned} \tag{4.1} $$

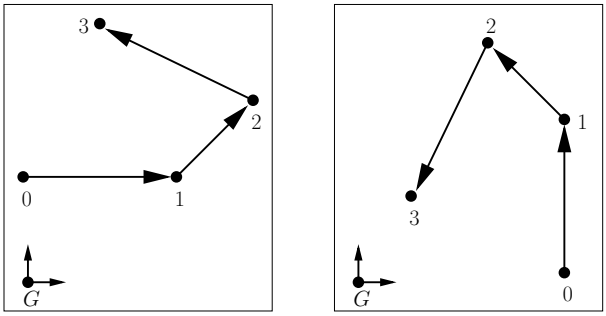


Figure 4.2: Though different in the global reference frame, G, the relative-movement information between these two programs is the same.

Essentially, Equation 4.1 describes the relative change in rotation and translation between the frames a and c. The inverse transform ${}^{b}\mathbf{w}_{a}$, which "undoes" the transform ${}^{a}\mathbf{w}_{b}$, can be derived from the earlier properties

$$
\begin{aligned}
{}^{b}\mathbf{w}_{a} &= \left[\begin{array}{c|c} {}^{b}\boldsymbol{R}_{a} & {}^{b}_{b}\boldsymbol{p}_{a} \\ \hline \mathbf{0} & 1 \end{array}\right] \\
&= \left[\begin{array}{c|c} ({}^{a}\boldsymbol{R}_{b})^{\mathsf{T}} & -{}^{b}_{a}\boldsymbol{p}_{b} \\ \hline \mathbf{0} & 1 \end{array}\right] \\
&= \left[\begin{array}{c|c} ({}^{a}\boldsymbol{R}_{b})^{-1} & -{}^{b}\boldsymbol{R}_{a}\,{}^{a}_{a}\boldsymbol{p}_{b} \\ \hline \mathbf{0} & 1 \end{array}\right] \\
&= \left[\begin{array}{c|c} ({}^{a}\boldsymbol{R}_{b})^{-1} & -({}^{a}\boldsymbol{R}_{b})^{-1}\,{}^{a}_{a}\boldsymbol{p}_{b} \\ \hline \mathbf{0} & 1 \end{array}\right] \\
&= ({}^{a}\mathbf{w}_{b})^{-1}.
\end{aligned}
$$

Note that the inverse transform *always* exists, since the only matrix inversion involves an orthonormal matrix, which must be full rank.

For any PRP system to be useful, it must be able to recognize and predict patterns in a rotation- and translation-independent manner. Without this ability, the PRP will not be able to assist users when they perform the same task in a different area or orientation in the workspace. As we demonstrate shortly, describing a robot program by relative-movement information, instead of absolute-position information, has the distinct advantage of yielding rotation and translation independence. For example, the two robot programs shown in Figure 4.2 are quite different in absolute terms but have the same relative-movement description. Using a relative-waypoint representation, the PRP system is able to recognize patterns *and* predict waypoints independent of rotation and translation. The following definition precisely delineates the concept of the relative-waypoint representation.

Definition 4.1. *Given a sequence of homogeneous coordinate-frame transforms with respect to a common frame, G,*

$$\mathbf{W} = \{{}^{G}\mathbf{w}_0, {}^{G}\mathbf{w}_0, \ldots, {}^{G}\mathbf{w}_N\},$$

and its Sequential Relative Homogeneous Coordinate-frame Transform (*SRHCT*) *is given by*

$$\begin{aligned}\widetilde{\mathbf{W}} &= \{{}^{0}\mathbf{w}_G {}^{G}\mathbf{w}_1, {}^{1}\mathbf{w}_G {}^{G}\mathbf{w}_0, \ldots, {}^{N-1}\mathbf{w}_G {}^{G}\mathbf{w}_N\} \\ &= \{{}^{0}\mathbf{w}_1, {}^{1}\mathbf{w}_2, \ldots, {}^{N-1}\mathbf{w}_N\}.\end{aligned}$$

This definition yields the following intuitive result.

Theorem 4.1. *A robot program specified by its SRHCT is independent of an initial rotation and translation.*

Proof. Suppose we have a robot program with an arbitrary number of waypoints, $\mathbf{W} = \{{}^{G}\mathbf{w}_0, {}^{G}\mathbf{w}_0, \ldots, {}^{G}\mathbf{w}_N\}$. We give the robot program an initial rotation and translation by premultiplying each waypoint by some homogeneous coordinate transform, ${}^{H}\mathbf{w}_G$, to yield

$$\begin{aligned}\mathbf{W}^H &= \{{}^{H}\mathbf{w}_G {}^{G}\mathbf{w}_0, {}^{H}\mathbf{w}_G {}^{G}\mathbf{w}_1, \ldots, {}^{H}\mathbf{w}_G {}^{G}\mathbf{w}_N\} \\ &= \{{}^{H}\mathbf{w}_0, {}^{H}\mathbf{w}_1, \ldots, {}^{H}\mathbf{w}_N\}.\end{aligned}$$

The SRHCT of the rotation and translated program is then

$$\begin{aligned}\widetilde{\mathbf{W}}^H &= \{{}^{0}\mathbf{w}_H {}^{H}\mathbf{w}_1, {}^{1}\mathbf{w}_H {}^{H}\mathbf{w}_2, \ldots, {}^{N-1}\mathbf{w}_H {}^{H}\mathbf{w}_N\} \\ &= \{{}^{0}\mathbf{w}_1, {}^{1}\mathbf{w}_2, \ldots, {}^{N-1}\mathbf{w}_N\}.\end{aligned}$$

Note that the SRHCT of the original robot program is, by definition,

$$\begin{aligned}\widetilde{\mathbf{W}} &= \{{}^{0}\mathbf{w}_G {}^{G}\mathbf{w}_1, {}^{1}\mathbf{w}_G {}^{G}\mathbf{w}_2, \ldots, {}^{N-1}\mathbf{w}_G {}^{G}\mathbf{w}_N\} \\ &= \{{}^{0}\mathbf{w}_1, {}^{1}\mathbf{w}_2, \ldots, {}^{N-1}\mathbf{w}_N\}.\end{aligned}$$

Since $\widetilde{\mathbf{W}}^H \equiv \widetilde{\mathbf{W}}$, a robot program specified by its SRHCT is independent of an initial arbitrary rotation and translation. □

Theorem 4.1 implies that that robot programs can be *recognized* independent of an initial rotation and translation. The ith SRHCT robot program is written as $\boldsymbol{X}^i = \{\boldsymbol{x}_0^i, \boldsymbol{x}_1^i, \ldots, \boldsymbol{x}_{N_i}^i\}$. The input to algorithm `Learn-HMM` (Figure 3.10) is the multiset of SRHCT robot programs $\boldsymbol{X} = \{\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots, \boldsymbol{X}^M\}$. Note that there is no

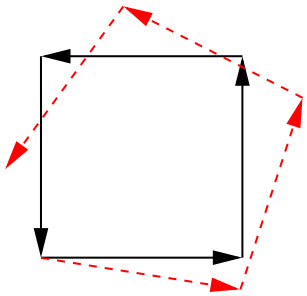


Figure 4.3: An error of 9° per rotation results in a compounded error of almost 50% of the leg length.

requirement that the robot programs have a common length or even describe the same physical task. The algorithm could assimilate tasks corresponding to, e.g., arc-welding a bed frame, spot welding a ship hull, painting a car, etc. Once the robot programs have been assimilated into a CDHMM and the user begins creating a new robot program, we compute predictions of the next waypoint, $\widehat{\boldsymbol{x}}_n^*$, based on Equation 3.9. However, $\widehat{\boldsymbol{x}}_n^*$ is a stacked column vector of the relative waypoint ${}^{n-1}\widehat{\mathbf{w}}_n$, which specifies the estimated relative movement of the end-effector from its current position, ${}^{G}\mathbf{w}_{n-1}$. Therefore, the estimated position of the next waypoint is found by postmultiplying the estimator, ${}^{n-1}\widehat{\mathbf{w}}_n$, by the current position of the end-effector in the global frame

$$ {}^{G}\widehat{\mathbf{w}}_n \quad = \quad {}^{G}\mathbf{w}_{n-1}\,{}^{n-1}\widehat{\mathbf{w}}_n. $$

The relative-movement description requires that the user specify the orientation of the end-effector very precisely. If the user repeats a task with a slight difference in orientation, then the Cartesian errors occurring between the two demonstrations can quickly accumulate and become large, similar to the compounding of dead-reckoning errors in a mobile robot (Thrun, 1998), as in Figure 4.3. But the compounding of errors in Figure 4.3 is certainly a worst-case scenario because, in the PRP domain, a human is controlling the robot and "closes the loop" to nullify much of the residual prediction error. However, the ability to compute rotation- and translation-independent predictions is critical to the success of the PRP system, since the similarities embedded in a robot program occur at different orientations and locations in the workspace, as in Figure 4.1.

4.2.2 Scale Invariance

If the PRP system has assimilated a robot program moving along a rectangle, it will be of little or no help in predicting the waypoints of different-sized rectangles, even rectangles with the same aspect ratio. The ability to recognize and predict scaled versions of previously assimilated tasks is called *scale invariance*. As mentioned earlier, the PRP system is able to recognize patterns independent of their location and orientation by using rotation- and translation-independent features, the SRHCT. While the same approach would work for *recognizing* scaled

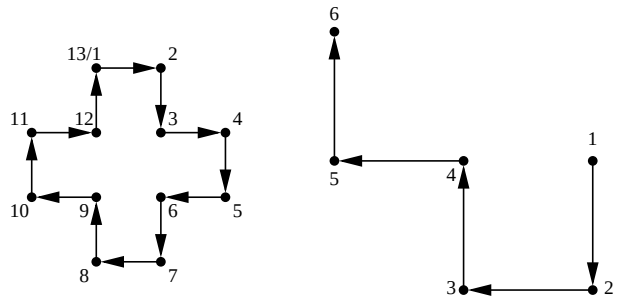


*Figure 4.4: The waypoints (*1–6*) from the pattern on the right form a scaled subtask ($\psi = 2$) of waypoints (*6-11*) from the pattern on the left.*

tasks, no feature transform appears feasible for *predicting* the waypoints of scaled tasks. For instance, we could reduce a robot program to a sequence of rays. To recognize a task based on previously assimilated tasks, we could compute the likelihood of the various rays. However, these half-infinite lines are of little help when trying to predict the position of the next waypoint, as they yield only the direction of the prediction. We considered a few scale-invariant feature transforms but none yielded the ability to predict reliably. Thus, we have turned to a more brute-force approach: iterative locally optimal maximum likelihood.

In this formulation, we want to find the value, $\psi \in (0, \infty)$, that scales waypoints in the current task, $\boldsymbol{X}^c_{0:n} = \{\boldsymbol{x}^c_0, \ldots, \boldsymbol{x}^c_n\}$, such that the likelihood of the scaled task is maximized given the estimated CDHMM,

$$\begin{aligned} \widehat{\psi}^* &= \arg\max_{\psi} \mathrm{p}(\boldsymbol{x}^c_0, \ldots, \boldsymbol{x}^c_n | \psi, \boldsymbol{\lambda}) \\ &= \arg\max_{\psi} \sum_{q_j \in \mathcal{Q}} \mathrm{p}(c_n{=}q_j, \boldsymbol{X}^c_{0:n} | \psi, \boldsymbol{\lambda}) \\ &\doteq \arg\max_{\psi} \sum_{q_j \in \mathcal{Q}} \alpha_n(j, \psi). \end{aligned} \tag{4.2}$$

In the PRP domain, we do not want to multiply each element of a stacked waypoint, $\boldsymbol{x}_n$, by the scale factor, ψ. Typically, we would like to scale only the Cartesian coordinates of the waypoint and leave the orientation components alone. If the CDHMM observation pdf is Gaussian, this implies that the covariance matrix, $\boldsymbol{\Sigma}$, is symmetric block-diagonal, with one block corresponding to the Cartesian coordinates and the other block corresponding to the orientation information. In real-world terms this arrises from the assumption that users make Cartesian errors

independent of orientation errors. With this in mind, we define the "scale matrix" such that

$$\begin{aligned} \boldsymbol{\Psi}_{i,j} &\triangleq \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j \text{ and not scaled} \\ \psi, & \text{if } i = j \text{ and scaled} \end{cases} \\ \frac{\partial}{\partial \psi}\boldsymbol{\Psi}_{i,j} &= \begin{cases} 0, & \text{if } i \neq j \\ 0, & \text{if } i = j \text{ and not scaled} \\ 1, & \text{if } i = j \text{ and scaled} \end{cases} \\ &\dot{=} \boldsymbol{\Psi}'_{i,j}. \end{aligned} \tag{4.3}$$

Then a scaled waypoint is found by premultiplying the stacked waypoint by the scale matrix, $\boldsymbol{\Psi}\boldsymbol{x}_n$, which is a column vector with the Cartesian elements scaled by ψ and the orientation elements unscaled. Similar to Equation 3.7, we define the scaled forward variables as

$$\alpha_n(j,\psi) \triangleq \begin{cases} b_j(\boldsymbol{\Psi}\boldsymbol{x}_n) \sum\limits_{q_i \in \mathcal{Q}} a_{j|i}\alpha_{n-1}(i,\psi), & n > 0 \\ b_j(\boldsymbol{\Psi}\boldsymbol{x}_0)\pi_j, & n = 0 \end{cases}. \tag{4.4}$$

The complete derivation for the following equation is given in Section A.2.3. Assuming the observation pdfs are Gaussian, the partial derivative of Equation 4.4 with respect to the scale factor is

$$\frac{\partial}{\partial \psi}\alpha_n(j,\psi) = \left\{(\boldsymbol{\Psi}'\boldsymbol{x}_n)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j - \boldsymbol{\Psi}\boldsymbol{x}_n)\right\}\alpha_n(j,\psi) + b_{j,\psi}(\boldsymbol{x}_n)\sum_{q_i \in \mathcal{Q}} a_{j|i}\frac{\partial}{\partial \psi}\alpha_{n-1}(i,\psi). \tag{4.5}$$

Since Equation 4.5 is recursive, the appropriate initial conditions at the beginning of time are needed (the partial derivative of Equation 4.4). We cannot solve for the optimal ML estimator of the scale factor explicitly, since Equation 4.5 is a sum of transcendental terms. We can only hope to find locally optimal ML solutions by iterative line-search techniques, such as cubic interpolation, gradient ascent, etc. (Bertsekas, 1995).

Once we find the (locally optimal) ML estimator of the scale, $\widehat{\psi}^*$, we compute predictions based on the scaled task by substituting Equation 4.4 into the ML prediction, Equation 3.9. The scaled task is computed by premultiplying each observation by the ML scale matrix, $\boldsymbol{\Psi}$, from Equation 4.3,

$$\boldsymbol{X}^{i,\widehat{\psi}^*} = \{\boldsymbol{\Psi}\boldsymbol{x}_0^i, \boldsymbol{\Psi}\boldsymbol{x}_1^i, \ldots, \boldsymbol{\Psi}\boldsymbol{x}_{N_i}^i\}. \tag{4.6}$$

However, even if we are able to identify scaled (sub)tasks based on having a CDHMM, we must still incorporate scaled tasks into a CDHMM. When we execute the `Learn-HMM` algorithm, this leads to a "chicken and egg"

Algorithm `Learn-Scaled-HMM`

$\boldsymbol{X} = \{\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots, \boldsymbol{X}^M\}$ is the multiset of all tasks.
$\epsilon \geq 0$ is a similarity threshold.
$\psi_{\min}$ is the minimum bracket for the scale.
$\psi_{\max} > \psi_{\min}$ is the maximum bracket for the scale.

```
1: V := ∅, E := ∅
2: G_X := (V, V^0, E, X, V, f, g)
3: for all X^i ∈ {X^0, X^1, ..., X^M}
4:    ψ̂* := compute-scale (G_X, X^i, ψ_min, ψ_max) (Figure 4.6)
5:    X^{i,ψ̂*} from Equation 4.6
6:    G_X := assimilate-task (G_X, X^{i,ψ̂*}, ε) (Figure 3.8)
7: end for all
8: λ := graph-to-HMM (G_X, M) (Figure 3.9)
```

Figure 4.5: The algorithm for assimilating scaled tasks into an HMM.

problem: without a CDHMM, we cannot determine the ML scale of a task; without any tasks, we cannot create a CDHMM. In Figure 4.5, we give an algorithm that uses the tasks already assimilated into the CDHMM to estimate the ML scale. We then scale the task according to Equation 4.6 and assimilate this scaled task into the CDHMM. However, the estimate is unreliable if a scaled version of the task has never been assimilated before. In this case, we assimilate the task unscaled. The bracketing of the scale can arise from the physical nature of manipulator robots. All robots have finite accuracy and reachability, and these numbers can serve as a bracket on the ML scale.

4.3 Offline Programming

In this section, we analyze the performance of the PRP system on five programs created using a conventional offline-programming environment (Figure 4.7). The programs have between 252 and 1899 waypoints with 16 to 196 subroutines. Collectively, these programs took over 70 work days to complete by a professional robot programmers. The programs were created to automate arc-welding production at several factories in Sweden. Each program was designed to produce a different type of product, from round tables to bed frames. Since the robot programs were developed independently of our PRP system, the behavior of the users was unmodified by these experiments. The programs provide the substrate to verify experimentally the ability of PRP to generate accurate predictions of users creating waypoints for complex, real-world tasks. In Section 4.3.2, we analyze the prediction accuracy of the PRP system as the complexity of the estimated CDHMM changes, by varying the parameter δ, and show that the system is capable of performing real-time operation. In Section 4.3.3, we show that prediction confidence, ϕ_n, strongly correlates with prediction accuracy. In Section 4.3.4, we test the hypothesis

Function `compute-scale`
$G_{\boldsymbol{\mathcal{X}}} = (V, V^0, E, \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{V}}, f, g)$ is the observation graph.
$\boldsymbol{X}^i = \{\boldsymbol{x}_0^i, \boldsymbol{x}_1^i, \ldots, \boldsymbol{x}_{N_i}^i\}$ is the task to assimilate.
$\psi_{\min}$ is the minimum bracket for the scale.
$\psi_{\max} > \psi_{\min}$ is the maximum bracket for the scale.

```
 1: N := maximum depth of G_X
 2: if N_i ≤ N then
 3:    λ := graph-to-HMM (G_X, i) (Figure 3.9)
 4:    ψ := arg max_ψ p(X^i | ψ, λ)
 5:    if ψ_min ≤ ψ ≤ ψ_max then
 6:       ψ̂* := ψ
 7:    end if
 8:    else then
 9:       /* scale is outside bracket: assume 1.0 */
10:       ψ̂* := 1
11:    end else
12: end if
13: else then
14:    /* X^i too long for G */
15:    ψ̂* := 1
16: end else
17: return ψ̂*
```

Figure 4.6: Computing the ML scale for a task.

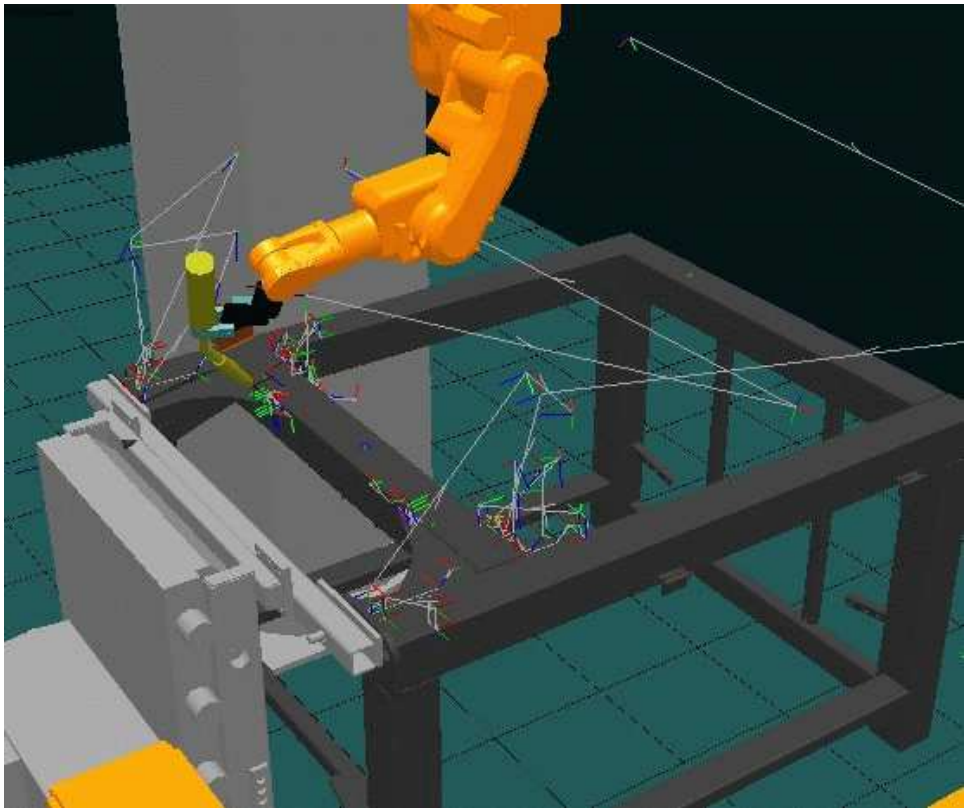


Figure 4.7: Screen shot from the offline package showing the workspace and waypoints of a program with 18 *subroutines.*

that users are well modeled by a CDHMM by observing the prediction accuracy of the PRP system as more waypoints are incorporated into the learning algorithm.

4.3.1 Methodology

To compute waypoint predictions, we segment the programs along the subroutines contained in each program. We emulate users creating the robot program by feeding the subroutine waypoints into the PRP system in a serial fashion. The PRP system computes a prediction of the next position by conditioning the CDHMM on previous waypoints from the subroutine (Equation 3.9). If the PRP system has sufficient confidence in the prediction (Equation 3.10), then we consider the prediction valid and compute its error. To determine the prediction error, we compare the predicted waypoint to the next waypoint in the subroutine and consider any difference to be an error in the prediction. At the end of each subroutine we incorporate its waypoints into the CDHMM, building an estimated user model. After predicting the final waypoint in a program, we reinitialize the CDHMM *tabula rasa*

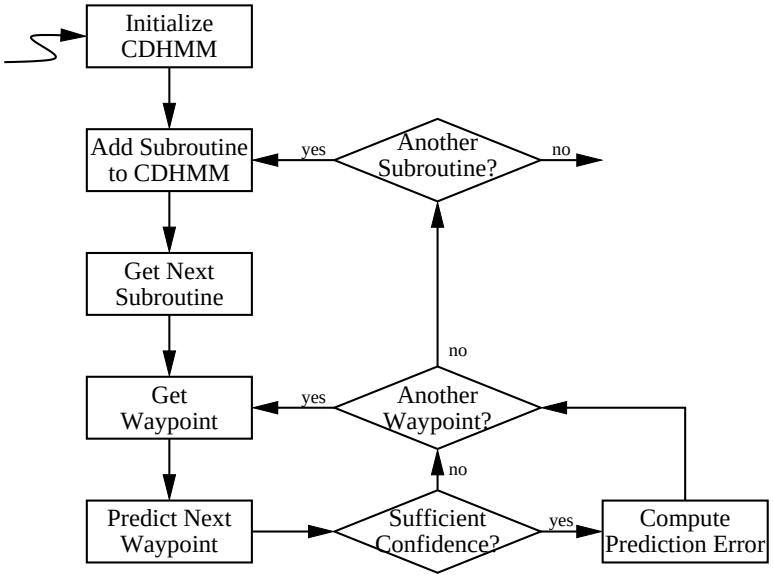


Figure 4.8: Flow chart for computing predictions for the offline programs.

and start the prediction of the next program. A flow chart of this procedure is given in Figure 4.8.

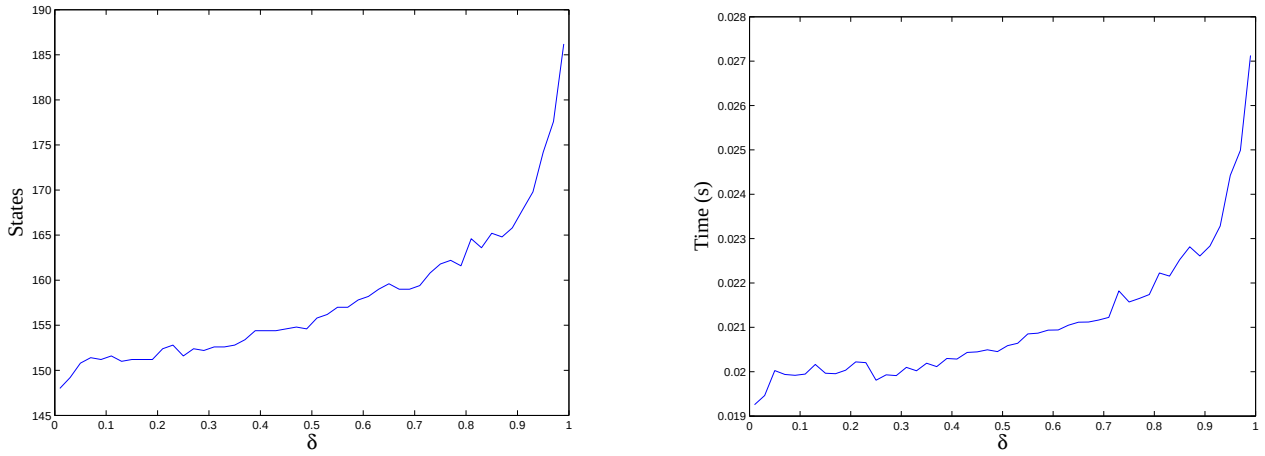
The definition of similarity, Equation 3.1, incorporates a symmetric PD precision matrix, $\boldsymbol{C}$, that represents the inverse covariance matrix of human error when defining a waypoint. We computed the covariance matrix by having users repeatedly move a robot to a point in space and analyzing the residual error.[3] Robotic arc-welding requires approximately 1 millimeter of Cartesian accuracy and about 0.1 radians of angular accuracy. The results presented in this work employ these physical tolerance constraints as a threshold for determining a "useful prediction," meaning that a waypoint within these tolerances will generally require no fine-tuning by the user.

At the end of welding, it is common to execute a gross repositioning where the robot moves across the workspace to the next weld, typically on the order of a meter. These movements tend not to be predictable and a small percentage error in predicting the gross repositioning will dominate the mean over many precise movements. We are more interested in the typical prediction error, which is better conveyed by the median.

4.3.2 Performance as a Function of Model Complexity

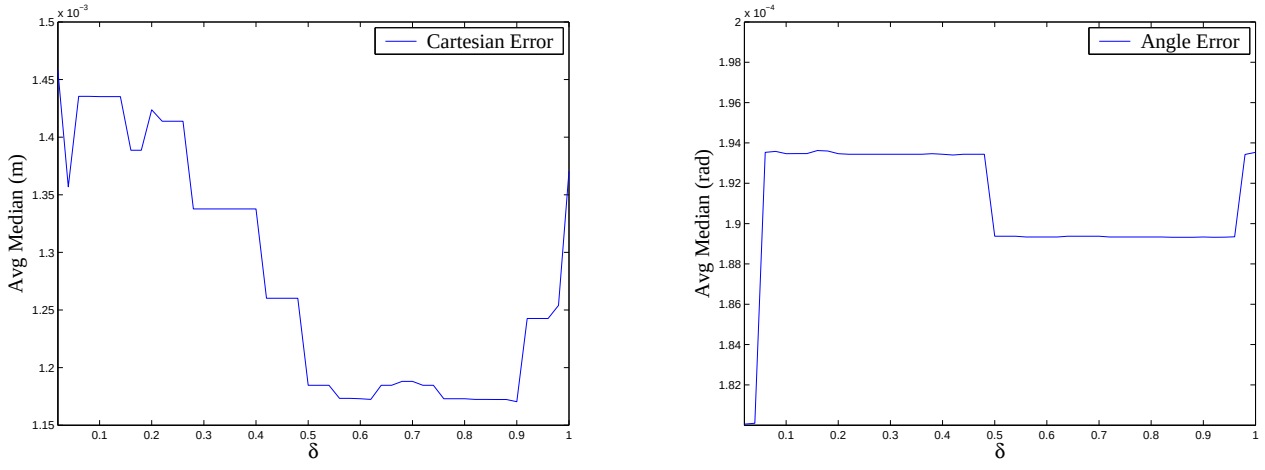
In Figure 4.9 we plot two measures of model complexity, CDHMM states and running time. As expected, the number of CDHMM states tends to increase as δ increases. Also, running time increases as the number of CDHMM

[3]The covariance matrix was computed from online-programming experiments, whereas this section deals with offline-programming. From our experience, the principal directions of variance appear to be similar, but the eigenvalues are smaller when using an offline environment.



(a) (b)

Figure 4.9: Model complexity as a function of δ.



(a) (b)

Figure 4.10: Median Cartesian error and angle error as a function of δ.

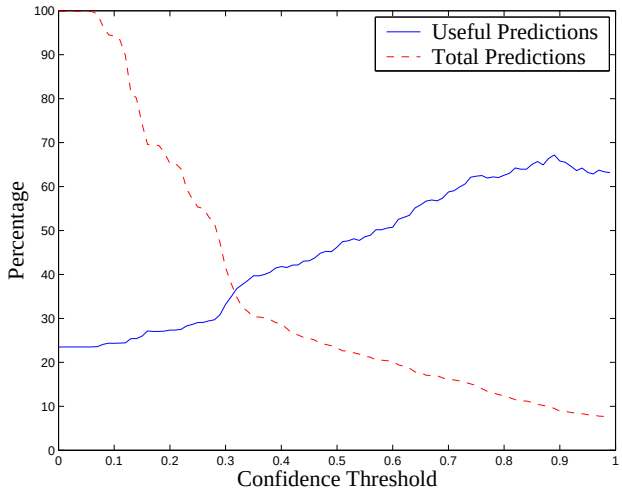
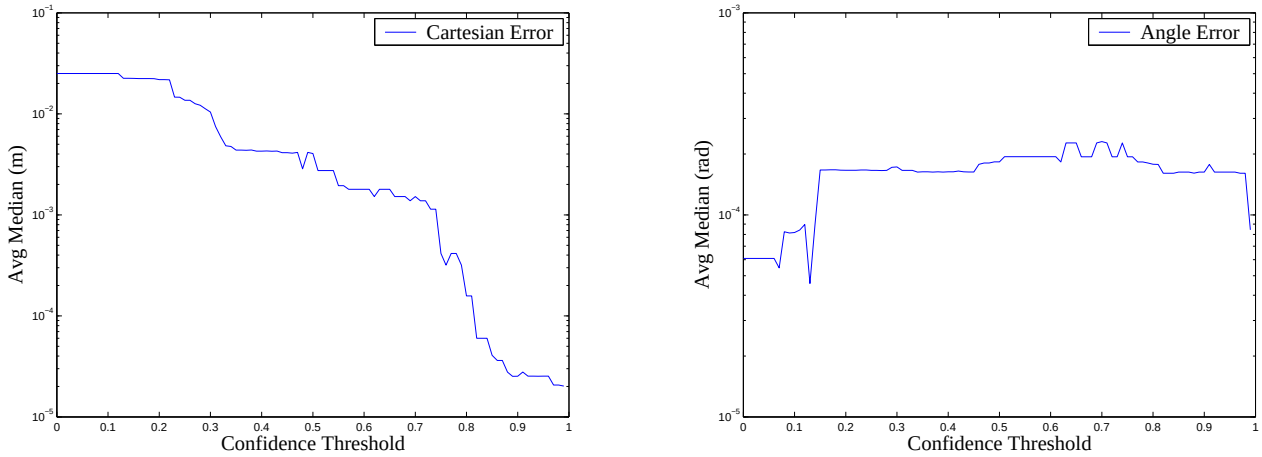


Figure 4.11: Percentage of predictions and useful predictions as a function of the confidence threshold.

states increases. Importantly, Figure 4.9 shows the average time per waypoint taken to estimate a CDHMM *and* compute a prediction. This time, ~ 25 milliseconds, is suited for real-time use in a robot-programming environment.

To determine the performance of the system as a function of model complexity, we held out one program and computed the performance on the remaining four programs. The held-out program is meant as a sort of test set, but since data are so sparse it is difficult to draw any statistically significant conclusions based on this four-program training set and one-program test set. Indirectly, the parameter δ also determines the accuracy of predictions by controlling the complexity of the CDHMM. In Figure 4.10, we plot the average median error on the four-program training set as a function of δ. For all values of δ, the angle error is extremely small, less than 0.2 milliradians or about 0.3% of the angle changed during prediction. This is because robots tend to change orientation in a fairly predictable manner during welding and gross repositioning. By exploiting this information, the PRP system can generate extremely accurate angle predictions. However, the Cartesian movements of the robot are determined by the size of the objects in the workspace, a much more unpredictable quantity. When the parameter is too small, the learning algorithm produces a CDHMM that is too simple to represent users. When the parameter is too large, the CDHMM begins to overfit the data. In PRP, as in many machine-learning applications, "everything should be made as simple as possible, but not simpler," and the *correct* complexity depends on the tasks at hand. In our case, the Cartesian error bottoms out on the interval $\delta \in (0.5, 0.9)$. This implies that a value in the neighborhood of $\delta \approx 0.7$ induces the right amount of complexity on the training set.

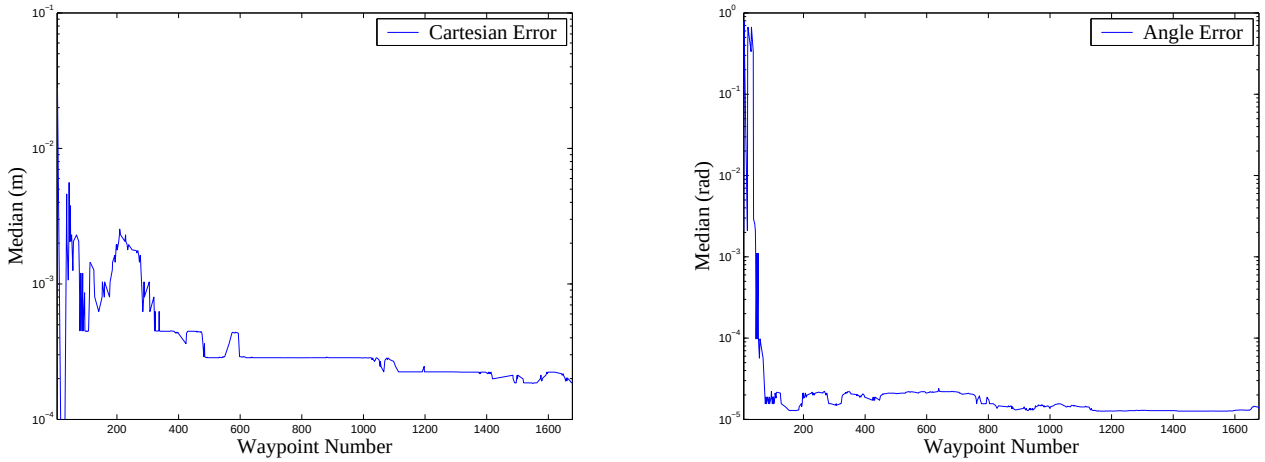


(a) (b)

Figure 4.12: Median Cartesian error and angle error as a function of the confidence threshold.

4.3.3 Performance as a Function of Confi dence

In this section we determine the correlation of prediction confidence to prediction accuracy. Specifically, we discard predictions with confidence below a threshold and analyze the accuracy of the remaining predictions. The hypothesis is that higher-confidence predictions are more accurate than lower-confidence predictions. Inevitably, as the confidence threshold increases there will be accurate predictions discarded due to insufficient confidence. Loosely speaking, the PRP system may have a "lucky guess"; without knowing the target waypoint at the time of prediction, it is impossible to discriminate between lucky guesses and well-informed estimates. Using the training set, we can determine a confidence threshold that strikes a balance between quality and quantity. We fix the value of $\delta = 0.7$ and have the learning algorithm construct a CDHMM based on Figure 4.8 for each program in the four-program training set and analyze the performance of the PRP system as a function of confidence threshold. In Figure 4.11 we plot the percentage of useful predictions as a function of the confidence threshold on the four-program training set. The percentage of useful predictions generally increases as the confidence threshold increases until about $\phi_n \geq 0.8$, when the percentage plateaus. In Figure 4.12 we plot the average median error as a function of the confidence threshold. For all confidence thresholds the angle error is extremely small, between 0.06 and 0.2 milliradians. The median angle movement during prediction was about 65 milliradians, meaning that the angle prediction error is between 0.09% and 0.3% of the angle change. However, the Cartesian prediction error improves as the confidence threshold increases, until it bottoms out at around 0.03 millimeters for $\phi_n \geq 0.8$, well below the physical tolerance required by arc-welding. The median Cartesian movement during prediction was



(a) (b)

Figure 4.13: Median error as a function of the number of waypoints assimilated into the CDHMM.

about 100 millimeters, meaning that the Cartesian prediction error is about 0.03% of the movement of the robot during a prediction. Empirically, based on this data set, confidence is related to Cartesian error with a normalized correlation coefficient of $\rho = -0.89$, significant to a p-value of $p \ll 0.01$. This strong correlation implies that, on the training set, higher-confidence predictions tend to be more accurate.

4.3.4 Temporal Performance

In this subsection we use parameters that performed well on the training set, $\delta = 0.7$ and $\phi_n \geq 0.7$, to analyze the performance on the hold-out test program. The CDHMM is initialized *tabula rasa* and incorporates waypoints from the test program incrementally according to the procedure in Figure 4.8. In Figure 4.13(a), we plot the median prediction error on the hold-out program as the user creates waypoints. After about 400 waypoints have been incorporated, the median Cartesian error stabilizes at around 0.25 millimeters. The median Cartesian movement during prediction was 50 millimeters, meaning that the Cartesian prediction error is about 0.5% of the movement of the robot during a prediction. Likewise, the median angle error stabilizes around 0.02 milliradians after about 150 waypoints. The median angle movement during prediction was 0.1 milliradians, meaning that the angle prediction error is about 20% of the angle change. Both of these values are well below the physical tolerance of 1 millimeter and 0.1 radians required by arc-welding. This asymptotic-like behavior of the PRP prediction error

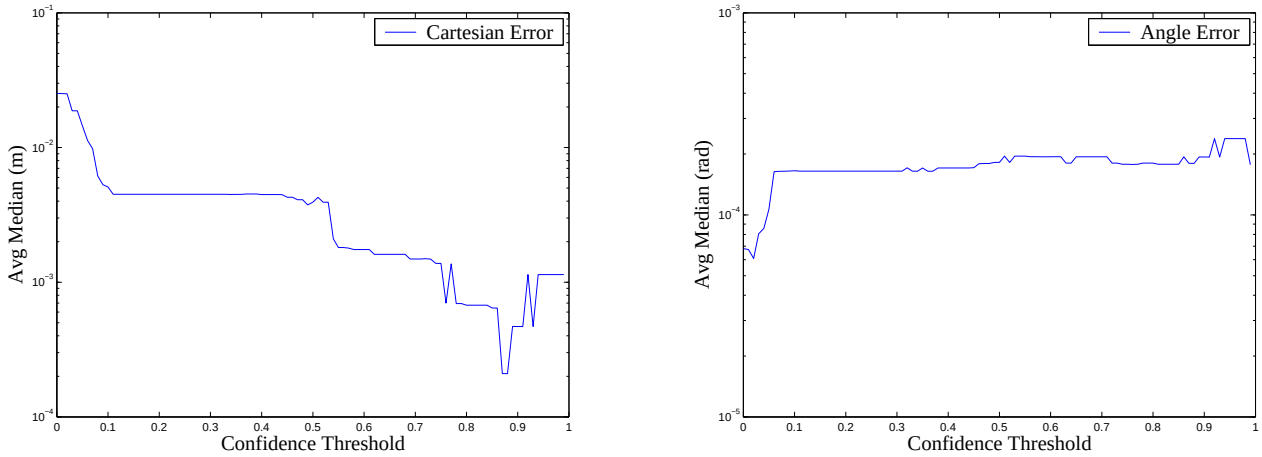
suggests that the estimate of users improves until sufficient information about the task has been acquired.[4] The PRP algorithm computed predictions with confidence $\phi_n \geq 0.7$ for about 25% of the waypoints on the hold-out program, which originally took over eight work weeks to create by a professional robot programmer using a conventional offline-programming package. Allowing the PRP system to move the robot to predicted waypoints automatically could save several days in programming time. It it important to remember that none of the programs, either in the training set or the hold-out, were created with a PRP system in mind. In other words, the programs were not created with any motivation for reusing previous work. The predictability of the waypoints results from the inherent similarity of the underlying tasks. It seems plausible that a programmer creating waypoints with a PRP system would be more likely to create waypoints in a predictable fashion, resulting in greater time savings, making robot-programming environments incorporating a PRP system even less cumbersome.

4.3.5 Discussion of Results

The results contained in this section validate the core principles of PRP. For the different programs analyzed, the PRP system computed predictions with confidence $\phi_n \geq 0.7$ for between 10% and 30% of possible waypoints. The median Cartesian prediction error for each program was between 0.03 millimeters and 0.25 millimeters, or 0.03% and 0.5% of the distance moved during prediction. The median angle prediction error for each program was between 0.06 milliradians and 0.2 milliradians, or 0.09% and 20% of the angle change during prediction. All of these values are well under the useful physical tolerances required by arc-welding of 1 millimeter of Cartesian accuracy and 0.1 radians of angular accuracy. On average, about 60% of predictions computed by the PRP system with a confidence of $\phi_n \geq 0.7$ were useful. The high accuracy of predictions indicates that the CDHMM can identify the inherent similarity contained in complex, real-world robot programs. Since the programs were created without consideration for PRP, the predictability of the waypoints is due to the underlying similarity of the tasks themselves. The PRP system was able to estimate a user model and compute predictions in about 25 milliseconds, which is sufficient for real-time operation. The asymptotically decreasing nature of prediction error, as a function of the number of waypoints incorporated into the learning algorithm, shows that the modeling of a robot programmer by a CDHMM is appropriate. The strong correlation between prediction accuracy and prediction confidence gives the PRP system a causal statistic for determining which predictions may be inaccurate. By "filtering" predictions based on confidence, the PRP system can avoid burdening the user with unhelpful suggestions.

Since the programs were created independently of PRP, we have no way of directly translating these results to programming-time savings. However, we can estimate hypothetical savings in programming time based on the results by assigning a time benefit for useful predictions and a time penalty for non-useful predictions. A reasonable benefit for a useful prediction is roughly 90%, since it may take the user a short amount of time to

[4] Parenthetically, all programs in the training set also showed the same general asymptotically decreasing shape in their error plots.



(a) (b)

Figure 4.14: Median Cartesian error and angle error as a function of the confidence threshold for the MAP structure-estimation algorithm.

determine if the predicted waypoint is sufficiently close to the desired position. A penalty of 10% for a non-useful prediction seems appropriate, since a prediction can be rejected out of hand by pressing an "undo" button. Using typical values for the PRP system parameters, $\delta = 0.7$ and $\phi_n \geq 0.7$, would result in a 10% reduction in programming time, which translates to 7 work days saved on the programs analyzed in this section. With a benefit of 75% and a penalty of 25%, the programming-time reduction is 7%. However, some waypoints, such as approach points, do not require much accuracy. Also, it seems likely that many non-useful predictions will also benefit the user; a prediction that is close to the intended waypoint, but still requires some fine-tuning, probably takes less time than the user moving the robot manually. Consequently, the binary classification of predictions as useful and non-useful probably represents a lower bound on programming-time savings. In Section 4.5.3 we present results on reducing programming time in laboratory experiments.

4.4 Empirical Comparisons

In this section, we compare the performance of our system to existing methods.

4.4.1 Comparison to Maximum *A Posteriori* Structure Estimation

In this subsection, we compare our learning algorithm to the MAP structure-estimation algorithm by Stolcke and Omohundro (1994b), derived in Section 3.9.[5] In Figure 4.14 we plot the average median error as a function of the confidence threshold for the MAP structure-estimation algorithm. The average median Cartesian error is almost two orders of magnitude larger than for similar confidence thresholds compared to our learning algorithm, about 1.5 millimeters compared to about 0.03 millimeters, respectively. On average, the MAP algorithm created CDHMMs with about 249 states, whereas our algorithm created CDHMMs with an average of 199 states for $\delta = 0.7$.[6] This is due to the term in the MAP merging criterion that penalizes the algorithm for merging states that would increase CDHMM perplexity, cf. Theorem 3.9. CDHMM perplexity is defined as the mean number of outbound transitions from states in the model (Rabiner & Juang, 1993). Since our algorithm does not consider CDHMM perplexity, it able to merge more states. Not surprisingly, the CDHMMs estimated from our learning algorithm have substantially higher perplexity than CDHMMs from the MAP structure-estimation algorithm. Specifically, our algorithm estimated CDHMMs with an average perplexity of 1.54, whereas the MAP algorithm estimated CDHMMs with an average perplexity of 1.13, giving our CDHMMs about 36% higher average perplexity. For the short-sequence similarities, such as those found in PRP (Figure 4.1), this increased perplexity does not appear to hurt performance. However, Stolcke and Omohundro (1994b) developed their MAP structure-estimation algorithm for speech-recognition applications, where long similarity sequences are expected. Their goal was to create a learning algorithm that kept model perplexity as low as possible to recognize the long speech sequences.

4.4.2 Comparison to Prediction-State Probability

We compare the performance of prediction confidence to prediction-state confidence, which is defined as the probability of the most-likely state given the current task, $\max_j \nu_n(j)$, cf. Equation 3.8. In Figure 4.15 we plot the percentage of useful predictions as a function of the prediction-state probability on the four-program training set. In Figure 4.16 we plot the average median error as a function of the prediction-state probability. Empirically, based on this data set, prediction-state probability is related to Cartesian error with a normalized correlation coefficient of $\rho = -0.82$, significant to a p-value of $p \ll 0.01$. While this prediction-state probability correlates well with prediction accuracy, it is not as strong as that of prediction confidence, where the normalized correlation coefficient was $\rho = -0.89$. However, the correlation of prediction-state probability with Cartesian error is limited to a small range of prediction-state probability, between about 0.3 and 0.5, which significantly limits its usefulness. The correlation for values of prediction-state probability greater than 0.5 is $\rho = -0.45$, whereas prediction confidence

[5] Comparisons are based on a reversed-engineered implementation of the algorithm by Stolcke and Omohundro (1994b) that we created.

[6] The total time taken to run the entire battery of experiments using our algorithm was 29.186 seconds, while the MAP structure-estimation algorithm took 542.066 seconds. This is hardly a fair comparison, since we have clearly spent more time optimizing our algorithm than a reverse-engineered implementation based on the pseudo-code from other researchers.

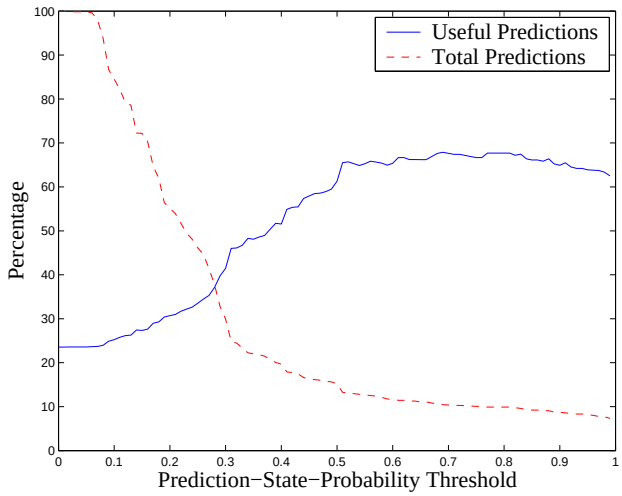
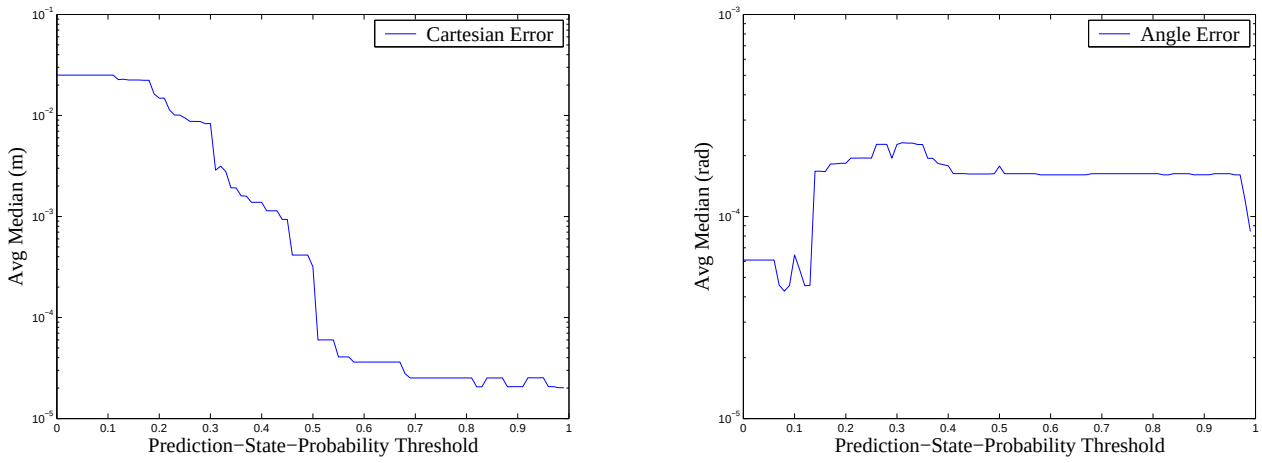


Figure 4.15: Percentage of predictions and useful predictions as a function of the prediction-state-probability threshold.



(a) (b)

Figure 4.16: Median Cartesian error and angle error as a function of the prediction-state-probability threshold.

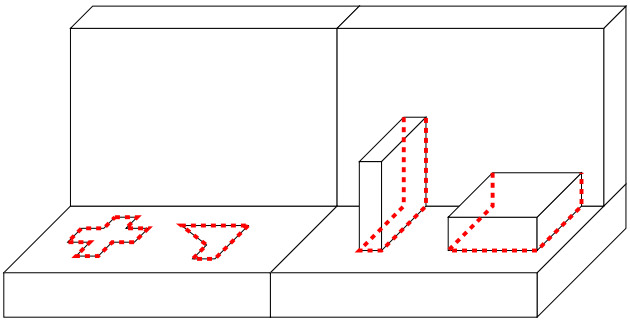


Figure 4.17: The four patterns for user demonstrations with 13, 6, 10, *and* 10 *waypoints respectively.*

has a correlation of $\rho = -0.92$ over the same interval. For high values of both statistics, the Cartesian prediction error is about the same, but prediction confidence is more gradual way to filter out the less accurate predictions.

4.5 Online Programming

In this section we analyze the performance of the PRP system in an online-programming environment. We collected 44 robot programs from 3 users in a laboratory setting. All users had previous experience with online robot programming and were allowed to practice moving the robot until they felt comfortable controlling it. In our terminology, the *repertoire* of the user consists of the four simple tasks shown in Figure 4.17. The two right-most programs, comprising 10 waypoints each, represent standard arc-welding tasks. The two left-most programs, comprising 13 and 6 waypoints respectively, are planar geometric movements. The mean distance between waypoints in these programs was 186 millimeters. These programs were created in a laboratory setting specifically to test the viability of PRP as a robot-programming tool. In Section 4.5.2 we analyze the performance of the PRP system when the learning algorithm incorporates programs in different orders. In Section 4.5.3 we show that the PRP system contributes to a significant reduction in programming time.

4.5.1 Methodology

To create a program, users move the robot with a joystick (Figure 4.18). When users feel that the end-effector is sufficiently close to the desired waypoint, they press a button on the teach pendant to indicate that the current robot position should be stored as the next waypoint in the program. We left the definition of "sufficiently close" to the users, which resulted in substantially larger deviations than required for a program to be viable for arc-welding.

To compute the precision matrix from Equation 3.1, we asked users to move the end-effector repeatedly to an unreferenced point in space by controlling the robot with the joystick. We then computed the residual error and

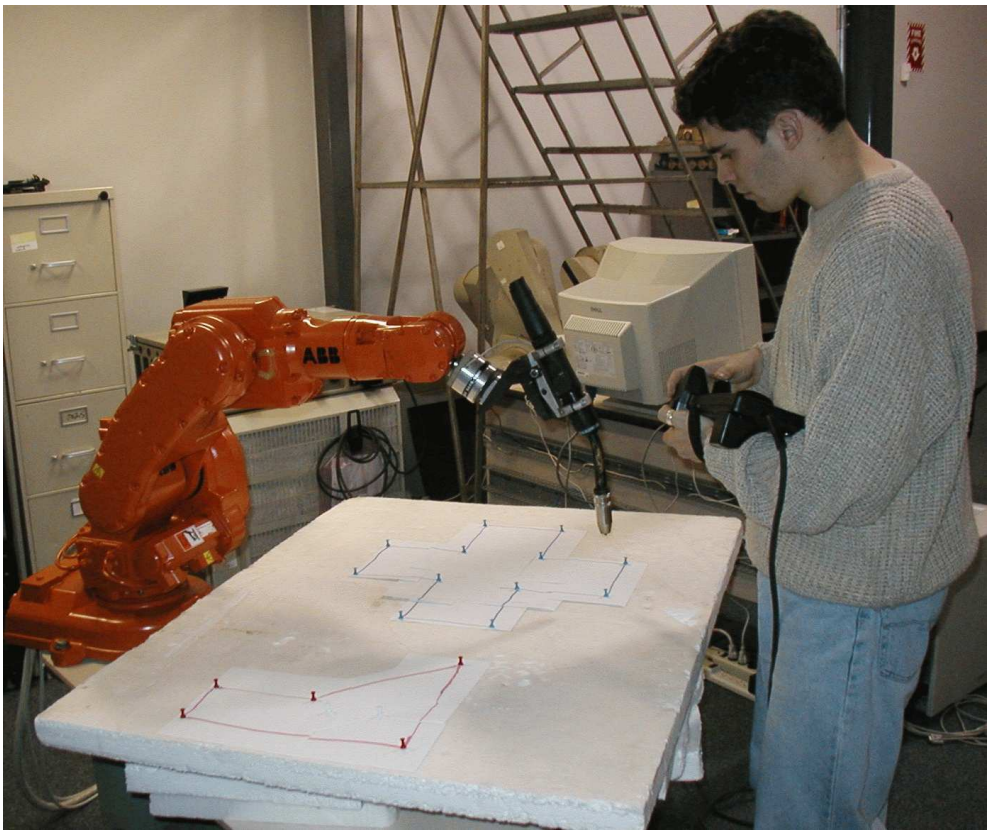


Figure 4.18: Creating waypoints for the patterns in Figure 4.17 with an ABB IRB140.

set the precision matrix equal to the inverse of the covariance, $\boldsymbol{C} = \mathrm{Var}(\boldsymbol{x})^{-1}$. Similarity in robot programs may occur in short subsequences. In these cases, conditioning the CDHMM on waypoints from the entire program may cause the PRP system to generate poor predictions (Equation 3.9), even if we discard predictions below a confidence threshold (Equation 3.10). This can be avoided by computing predictions based on a *horizon*, h, of recent waypoints,

$$\widehat{\boldsymbol{x}}_n^* \quad = \quad \arg\max_{\boldsymbol{x}_n} \mathrm{p}\big(\boldsymbol{x}_n | \boldsymbol{X}^c_{n-h:n-1}, \boldsymbol{\lambda}\big) \,.$$

The experiments in this section will explicitly incorporate this notion of a horizon.

4.5.2 Leave-One-Out Performance

As mentioned previously, our CDHMM structure-estimation learning algorithm is sensitive to the order in which robot programs are incorporated into the model. To analyze this effect, we created 44 permutations of the 44 robot

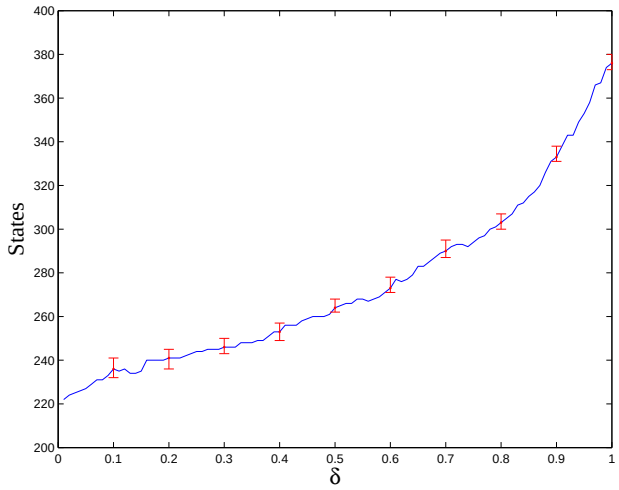


Figure 4.19: Median number of states as a function of δ, over the 44 different orderings of the online robot programs collected from Figure 4.17. The error bars indicate the minimum and maximum states induced by the permutations.

programs from Figure 4.17 so that their waypoints were introduced into the learning algorithm in a differing order. In Figure 4.19 we plot the median number of states as a function of δ. The error bars in Figure 4.19 show the minimum and maximum number of states caused by the different permutations. Figure 4.19 implies that varying the parameter δ has a much greater impact on model complexity than does the order in which the robot programs are introduced into the learning algorithm.

Using the 44 permutations, we also computed the leave-one-out statistics for the accuracy of two prediction criteria. The first criterion uses only high-confidence predictions ($\phi_n \geq 0.8$) over a relatively long horizon ($h = 3$). The second criterion allows for lower-confidence predictions ($\phi_n \geq 0.5$) over a shorter horizon ($h = 2$). In Figure 4.20 we plot the Cartesian error for the 44 permutations as a function of δ for the two prediction criteria. The average median Cartesian error for both criteria, for most values of δ, was about 15 millimeters, or 8% of the distance traveled by the robot. The substantial overlap between the distributions means that neither criterion is different in a statistically significant sense. While the impact of model complexity, induced through δ, has a significant impact on Cartesian prediction accuracy,[7] the influence from the order in which programs are incorporated into the CDHMM dominates.

[7] High-confidence: $\rho = -0.83$, Low-confidence: $\rho = -0.61$; both have a p-value of $p \ll 0.01$.

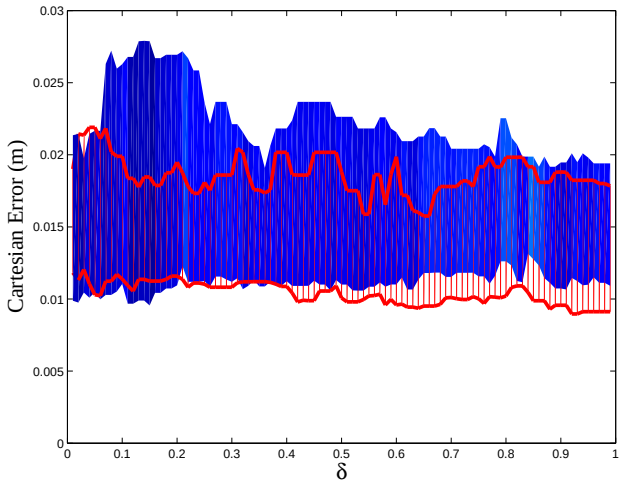


Figure 4.20: Cartesian errors caused by the different orderings of the online robot programs as a function of δ. The solid surface is the central 50% distribution for a high-confidence prediction criterion and the mesh surface shows the central 50% distribution for a low-confidence prediction criterion.

4.5.3 Impact on Programming Time

Users of the PRP system will be most interested in the reduction of the time needed to create robot programs. To determine the impact on programming time, users were asked to complete one of the tasks from Figure 4.17. The baseline is the time taken to create a robot program without PRP prediction support. The PRP system constructed a CDHMM from waypoints of other robot programs using a value of $\delta = 0.8$. We then asked users to create the programs with the PRP system offering prediction support with two different prediction criteria. As in Section 4.5.2, one criterion incorporates high-confidence predictions over a long horizon ($\phi_n \geq 0.8, h = 3$) and the other criterion allows low-confidence predictions over a short horizon ($\phi_n \geq 0.5, h = 2$). During programming, users move the robot with a joystick. When users feel that the current end-effector position is sufficiently close to the target waypoint, they press a button on the teach pendant. If the PRP system computes a prediction of sufficient confidence, this waypoint is suggested to users with an audible signal. Users can ignore the suggestion or they can allow the PRP system to move the robot to the predicted waypoint automatically by holding down another button. Users can stop the PRP system from moving the robot by releasing the button if, for example, there is an obstacle in its path. It is not uncommon for users to decide that the prediction must be refined. Fine-tuning can be done using conventional joystick positioning, or the previous waypoint can be restored by pressing an "undo" button. This fine-tuning or undoing time was included in the total programming time for that PRP prediction criterion result.

The impact of the PRP system on programming time is summarized in Table 4.1. The first row shows the

Prediction Criterion	mean (sec)	std (sec)	change	Wilcoxon Conf
None	292.2	78.61	N/A	N/A
$\phi_n \geq 0.8, h=3$	193.2	32.07	-33.88%	99.95%
$\phi_n \geq 0.5, h=2$	178.0	33.39	-39.08%	99.99%

Table 4.1: Programming time used to complete the tasks in Figure 4.17 with no prediction, high-confidence prediction, and low-confidence prediction.

baseline programming time used to complete the tasks without PRP prediction support. The second and third rows show the impact of PRP prediction using the high-confidence and low-confidence criteria, respectively. The Wilcoxon rank sum test confidence values (Fraser, 1957) in the final column indicate the statistical significance between the PRP prediction criteria and the baseline programming times. From Table 4.1 we see that programming time was reduced by over one third when using either PRP prediction criterion, with high statistical confidence. The difference in programming time between the two PRP prediction criteria was not significant.

4.5.4 Discussion of Results

The results contained in this section demonstrate the viability of PRP as a robot-programming tool. The prediction error for these online-programming tasks was much larger than the offline-programming counterparts. In the online-programming experiments, the prediction error was about 15 millimeters, or 8% of the distance traveled by the robot. In the offline-programming experiments, the prediction error was less than 0.25 millimeters, or 0.5% of the distance traveled by the robot. The prediction error for the online-programming experiments of about 15 millimeters is too large for arc-welding applications. This difference is primarily due to users providing their own definition of "sufficiently close" in the online-programming case, as opposed to the offline-programming experiments that required waypoints viable for arc-welding. Despite this variation, the PRP system was able to assist users in reducing programming time by over a third, in a statistically significant manner.

Not recorded in Table 4.1 was the number of collisions with objects in the environment. However, collisions are not uncommon during conventional operation when humans create automation programs, since, for example, arc-welding requires submillimeter separation from the workpiece. Industrial manipulators are designed to handle the eventual collision without inflicting damage. However, this sheds light on one potential problem with PRP in an online-programming environment: showing users the position of the predicted waypoint before moving the robot. There does not seem to be an efficient solution that allows users to visualize *a priori* where the PRP system intends to move the robot. One potential solution is to show users with a virtual-reality model (i.e., offline system) where the predicted waypoint is, allowing users to determine if the prediction is appropriate. But this implementation

may cause users to spend more time determining if the waypoint is acceptable than if they simply allowed the PRP system to move the robot automatically and (potentially) undid it. Our solution was to have users press and hold a button if they decide to let the PRP system move the robot. The PRP system moves toward the predicted waypoint slowly at first and then quickly comes to full speed. If users think the prediction is unacceptable, or that a collision may occur, then they can release the button to stop the PRP system immediately. This solution appears to work well in practice.

4.6 Relationship to Learning By Observation

Beyond its usefulness as an application in its own right, one of the goals of PRP was to test the ability for our CDHMM structure-estimation learning algorithm to model user subgoal selection in real-world demonstrations, as in Figure 1.3 and Figure 1.4. From this perspective, PRP removed extraneous factors such as sensor noise and environment considerations. Also, users were required to supply the PRP system with the task subgoals in the form of waypoints. This left the PRP system with the simplified LBO problem of predicting only the *next* subgoal (waypoint) in the sequence, without consideration for the specific actions needed to achieve that subgoal.[8] From the complex, real-world data, our PRP system was able to predict these subgoals with extremely high accuracy. Furthermore, the learning algorithm was able to estimate a CDHMM and predict the next subgoal in about 25 milliseconds, which is adequate for real-time use in an LBO system. One potential problem with our learning algorithm in an LBO application, mentioned in Section 4.4.1, is the relatively high perplexity of the estimated CDHMMs. On the offline-programming data in Section 4.3, our CDHMMs had an average perplexity of about 1.54. Intuitively, this means that, over 10 states, a random walker will be presented with 5 decisions about which state to choose next; for comparison, the algorithm of Stolcke and Omohundro (1994b) would require, on average, only 1 decision over the same state sequence. While we generated accurate predictions of a subsequent subgoal, this could mean that an LBO system using our CDHMM structure-estimation learning algorithm may have some difficulty determining the long sequences of subgoals needed to automate an entire task.

4.7 Summary

We have presented a novel method for predicting the waypoints in manipulator robot programs called Predictive Robot Programming. Using the learning algorithm derived in Chapter 3, we estimate the structure of CDHMMs based on previously created waypoints. This CDHMM is then used to predict future waypoints. On complex, real-

[8] Manipulator robots typically move between waypoints in one of three interpolated modes: joint-space linear, Cartesian-space linear, or Cartesian-space circular. These modes are simple to predict with a binary flag and nowhere near as complicated as the action sequences used in LBO applications.

world robot programs, the PRP system was able to generate a large percentage of highly accurate predictions. On all programs analyzed, the median Cartesian prediction error was well under a millimeter and the median angular prediction error was well under a milliradian. Both of these values are below the physical tolerances required by the target process, robotic arc welding. In a laboratory setting, we showed that the PRP system was able to reduce programming time by over a third in a statistically significant manner. These results suggest that PRP is a viable tool for reducing the time needed to program industrial manipulator robots.

Chapter 5

Hypothesizing About User Actions

This chapter describes our approach to hypothesizing about the response of the user to different conditions with sequenced linear dynamical systems. For the motor-skill tasks that we consider in this work, user actions correspond to a trajectory in state space. Our method uses a closed-form least-squares procedure to fit a single Linear Dynamical System (LDS) to a simple trajectory. These LDS estimates form the elemental building blocks used to describe complicated trajectories through an automatic online-segmentation procedure that can represent complicated trajectories with high accuracy. The response of each LDS correspond to a hypothesis about what the user would do under those conditions and we show how multiple trajectories can be incorporated to improve the generalization ability of the system. Each estimated LDS induces a supervisory control law, mapping current state to desired state, that encodes the target trajectory in a generative manner. We provide a proof of optimality and guarantee stability of the control law under a wide range of conditions.

5.1 Introduction

One of the key issues in machine learning is feature representation. When the objective is learning from human demonstrations, then the central idea is the representation of trajectories. LBO systems observe users performing tasks and synthesize the information to automate the task and achieve a goal, potentially in previously unseen conditions. These systems must be able to incorporate information from multiple demonstrations, including those occurring in different environments. Trajectory representation for LBO systems has somewhat different requirements from traditional robotics, since LBO systems must be able to reproduce tasks in previously unseen environments. In other words, LBO systems must emulate "what the user would have done" in novel conditions. Consequently, it

is essential that LBO systems represent trajectories in a *generative* manner; in other words, a control law mapping the current state of the system to the desired state. To facilitate learning, the representation should reduce the number of parameters needed to specify the trajectory, and similar trajectories should have similar parameters. The representation must be encoded so that it can be mapped to different environments easily to incorporate trajectories demonstrated in different environments. To address these requirements, we propose a trajectory-representation approach based on sequenced linear dynamical systems. A single Linear Dynamical System (LDS) can represent a trajectory in an extremely compact form by inducing a simple supervisory control law. Due to its simplicity, of course, a single LDS cannot faithfully reproduce the complicated trajectories needed by LBO systems. However, we create sophisticated behavior from a collection of simple components by segmenting complicated trajectories into simple ones, with each segment represented by a single LDS. Complicated trajectories can then be reproduced with high accuracy, using the LDS estimates in a sequential fashion.

We first develop the concept that a single LDS can represent a simple trajectory (Section 5.3). We use a closed-form least-squares procedure to estimate the optimal LDS for that trajectory, Equation 5.3. The supervisory control law induced by the LDS is then used to reproduce the trajectory (Section 5.4). We call the endpoint of the trajectory an *attractor*, because the control law tends, or is attracted, to that point. This generative representation is desirable since, for example, modifying the initial conditions causes the LDS to adapt its response to novel situations automatically. Similarly, we can modify the attractor to reproduce the trajectory in a different part of the workspace. By combining multiple demonstrations, the system can produce a better generalization of "what the user would have done" in a wide range of conditions. Since the method is based on the least-squares principle, it is straightforward to incorporate multiple demonstrations into the estimated LDS; we derive this formulation in Section 5.5. Because the representation is based on a control law, its stability is extremely important. In Section 5.6 we show that under reasonable conditions the control law will produce bounded trajectories that terminate at the desired attractor point. Specifically, we prove that an estimated LDS is stable in the sense of Lyapunov. To represent more complicated trajectories, we segment these trajectories into a sequence of simple trajectories, where each segment is represented by the single-LDS building block. In Section 5.7 we derive an online method for automatically segmenting a complicated trajectory using a normalized-prediction-error criterion. We demonstrate how the sequence of LDS estimates reconstructs a complicated trajectory in Section 5.8. The figures in this chapter use two-dimensional examples for illustrative purposes only; the ideas and the derivation of the LDS estimates generalize to an arbitrary dimension, as well as state-variable derivatives such as velocity and acceleration. Likewise, the proofs of optimality and stability are independent of the observation dimension.

5.2 Related Work

According to Ijspeert et al. (2001), "there seems to be consensus that among the most important desirable properties of movement encoding are: 1) the ease of representing and learning a goal trajectory, 2) compactness of the representation, 3) robustness against perturbations and changes in a dynamic environment, 4) ease of re-use for related tasks and easy modification for new tasks, and 5) ease of categorization for movement recognition." To varying degrees, these criteria are applicable to the motor-skill LBO systems that we consider. We would also recommend the ability to incorporate multiple examples of a trajectory in order to infer user *intent*. The most obvious method for representing a trajectory, cubic-spline interpolation (Craig, 1989), is sensitive to changes in initial conditions and is difficult to recompute for different operating conditions. A similar method for representing the trajectories of mobile robots using piecewise-cubic Bézier curves has also been proposed (Hwang et al., 2003). As Schaal et al. (2003) note, spline methods "are not very robust in coping with unforeseen perturbations of the movement" and require "more complex computations in terms of scaling laws." This observation underscores the need to represent trajectories in a generative manner. Other trajectory-representation methods include symbolic `if-then` coding (Nicolescu, 2003), neural networks (Friedrich et al., 1996), and optimization methods (Schaal, 1997). Ijspeert et al. (2002) describe a trajectory-representation method based on nonlinear attractor dynamics. This approach is particularly appealing, since it is a generative method, providing a control law so that a system can reproduce trajectories in different environment configurations by adjusting the attractor point. This method also allows the representation of a wide variety of trajectories, including discrete and rhythmic movements. However, this formulation is sensitive to the speed with which trajectories are demonstrated, which may be undesirable in some applications. This system can also modulate the approximated trajectory based on objects in the environment. Due to its internal representation, this approach only allows a single scaling or shifting parameter applied to the entire trajectory, so if only some objects in the environment move then complete retraining would be required. Estimating the parameters of a linear, time-invariant dynamical system is squarely within the domain of System Identification (Ljung, 1999). However, this field is primarily concerned with experimental setup and unbiasedness, less with stability and compactness. In some sense, we have co-opted an elementary concept from system identification and are applying the ideas to trajectory representation.

5.3 Simple Trajectory with a Known Attractor Point

Consider a trajectory sampled at discrete intervals, $\boldsymbol{X} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, where each observation is a $(d \times 1)$ real vector and there are more observations than rows, $N \geq d$. Our approach attempts to find an LDS that captures the "shape" of the trajectory and terminates at the final observation. In other words, the supervisory control law induced by the LDS should be stable and have its unique attractor point at the final observation in the trajectory,

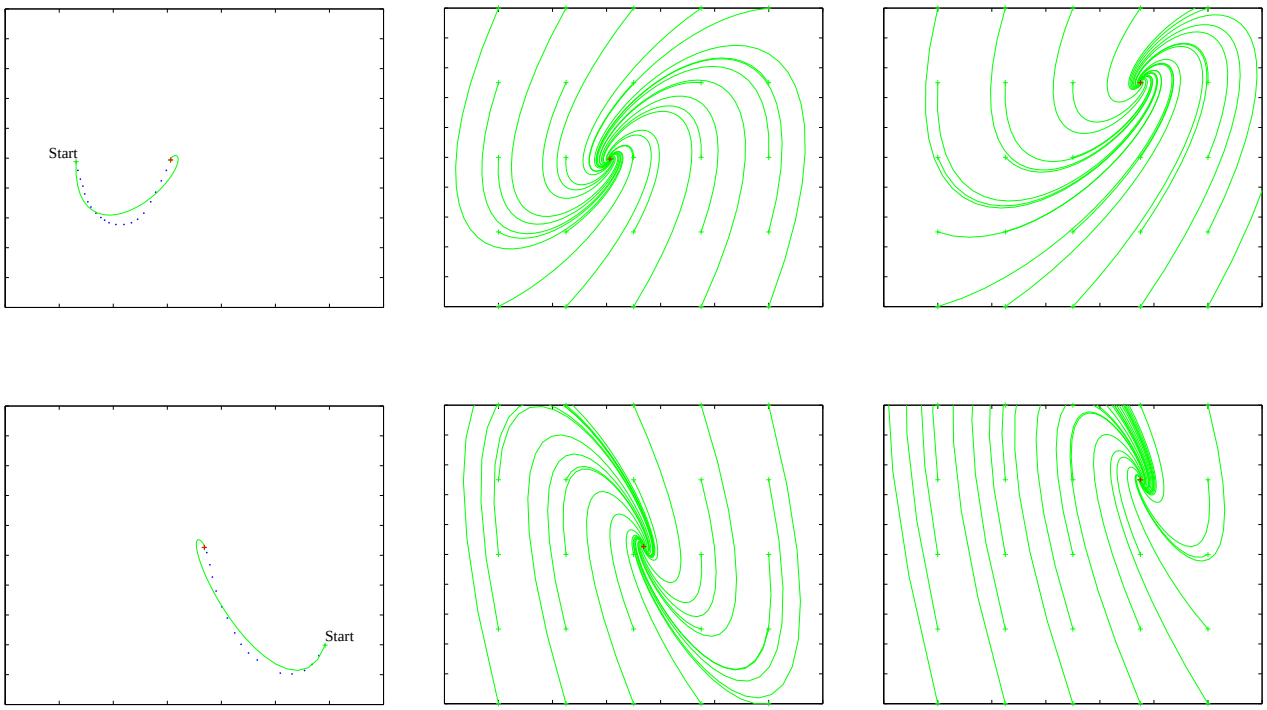


Figure 5.1: The left column shows two simple trajectories fitted by an LDS according to Equation 5.3. The dotted line represents the samples of the user trajectory and the solid line represents the trajectory reconstructed by Equation 5.4. The middle column shows the response of the LDS to various initial conditions, and the right column shows the response of the LDS when the attractor is shifted.

$\boldsymbol{x}_N$. A discrete-time LDS with constant input is given by the difference equation

$$\boldsymbol{x}_{n+1} = \boldsymbol{A}\boldsymbol{x}_n + \boldsymbol{B}\boldsymbol{u}.$$

Assuming the system is stable and the matrix $(\boldsymbol{I} - \boldsymbol{A})^{-1}$ exists, then the unique attractor, $\boldsymbol{x}_\infty$, is computed as

$$\boldsymbol{x}_\infty = (\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}\boldsymbol{u}.$$

By choosing $\boldsymbol{B} = (\boldsymbol{I} - \boldsymbol{A})$, then the input is the point of attraction, $\boldsymbol{x}_\infty = \boldsymbol{u}$. Using the LDS formulation, we assume that observations are generated according to

$$\begin{aligned} \boldsymbol{x}_{n+1} &= (\boldsymbol{I} + \boldsymbol{R})\boldsymbol{x}_n - \boldsymbol{R}\boldsymbol{x}_N \\ &= \boldsymbol{R}(\boldsymbol{x}_n - \boldsymbol{x}_N) + \boldsymbol{x}_n. \end{aligned} \tag{5.1}$$

We give Equation 5.1 this particular form because, if the matrix $(\boldsymbol{I}+\boldsymbol{R})$ is stable, then the attractor of Equation 5.1 is $\boldsymbol{x}_N$, since $\boldsymbol{B} = (\boldsymbol{I} - \boldsymbol{A})$. Given a trajectory, we rewrite Equation 5.1 as the column-stacked matrix equation

$$\begin{aligned} [\boldsymbol{x}_1 \cdots \boldsymbol{x}_N] &= \boldsymbol{R}\left([\boldsymbol{x}_0 \cdots \boldsymbol{x}_{N-1}] - [\boldsymbol{x}_N \cdots \boldsymbol{x}_N]\right) + [\boldsymbol{x}_0 \cdots \boldsymbol{x}_{N-1}], \\ &\Rightarrow \\ \boldsymbol{X}_{1:N} &\doteq \boldsymbol{R}\left(\boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N\right) + \boldsymbol{X}_{0:N-1}. \end{aligned} \tag{5.2}$$

Theorem 5.1. *If the matrix $(\boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N)$ has full row rank, then the least-squares solution for $\boldsymbol{R}$ is*

$$\widehat{\boldsymbol{R}} = \left(\boldsymbol{X}_{1:N} - \boldsymbol{X}_{0:N-1}\right)\left(\boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N\right)^{\mathsf{R}}, \tag{5.3}$$

where $\boldsymbol{C}^{\mathsf{R}}$ denotes the right pseudoinverse of $\boldsymbol{C}$.

The proof is given in Section A.1.8. The estimated matrix $\widehat{\boldsymbol{R}}$ captures the salient features of the trajectory such as direction, curvature, and speed, as in Figure 5.1. The final observation in the demonstrated trajectory, $\boldsymbol{x}_N$, becomes the attractor, which specifies the terminus of the estimated trajectory.

5.4 Reproducing a Simple Trajectory

To reproduce the trajectory, the LDS difference equation is initialized with $\widehat{\boldsymbol{x}}_0 \equiv \boldsymbol{x}_0$, which induces from Equation 5.1 the autonomous control law

$$\widehat{\boldsymbol{x}}_{n+1} = \widehat{\boldsymbol{R}}\left(\widehat{\boldsymbol{x}}_n - \boldsymbol{x}_N\right) + \widehat{\boldsymbol{x}}_n. \tag{5.4}$$

This iteration repeats until reaching the stopping criterion, $\|\widehat{\boldsymbol{x}}_n - \boldsymbol{x}_N\|_2 \leq \zeta$, where $\zeta > 0$ is some scalar threshold. Since the trajectory is reproduced using an autonomous control law, it can be modified by shifting the initial conditions, the attractor point, or both; there are no complicated scaling laws that plague other trajectory-representation methods such as spline interpolation (Craig, 1989). Modifying either the initial conditions or the attractor will cause the estimated trajectory to stretch, shrink, etc. Shifting the initial conditions and the attractor by the same amount would cause Equation 5.4 to reproduce a trajectory of the same shape, simply offset by the shift amount. In Figure 5.1 we use Equation 5.3 to fit an LDS to different trajectories and then use Equation 5.4 to reproduce the estimated trajectories. We also show the response of the LDS to various initial conditions and to a shifted attractor point. The response of the LDS to different initial conditions corresponds to the hypothesis of "what the user would have done" in that situation.

When applied to the real-time control of a system such as a robot, then the deterministic iteration in Equa-

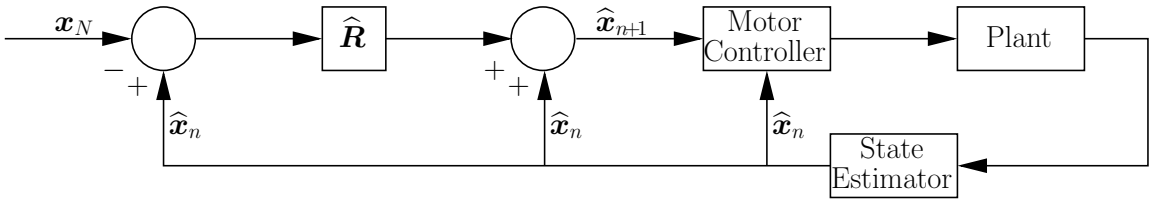


Figure 5.2: Block diagram for real-time control using an estimated LDS.

tion 5.4 becomes the desired state of the system. Like other researchers (Schaal et al., 2003), we assume that the system can estimate its current state, $\widehat{\boldsymbol{x}}_n$, and that there exists a low-level controller that can actuate toward a desired state, $\widehat{\boldsymbol{x}}_{n+1}$. The block diagram for the supervisory control strategy using the estimated LDS is shown in Figure 5.2. However, if the system is nonholonomic or saturating, then it may not be able to achieve the desired state within the allotted time. At each iteration of the control locus, the system senses its new state and computes a new desired state. In this sense, Equation 5.4 computes the sequence of states specifying *how* to achieve a goal, even if the system diverges from the intended trajectory.

5.5 Combining Multiple Trajectories

Since our representation uses a least-squares procedure, it is simple to incorporate multiple examples of a trajectory to create an estimate that generalizes better. Suppose we have two demonstrations of a trajectory, $\boldsymbol{X}^1 = \{\boldsymbol{x}_0^1, \ldots, \boldsymbol{x}_{N_1}^1\}$ and $\boldsymbol{X}^2 = \{\boldsymbol{x}_0^2, \ldots, \boldsymbol{x}_{N_2}^2\}$, though in principle and in practice the following method works with an arbitrary number of demonstrations. We rewrite Equation 5.2 by column-stacking both trajectories and define the matrices:

$$\begin{aligned}
\boldsymbol{X}_{1:N}^{1,2} &\triangleq \left[\boldsymbol{x}_1^1 \cdots \boldsymbol{x}_{N_1}^1 \boldsymbol{x}_1^2 \cdots \boldsymbol{x}_{N_2}^2\right]; \\
\boldsymbol{X}_{0:N-1}^{1,2} &\triangleq \left[\boldsymbol{x}_0^1 \cdots \boldsymbol{x}_{N_1-1}^1 \boldsymbol{x}_0^2 \cdots \boldsymbol{x}_{N_2-1}^2\right]; \\
\boldsymbol{\Gamma}_N^{1,2} &\triangleq \left[\boldsymbol{x}_{N_1}^1 \cdots \boldsymbol{x}_{N_1}^1 \boldsymbol{x}_{N_2}^2 \cdots \boldsymbol{x}_{N_2}^2\right].
\end{aligned}$$

Similar to Equation 5.3, the least-squares estimate for these combined trajectories is

$$\widehat{\boldsymbol{R}}^{1,2} = \left(\boldsymbol{X}_{1:N}^{1,2} - \boldsymbol{X}_{0:N-1}^{1,2}\right)\left(\boldsymbol{X}_{0:N-1}^{1,2} - \boldsymbol{\Gamma}_N^{1,2}\right)^{\mathsf{R}}. \tag{5.5}$$

The ability for an LBO system to incorporate multiple demonstrations of a task is critical to inferring user intent. Computing the least-squares solution to the LDS allows our system to hypothesize better about "what the user would have done" by reducing the unintended, demonstration-specific artifacts often found in user actions. In Fig-

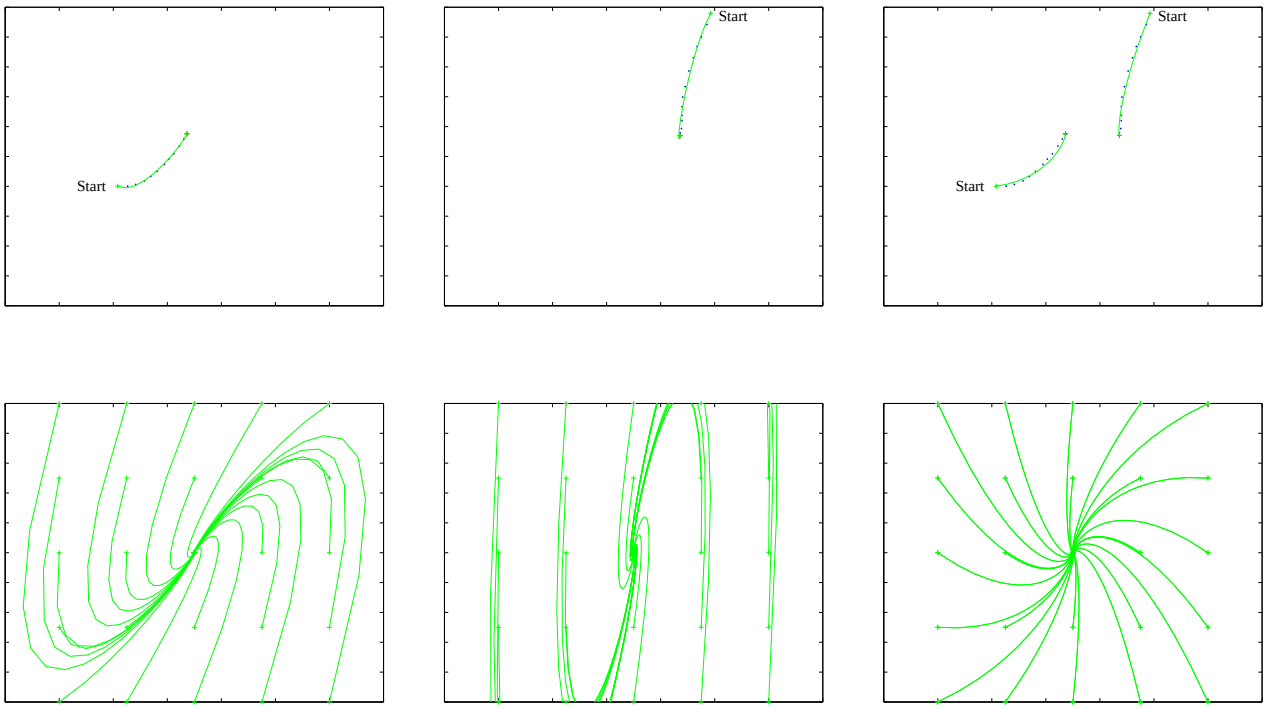


Figure 5.3: The left and middle columns show two examples of trajectories with a slight counter-clockwise curvature, with the original trajectories on top and the response of the LDS on bottom. Individually, neither estimate produces the intended generalization. In the right column is the combined estimate of the two trajectories, computed from Equation 5.5, which yields the desired slight counter-clockwise generalization.

ure 5.3 we provide two examples of a simple trajectory, both with a slight counter-clockwise curvature. The LDS fitted to each individual trajectory is optimal in the least-squares sense. Considering the response of these control laws to various initial conditions, it is clear that the individual LDS estimates do not generalize to "slight counter-clockwise curvature." However, computing the least-squares estimate of the *combined* trajectories produces the intended generalization.

5.6 Stability of the LDS Estimate

Define the vector $\boldsymbol{\delta}_n \triangleq \boldsymbol{x}_n - \boldsymbol{x}_N$ and the column-stacked matrices

$$\begin{aligned} \boldsymbol{\Delta}_{1:N} &\triangleq \boldsymbol{X}_{1:N} - \boldsymbol{\Gamma}_N \\ &\equiv [\boldsymbol{\delta}_1 \cdots \boldsymbol{\delta}_N]\,; \\ \boldsymbol{\Delta}_{0:N-1} &\triangleq \boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N \\ &\equiv [\boldsymbol{\delta}_0 \cdots \boldsymbol{\delta}_{N-1}]\,. \end{aligned}$$

If the matrix $\boldsymbol{\Delta}_{0:N-1}$ has full row rank, then we can rewrite Equation 5.3 as

$$\begin{aligned} \widehat{\boldsymbol{R}} &= (\boldsymbol{X}_{1:N} - \boldsymbol{X}_{0:N-1})\,(\boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N)^{\mathsf{R}} \\ &= ((\boldsymbol{X}_{1:N} - \boldsymbol{\Gamma}_N) - (\boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N))\,(\boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N)^{\mathsf{R}} \\ &= (\boldsymbol{X}_{1:N} - \boldsymbol{\Gamma}_N)\,(\boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N)^{\mathsf{R}} - \boldsymbol{I} \\ &\doteq \boldsymbol{\Delta}_{1:N}\boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{R}} - \boldsymbol{I}. \end{aligned}$$

We note that the estimated LDS is

$$\begin{aligned} \boldsymbol{x}_{n+1} &= \widehat{\boldsymbol{R}}\,(\boldsymbol{x}_n - \boldsymbol{x}_N) + \boldsymbol{x}_n \\ &= \left(\widehat{\boldsymbol{R}} + \boldsymbol{I}\right)\boldsymbol{x}_n - \widehat{\boldsymbol{R}}\boldsymbol{x}_N \\ &\doteq \left(\boldsymbol{\Delta}_{1:N}\boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{R}} - \boldsymbol{I} + \boldsymbol{I}\right)\boldsymbol{x}_n - \widehat{\boldsymbol{R}}\boldsymbol{x}_N \\ &= \left(\boldsymbol{\Delta}_{1:N}\boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{R}}\right)\boldsymbol{x}_n - \widehat{\boldsymbol{R}}\boldsymbol{x}_N. \end{aligned}$$

There are two broad approaches to ensure the stability of a discrete time-invariant dynamical system. The primary method for *linear* systems is showing that the system matrix is *convergent*, i.e., having eigenvalues inside the unit circle (Antsaklis & Michel, 1997). The magnitude of the largest eigenvalue of a matrix $\boldsymbol{A}$ is called the *spectral radius* and is written $\rho(\boldsymbol{A})$. If the matrix $\boldsymbol{A}$ is constant and the values known, then the eigenvalues can be evaluated directly to ensure stability. When the system matrix is determined algorithmically from inputs unknown *a priori*, then properties of the spectral radius must be invoked. For instance, it is well known that $\rho(\boldsymbol{A}) \leq \|\boldsymbol{A}\|_p$, where $\|\cdot\|_p$ is any of the matrix p-norms (Golub & Van Loan, 1996). We note that our LDS estimate is computed from the product of two matrices, $\boldsymbol{\Delta}_{1:N}\boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{R}}$. There are recent results indicating that bounding the eigenvalues of a product of two matrices is, in general, *undecidable* (Blondel & Tsitsiklis, 2000). Even with this discouraging result, it is certainly possible to place restrictions on the constituent matrices to bound the eigenvalues of the product. Using this approach, we investigated a variety of conditions to guarantee stability by restricting the class

of acceptable trajectories. Unfortunately, we were unsuccessful in deriving any *useful*, *reasonable*, or *meaningful* conditions. Our difficulty seems in line with anecdotal evidence from other researchers (Blondel et al., 2003).

The more general approach for ensuring stability of a linear or nonlinear dynamical system is by Lyapunov's direct (or second) method. In the discrete-time LDS case, for stability in the sense of Lyapunov (i.s.L.), it is sufficient to find a symmetric PD matrix $\widetilde{\boldsymbol{P}}$ such that $\widetilde{\boldsymbol{P}} - \boldsymbol{A}^{\mathsf{T}}\widetilde{\boldsymbol{P}}\boldsymbol{A} = \widetilde{\boldsymbol{Q}}$, where $\widetilde{\boldsymbol{Q}}$ is some symmetric PSD matrix (Antsaklis & Michel, 1997). Since the eigenvalues of $\boldsymbol{A}$ and $\boldsymbol{A}^{\mathsf{T}}$ are the same, stability i.s.L. can be determined equivalently by choosing a symmetric PD matrix $\boldsymbol{P}$ such that

$$\boldsymbol{P} - \boldsymbol{A}\boldsymbol{P}\boldsymbol{A}^{\mathsf{T}} \;=\; \boldsymbol{Q}, \tag{5.6}$$

where $\boldsymbol{Q}$ is some symmetric PSD matrix. This equation sometimes goes by the name of the Discrete Algebraic Lyapunov Equation (DALE). Finding the matrix $\boldsymbol{P}$ is more of an art than a science and typically relies on problem-specific intuition. Applying this intuition sidesteps the undecidability problem of bounding the eigenvalues of a product of arbitrary matrices mentioned earlier.

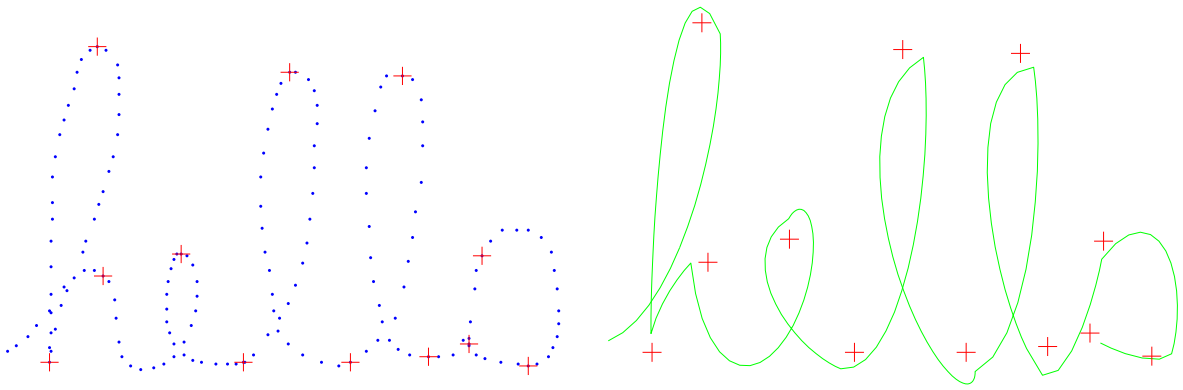
Theorem 5.2. *Equation 5.4 is stable i.s.L. about the attractor $\boldsymbol{x}_N$ if the matrix $\boldsymbol{\Delta}_{0:N-1}$ has full row rank.*

The proof is given in Section A.1.10.

The assumptions of Theorem 5.2 imply that, if the trajectory is anything other than a perfect line, then Equation 5.4 will produce bounded trajectories that terminate at the desired attractor. However, as observations become increasingly collinear, there could be numerical-precision problems, causing the matrix $\boldsymbol{\Delta}_{0:N-1}$ to become effectively rank deficient. This is typically manifested by the estimated matrix $\widehat{\boldsymbol{R}}$ becoming poorly conditioned, which can be determined by checking if the condition number exceeds some threshold, $\kappa(\widehat{\boldsymbol{R}}) > \kappa_{\max}$. A poorly conditioned matrix is then replaced by a scaled identity matrix. Since every vector is an eigenvector of a scaled identity matrix, Equation 5.4 will then induce a control law that produces straight lines terminating at the attractor from any initial condition.

5.7 Complicated Trajectories with Unknown Attractor Points

In this section, we consider trajectories that cannot be faithfully reproduced using a single LDS estimate. We formulate the problem as finding a *sequence* of LDS estimates that represent the complicated trajectory. Furthermore, we assume that there are no auxiliary signals indicating an appropriate segmentation. Somewhat arbitrarily, we pursued a method that allows the system to perform the segmentation *online*, as observations become available, instead of a batch-processing method that requires the trajectory be complete. It appears that finding the optimal segmentation of a trajectory has no closed-form solution and is heuristic in nature, similar to determining the op-



(a) (b)

Figure 5.4: Trajectory of the cursive word "hello" with attractors extracted automatically using Equation 5.7 (Figure 5.4(a)). The solid curve indicates the estimated trajectory from Equation 5.4 (Figure 5.4(b)).

timal number of clusters in the k-means algorithm. Our segmentation heuristic is based on the predictability of subsequences of the trajectory, where its observations can be modeled by a single LDS. At an unknown point in time, N_i, the ith subsequence is no longer well represented by the LDS that generated recent observations. To determine this segmentation, we compute a prediction of the next observation in the trajectory at each time step. If the prediction is sufficiently accurate, then we consider the current LDS estimate appropriate. If the prediction is inaccurate, then a new LDS should be computed. Specifically, at time n, we estimate an LDS according to Equation 5.3 between time N_{i-1} and time $n-1$. This matrix is written as $\widehat{\boldsymbol{R}}_{N_{i-1}:n-1}$. We assign $\boldsymbol{x}_n$ as the attractor of the LDS and predict the previous observation,

$$\widehat{\boldsymbol{x}}_{n-1} \quad = \quad \widehat{\boldsymbol{R}}_{N_{i-1}:n-1} \left(\boldsymbol{x}_{n-2} - \boldsymbol{x}_n \right) + \boldsymbol{x}_{n-2}.$$

We define the normalized prediction error as

$$\varepsilon_n \quad \triangleq \quad \frac{\left\| \boldsymbol{x}_{n-1} - \widehat{\boldsymbol{x}}_{n-1} \right\|_2}{\left\| \boldsymbol{x}_{n-1} - \boldsymbol{x}_{n-2} \right\|_2}. \tag{5.7}$$

This definition attempts to normalize the error so that its value is independent of the speed or sampling rate of the trajectory. For instance, if the user moves very quickly, then we expect predictions to be less accurate, on an absolute scale. On the other hand, when a trajectory is sampled at a relatively high rate, we expect predictions to be more accurate on an absolute scale, since not much time has elapsed between observations. Normalizing by the distance between observations accounts for this discrepancy, so that an inaccurate prediction is somewhat invariant

with respect to these issues. If the normalized prediction error exceeds a predetermined threshold, $\varepsilon_n > \varepsilon_{\max}$, then we assume that the observation $\boldsymbol{x}_n$ is not an appropriate attractor for this segment. Therefore, we assign the segmentation time as $N_i = n - 1$ and the attractor as $\boldsymbol{x}_{N_i} = \boldsymbol{x}_{n-1}$. We then segment the trajectory between time N_{i-1} and time N_i and compute the corresponding LDS as in Equation 5.3,

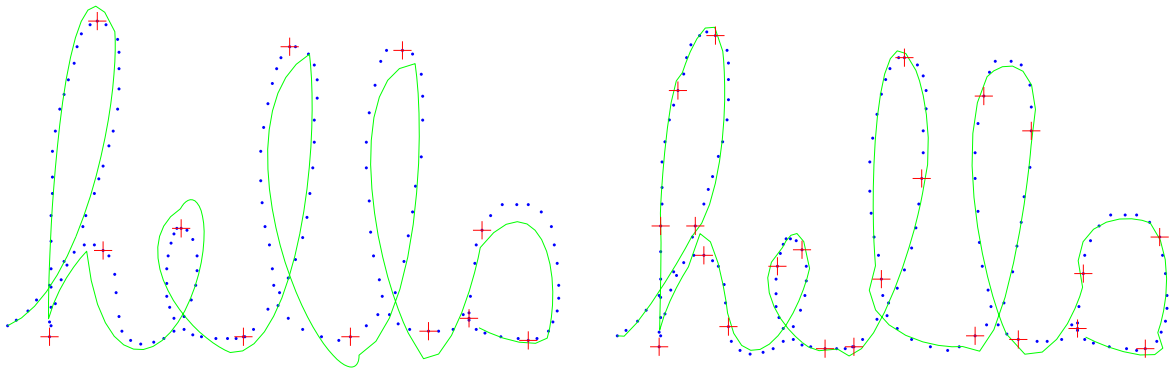
$$\widehat{\boldsymbol{R}}_{N_i} \quad = \quad \left(\boldsymbol{X}_{N_{i-1}+1:N_i} - \boldsymbol{X}_{N_{i-1}:N_i-1}\right)\left(\boldsymbol{X}_{N_{i-1}:N_i-1} - \boldsymbol{\Gamma}_{N_i}\right)^{\mathsf{R}}.$$

After the entire trajectory has been segmented, a sequence of M LDS estimates, $\{(\widehat{\boldsymbol{R}}_{N_1}, \boldsymbol{x}_{N_1}), \ldots, (\widehat{\boldsymbol{R}}_{N_M}, \boldsymbol{x}_{N_M})\}$, now describes the complicated trajectory. In most cases this sequence of matrix-attractor pairs will reduce the number of parameters needed to represent the trajectory. Let N be the number of $(d \times 1)$ observations in the original trajectory. The number of parameters used to specify the trajectory is then Nd. With the sequenced-LDS method, the number of parameters needed to specify the approximated trajectory is $M(d^2 + d)$, where M is the number of LDS estimates, since d^2 numbers are needed for the matrix $\widehat{\boldsymbol{R}}_{N_i}$ and d for the attractor $\boldsymbol{x}_{N_i}$. In Figure 5.4, we apply the segmentation algorithm to the trajectory of the cursive word "hello." The original trajectory is specified by 350 numbers. The LDS representation used 12 matrix-attractor pairs, or 72 numbers, a reduction of almost 80%. However, the reduced-parameter representation results in an error between the original and approximated trajectories. Like many algorithms, there is a trade-off between the expressiveness of the system and the number of parameters used. In Figure 5.5 we segment the cursive word "hello" with high and low values of the normalized-prediction-error threshold, $\varepsilon_{\max}$. Qualitatively, the approximated trajectory in Figure 5.5(a) is less faithful to the original trajectory than the estimate in Figure 5.5(b), which uses nearly twice as many matrix-attractor pairs.

5.7.1 Analysis of Normalized-Prediction-Error Threshold

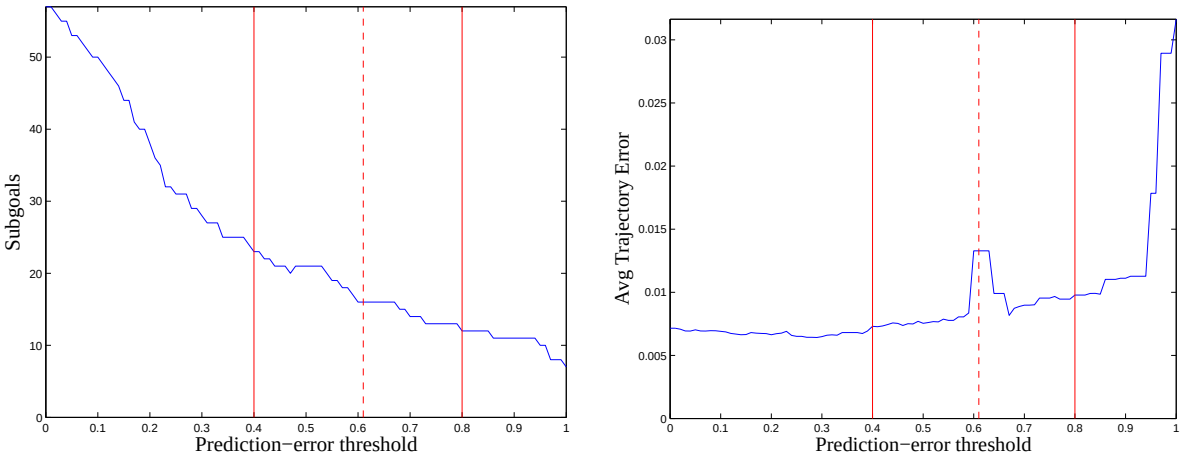
To quantify this trade-off between simplicity and accuracy, we compute the error between the original and approximated trajectories. The error is defined to be the Euclidean distance between an observation in the original trajectory and the closest point in the approximated trajectory. In Figure 5.6 we plot the resulting average error and number of matrix-attractor pairs as a function of the normalized-prediction-error threshold, $\varepsilon_{\max}$. The curves show the anticipated result: as the number of parameters decreases, the average error generally increases. The *optimal* value for the normalized-prediction-error threshold is task dependent: some tasks require high accuracy, while others favor a simpler representation. As such, the correct value of $\varepsilon_{\max}$ depends on the task at hand.

However, since we use an online approach to trajectory segmentation, lower values of the normalized-prediction-error threshold do not necessary result in better approximation. Notice the "bump" in the approximation-error plot, around the value of $\varepsilon_{\max} \approx 0.6$, highlighted by the dashed line in Figure 5.6(b). This temporary increase in approximation error results because our method computed a poor segmentation of the trajectory, as in Figure 5.7. Locally optimal segmentations resulting in poor global error arise because the segmentation algorithm is unaware of future



(a) (b)

Figure 5.5: Representation of the cursive word "hello" from different values of the normalized-prediction-error threshold. The segmentation with $\varepsilon_{\max} = 0.8$ in Figure 5.5(a) uses 12 matrix-attractor pairs. The segmentation with $\varepsilon_{\max} = 0.4$ in Figure 5.5(b) uses 22 matrix-attractor pairs.



(a) (b)

Figure 5.6: Trade-off between simplicity and accuracy: number of matrix-attractor pairs and average approximation error as a function of normalized-prediction-error threshold.

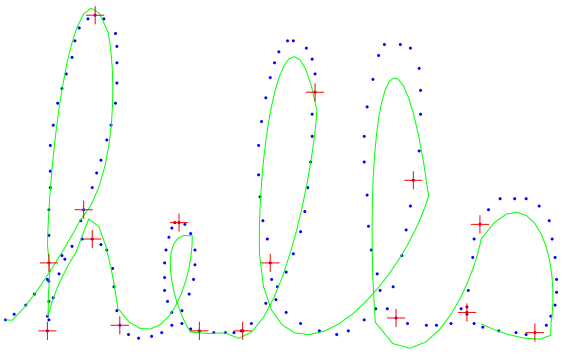


Figure 5.7: Some values of the normalized-prediction-error threshold result in poor approximations using our online-segmentation method.

observations. These idiosyncrasies are inevitable for some trajectories when using an online-segmentation method such as ours.

5.8 Reproducing a Complicated Trajectory

To reproduce a complicated trajectory, given a sequence of matrix-attractor pairs, we simply initialize the difference equation with $\widehat{\boldsymbol{x}}_0 = \boldsymbol{x}_0$ and repeatedly apply

$$\widehat{\boldsymbol{x}}_{n+1} \quad = \quad \widehat{\boldsymbol{R}}_{N_1} \left(\widehat{\boldsymbol{x}}_n - \boldsymbol{x}_{N_1} \right) + \widehat{\boldsymbol{x}}_n,$$

until achieving $\| \widehat{\boldsymbol{x}}_n - \boldsymbol{x}_{N_1} \|_2 \leq \zeta$. At this point, the next matrix-attractor pair is used $(\widehat{\boldsymbol{R}}_{N_2}, \boldsymbol{x}_{N_2})$, and so forth, until the final attractor point is reached, as in Figure 5.4(b). Since each LDS estimate is stable i.s.L., any trajectories generated by the difference equations will be bounded and terminate at the final attractor point in finite time. Finite termination is guaranteed regardless of initial conditions *or* deviations from the prescribed control law.

5.9 Relationship to Learning By Observation

There are several aspects of the sequenced-LDS representation that are appealing for LBO applications. When a complicated trajectory is segmented, the attractors mark important points in the trajectory. In an LBO system, these attractors become the *subgoals* of the demonstrated task. For example, in our LBO system (Chapter 6), the subgoals are associated with objects in the environment. As the objects move, the subgoals update accordingly. Because the trajectory is represented in a generative manner, the LBO system can modify the locations of the

attractors and each LDS will adapt its response to these new conditions automatically. This allows our LBO system to hypothesize about what actions the user would perform in novel conditions. The ability to combine multiple trajectories together to estimate an LDS that generalizes better is helpful in inferring user intent.

At the same time, we note that our proofs of optimality and stability do not respect the dynamics of the robot and are guaranteed only for idealized systems. This ignores the fact that all robots have saturating actuators and many are nonholonomic. It would be useful to provide system-specific proofs, but we note that, for example, showing the stability of saturating dynamical systems is generally undecidable (Blondel et al., 2000).

5.10 Summary

We have presented a novel method for representing trajectories using sequenced linear dynamical systems. Simple trajectories are represented by a single LDS estimate while complicated trajectories use the sequenced representation. The LDS estimates are computed in closed form by a least-squares procedure. We showed that multiple demonstrations can be combined in order to improve generalization. We provided a proof of optimality and we guarantee stability of the LDS estimates under reasonable conditions. In the introduction, we posed several questions about hypotheses of user actions (Section 1.3.2). Specifically, the first question was:

- Can we optimally represent hypotheses of user actions to novel conditions?

In our sequenced-LDS formulation, we define optimality as minimizing the squared one-step prediction error and derived an optimal supervisory control law (Theorem 5.1). This allows our system to compute least-squares hypotheses based on the available information. However, the term *optimal* is somewhat ambiguous and there are other reasonable interpretations of "optimality." While the control law is optimal with respect to the demonstrations, our formulation ignores the dynamics of the system and environment factors, such as collision avoidance. These considerations seem like plausible extentions and, after writing down a precise cost function, it should be possible to derive an optimal estimate, though it may take quite a bit of effort to arrive at the answer. The second question asked was:

- Under what conditions will the model succeed and fail?

We showed that the supervisory control law is stable i.s.L. (Theorem 5.2) for essentially all possible demonstrations. The exception is when the observations from a trajectory form a straight line. However, this condition can be checked quickly and ameliorated with a simple `if − then` statement (Section 5.6). Another potential problem with our formulation arrise from the online approach we use to segment complicated trajectories (Section 5.7). It may be more intuitive to specify the segmentation according to an allowable error and performing offline (i.e., batch) segmentation so that the result achieves an error less than the requested amount.

Chapter 6

Learning By Observation

We present a computational approach to Learning By Observation (LBO) that allows users to program mobile robots by demonstrating a task. Unlike previous approaches, our system incorporates statistical-learning techniques and concepts from control theory to reduce the amount of domain knowledge needed to infer the intent of users. To improve the generalization ability of the system, users can demonstrate the task multiple times. We extract task subgoals from these demonstrations and automatically associate them with objects in the environment. As these objects move, the subgoals are updated accordingly. This gives our system the ability to learn from demonstrations of a task performed in different environments. We describe the concepts used in our LBO system, as well as experimental laboratory results in learning motor-skill tasks.

6.1 Introduction

User demonstrations can be viewed as a sequence of goal-directed actions. Consider a user teaching a robot to vacuum a room by walking the route. It is undesirable to require retraining each time furniture is moved. By capturing the *intent* of users, an LBO system can determine the sequence of subgoals and overall goal of the task, despite changes in the environment, human imprecision, or sensor noise. The objective of our work is to create an LBO system that reduces the dependence on *a priori* task-specific information while producing robust behavior. To this end, we have created an LBO system called "Dollop"[1] that employs statistical-learning techniques, as well as concepts from control theory, to allow users to program a mobile robot through one of the most natural methods possible: walking around. The target tasks for Dollop are motor-learning skills where the demonstrations may occur in different environments. We show that a computational approach to LBO can extract user intentions

[1] In contrast to many engineering projects, Dollop is not an acronym.

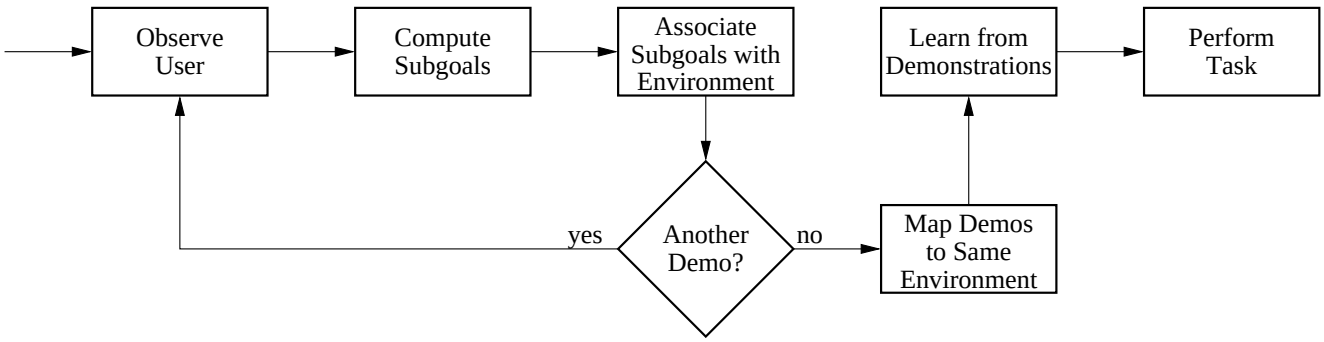


Figure 6.1: Conceptual flow diagram of Dollop.

Figure 6.2: Agent Orange, the mobile robot used in these experiments.

accurately in laboratory experiments.

The conceptual flow diagram of Dollop is shown in Figure 6.1. While observing users, Dollop extracts important points, or subgoals, from the trajectory from the sequenced Linear Dynamical System (LDS) formulation (Section 6.3). These subgoals are automatically associated with objects in the environment so that as the environment changes the subgoals are updated accordingly (Section 6.4). Dollop forms a hypotheses of user intent by allowing the LDS estimates to adapt their responses automatically to these new conditions. When learning from multiple demonstrations performed in different environments, Dollop hypothesizes what the user would have have done if all demonstrations had been performed in the same environment (Section 6.5). Our learning algorithm (Chapter 3) then estimates a CDHMM that describes the subgoals from the demonstrations and determines the most likely sequence of subgoals needed to complete the task.

The mobile robot used in these experiments was an ActivMedia Pioneer II DX named Agent Orange, equipped with a SICK scanning laser range finder, shown in Figure 6.2. We used the Carnegie Mellon Robot Navigation

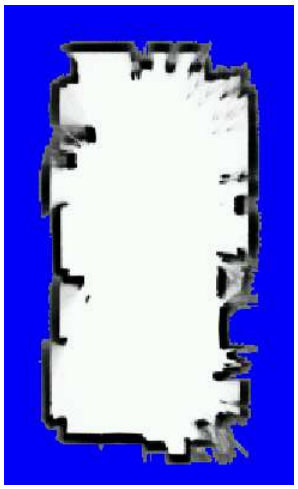


Figure 6.3: Background grid map of the laboratory.

Toolkit (Carmen) (Montemerlo et al., 2003) to provide base services such as localization, mapping, and low-level control. The localization is based on a variant of Monte Carlo Localization (MCL) (Thrun et al., 2001). Before operation, Agent Orange is driven around the workspace manually with a joystick, and Carmen computes an occupancy grid map of the area, as in Figure 6.3. (Computing a map need only be done during the first use in a given area.) During operation, Carmen uses a laser reading to compute the likelihood of being at various locations and orientations in the map, building a probabilistic hypothesis about the position of the robot in the workspace. As the robot moves, Carmen incorporates odometry information to update its hypothesis; subsequent laser readings cause Carmen to refine its position estimate further. This iterative process repeats until the hypothesis converges. If properly initialized, the hypothesis converges in well under a second and, in our experience, the estimated position appears accurate to within a centimeter or two. MCL frees Dollop from relying on external-reference sensors such as GPS, or error-accumulating methods such as dead-reckoning.

6.2 Observing Users

The primary sensor used in these experiments is a scanning laser range finder. This sensor gives a two-dimensional polar-coordinate binary reading that indicates the range to the nearest object along a particular bearing. The laser scans the horizontal plane at a fixed height, determined by its position on Agent Orange, about a half meter above the floor. The angular resolution is one degree and the distance resolution is one centimeter. In terms of gross errors, the laser appears to return only false-negative readings, i.e., indicating free space when an object is present. This seems to be caused when the object does not reflect enough energy back to the receiver due to a glancing

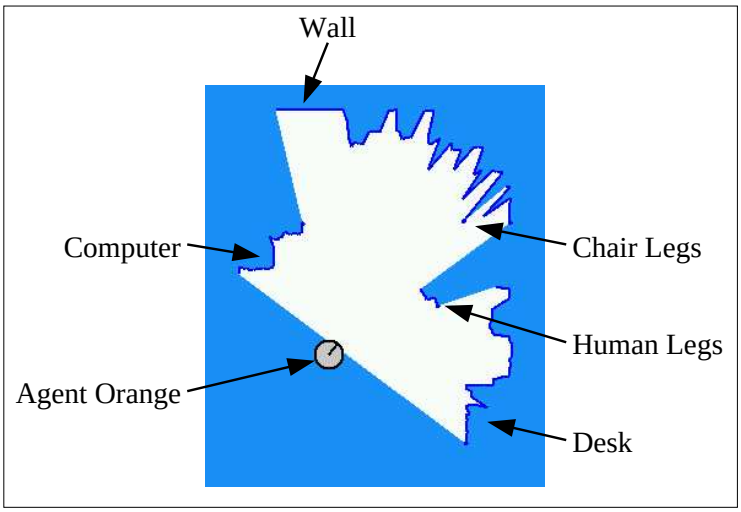


Figure 6.4: Sample laser scan.

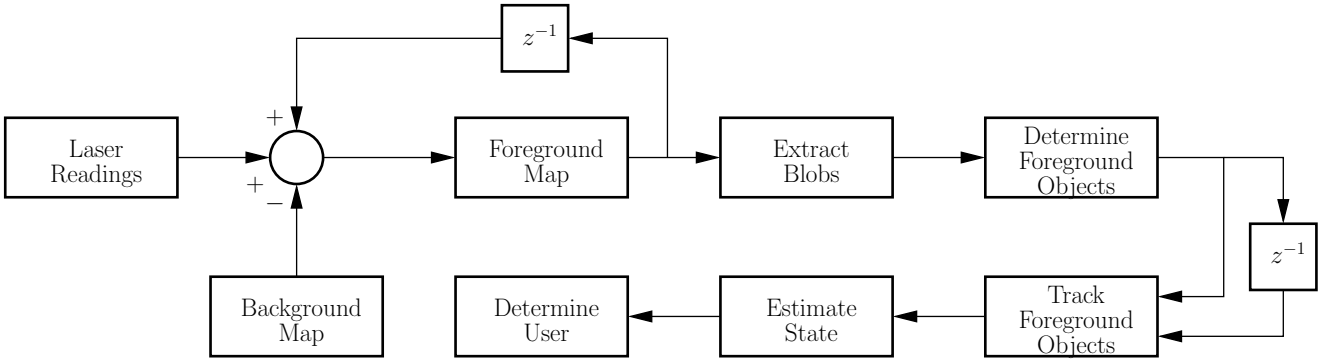


Figure 6.5: Flow diagram for observing users.

incident angle, for example, when the robot is nearly parallel to a wall.[2] While the laser provides extremely accurate measurements, it cannot discriminate between geometrically similar objects in the view plane, such as a the waist of a person and a trash can. Therefore Dollop must use extensive processing to extract meaningful information from these measurements, with an example shown in Figure 6.4. The method for processing laser readings, shown graphically in Figure 6.5, requires a background occupancy grid map of the workspace and is similar to the methodology of Fod et al. (2002). Objects detected by the laser are classified as belonging to one of two classes: background or foreground. Background objects are those that can be explained by the background occupancy grid map, while foreground objects are those that cannot. For the experiments in this work, background objects are things always found in the laboratory: desks, stools, computers, etc. Foreground objects consist of experiment-

[2]Certain round, shiny objects, like polished metal chair legs, can also fail to reflect the laser, yielding a false-negative reading.

Name	Meaning
Laser Readings	Robot-centric binary readings of the distance and bearing to objects
Background Occupancy Grid Map	Map of "empty" laboratory, with a discrete grid indicating the occupancy of each real-valued cell
Background Object	Laser readings that can be explained by the background occupancy grid map
Foreground Occupancy Grid Map	Map indicating the locations of laser readings not explained by the background occupancy grid map, with each real-valued cell updated in an autoregressive manner
Occupied Foreground Cell	Any cell in the foreground occupancy grid map with a value above a predefined threshold
Blob	Group of adjacent occupied foreground cells
Foreground Object	Any blob enveloping an area larger than a predefined threshold
User	The foreground object with the highest estimated velocity

Table 6.1: Terms used in the observation process.

specific objects, such as slalom cones and humans. A laser reading is designated as a *foreground cell* if there is no background object within some radius, e.g., 20 centimeters, of the measurement. Foreground cells are stored in a separate foreground occupancy grid map, with the cells updated in an autoregressive manner. A foreground cell is considered *occupied* if its value exceeds a predetermined threshold, and adjacent occupied foreground cells are called a *blob*. We determine the blobs by performing a breadth-first search on occupied foreground cells. A *foreground object* is a blob that envelops more than a required area, e.g., 100 square centimeters. The position of the foreground object is the centroid of the blob.

6.2.1 Foreground Object State Estimation

There appear to be two primary sources of error in determining the position of a foreground object. The first is due to the fact that line-of-sight sensing yields a much different blob centroid when viewing the same object from different perspectives, as in Figure 6.6. The second source of noise stems from positioning uncertainty. The MCL algorithm used by Carmen has an error of about one to two centimeters. Since laser readings are robot-centric, the position of a foreground object in a global-frame map requires the position of the robot in the global frame. Consequently, errors in the MCL robot position are propagated to foreground-object-position errors. However, for most objects that we consider in these experiments, blob-centroid errors (Figure 6.6) tend to dominate over MCL errors.

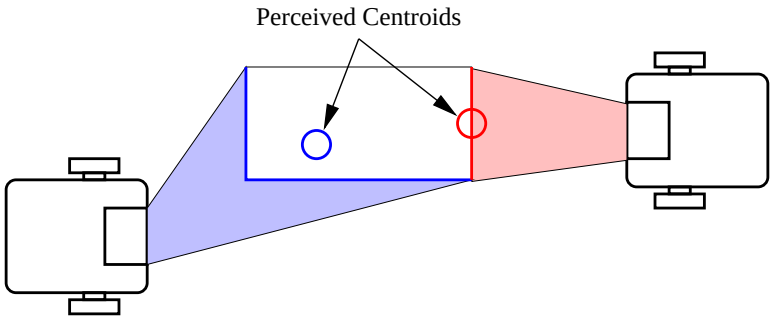


Figure 6.6: Viewing the same object from different perspectives causes the Dollop to compute different blob centroids.

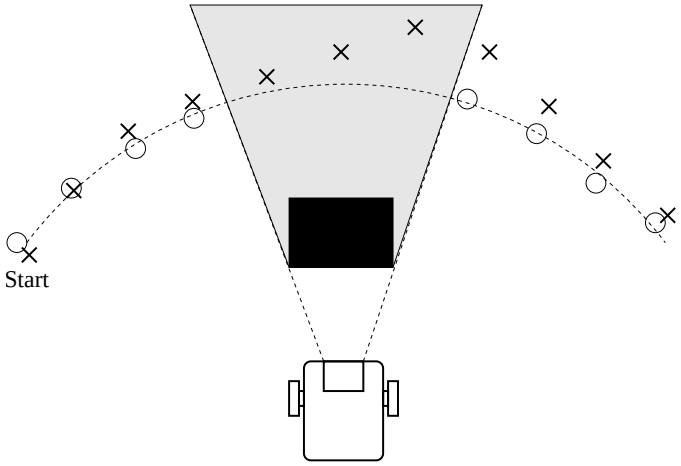


Figure 6.7: Hypothetical tracking by a Kalman filter of an object temporarily occluded by a rectangular obstacle. The object follows the dash arc path. The circles represent measurements of the object position, while the crosses mark the Kalman estimates of the object position. Since we do not incorporate estimates of acceleration, objects are assumed to move in straight lines while occluded.

We use a Kalman filter (Stengel, 1986) to estimate the position and velocity of a foreground object and to filter out sensor noise. Since Dollop must operate in relatively unstructured environments, it must be prepared to cope with losing sight of various objects of interest, including users. There are several attractive choices to cope with temporary object occlusion, including particle filters and Kalman predictions. In our experiments, Kalman predictions were more reliable in modeling occluded foreground-object motion. It is well known that estimates of the derivative of a signal amplify noise; incorporating these derivatives may cause system performance to decline. We found that incorporating an estimate of velocity, but not acceleration, generally produced the best performance in tracking occluded objects. This implies that our Kalman predictions of occluded objects will proceed in straight lines, as in Figure 6.7. The measurement noise covariance, $\boldsymbol{R}$, and state noise covariance, $\boldsymbol{Q}$, were determined experimentally in a lab. The Kalman gain matrix, $\boldsymbol{K}$, and the steady-state covariance, $\boldsymbol{V}$, are the solution to the

Riccati equation.

The laser can sense only the $\{x, y\}$ position of a foreground object, which is the blob centroid. We use a linear motion model, where the estimated state vector is $\widehat{\boldsymbol{b}}_n^i = [x\ y\ \dot{x}\ \dot{y}]^\mathsf{T}$, and the dynamical-system parameters are

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

where $\boldsymbol{A}$ is the system matrix and $\boldsymbol{C}$ is the output-selector matrix. The Kalman prediction of the foreground-object state at the subsequent time step is

$$\widehat{\boldsymbol{b}}_{n+1|n}^i \quad = \quad \boldsymbol{A}\widehat{\boldsymbol{b}}_n^i.$$

Let the sensed position of the foreground object at time $n+1$ be $\boldsymbol{C}\boldsymbol{b}_{n+1}^i$. The error between the sensed position and the prediction is defined as

$$\begin{aligned} \boldsymbol{\epsilon}_{n+1} &\triangleq \boldsymbol{C}\boldsymbol{b}_{n+1}^i - \boldsymbol{C}\widehat{\boldsymbol{b}}_{n+1|n}^i \\ &= \boldsymbol{C}(\boldsymbol{b}_{n+1}^i - \boldsymbol{A}\widehat{\boldsymbol{b}}_n^i). \end{aligned}$$

The Kalman gain matrix, $\boldsymbol{K}$, essentially determines how much to "believe" the sensor and how much to trust the prediction by influencing the estimated foreground-object state, given the previous estimated state *and* the sensed position of the foreground object,

$$\begin{aligned} \widehat{\boldsymbol{b}}_{n+1}^i &= \widehat{\boldsymbol{b}}_{n+1|n}^i + \boldsymbol{K}\boldsymbol{\epsilon}_{n+1} \\ &\doteq \boldsymbol{A}\widehat{\boldsymbol{b}}_n^i + \boldsymbol{K}\boldsymbol{C}\left(\boldsymbol{b}_{n+1}^i - \boldsymbol{A}\widehat{\boldsymbol{b}}_n^i\right). \end{aligned} \tag{6.1}$$

Coupling this estimate with the Kalman prediction covariance, $\boldsymbol{V}$, allows Dollop to compute the likelihood that a *sensed* foreground object, $\boldsymbol{C}\boldsymbol{b}_{n+1}^j$, is really a previously seen foreground object, $\widehat{\boldsymbol{b}}_n^i$,

$$\mathrm{p}\left(\boldsymbol{C}\boldsymbol{b}_{n+1}^j | \widehat{\boldsymbol{b}}_n^i\right) \quad \sim \quad \mathcal{N}(\boldsymbol{C}(\boldsymbol{b}_{n+1}^j - \boldsymbol{A}\widehat{\boldsymbol{b}}_n^i), \boldsymbol{V}).$$

This gives Dollop a probabilistic model of motion needed to track foreground objects through time.

6.2.2 Tracking Foreground Objects Through Time

In any practical situation, there will be multiple foreground objects in the workspace at any given time. For tracking purposes, it is necessary to match foreground objects at successive time steps. Let the set of *estimated* foreground-object states at time n be $\widehat{\boldsymbol{\mathcal{B}}}_n$ and the set of *sensed* foreground-object states at time $n+1$ be $\boldsymbol{\mathcal{B}}_{n+1}$. We would like to find the matching between the sensed foreground-object states at time $n+1$ and the estimated foreground-object states at time n. Our foreground-object matching is a simple greedy search for the maximum-likelihood match between foreground objects. The algorithm computes a matching function $\boldsymbol{f} : \boldsymbol{\mathcal{B}}_{n+1} \rightarrow \widehat{\boldsymbol{\mathcal{B}}}_n$ with the properties

- If the foreground object $\boldsymbol{b}^j_{n+1}$ is new, then $\boldsymbol{f}(\boldsymbol{b}^j_{n+1}) = \emptyset$.
- Otherwise, $\boldsymbol{f}(\boldsymbol{b}^j_{n+1}) = \widehat{\boldsymbol{b}}^j_n$ is the estimated state of the foreground object at the previous time step n.

If a Kalman prediction cannot be matched to a sufficiently close foreground object at time $n+1$, then the object is considered *occluded*. If a foreground object remains occluded for more than some predetermined amount of time, e.g., 5 seconds, then it is considered *disappeared*. The algorithm `greedy-foreground-object-match` is given in Figure 6.8. This greedy search for foreground-object tracking works well in environments that are relatively sparse. However, a joint optimization procedure may be more appropriate in workspaces with a large number of coinciding dynamic foreground objects. After matching foreground objects, we then estimate their velocities using the Kalman updates. For example, Equation 6.1 becomes

$$\widehat{\boldsymbol{b}}^j_{n+1} \quad = \quad \begin{cases} \boldsymbol{b}^j_{n+1}, & \boldsymbol{f}(\boldsymbol{b}^j_{n+1}) = \emptyset \\ \boldsymbol{A}\boldsymbol{f}(\boldsymbol{b}^j_{n+1}) + \boldsymbol{K}\boldsymbol{C}\left(\boldsymbol{b}^j_{n+1} - \boldsymbol{A}\boldsymbol{f}(\boldsymbol{b}^j_{n+1})\right), & \text{otherwise} \end{cases} .$$

We consider the user to be the foreground object with the highest estimated velocity,

$$\widehat{\boldsymbol{b}}^*_n = \arg \max_{\widehat{\boldsymbol{b}}^i_n \in \boldsymbol{\mathcal{B}}_n} \left\| \widehat{\boldsymbol{b}}^i_n - \boldsymbol{C}^\mathsf{T}\boldsymbol{C}\widehat{\boldsymbol{b}}^i_n \right\|_2 .$$

We then pass the estimated user state through a moving-average low-pass filter to remove jitter that is transmitted through the Kalman filter. The bank of filters for estimating the position and velocity of users is shown in Figure 6.9.

6.2.3 Tracking Example

We placed two obstacles, i.e., trash cans, about four meters apart in a laboratory, shown in Figure 6.10(a), and then asked a user to walk several iterations of a "figure-eight" trajectory while Agent Orange observed the user from a fixed location. We plot the results of tracking the user in Figure 6.10(b). Though the user is temporarily occluded

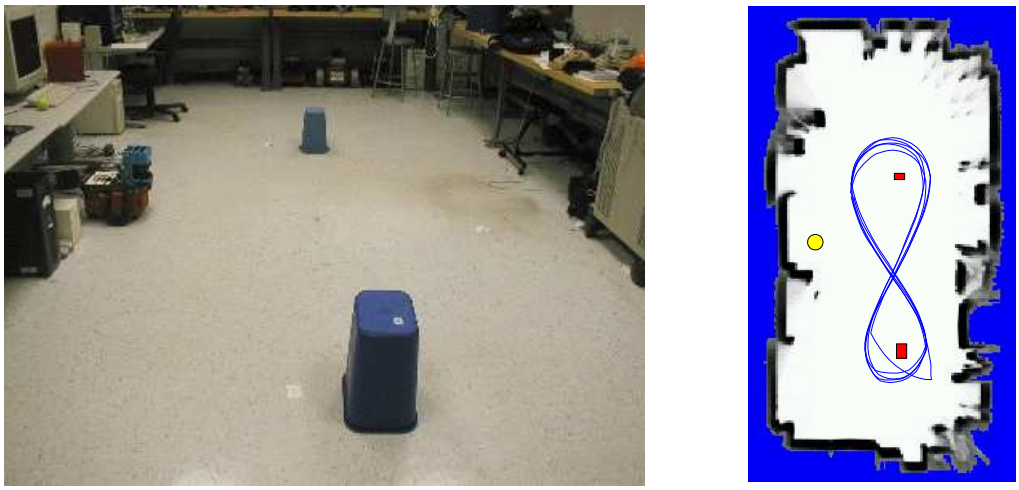
Algorithm `greedy-foreground-object-match`

$\boldsymbol{\mathcal{B}}_{n+1}$ is the set of sensed foreground objects at time $n+1$.
$\widehat{\boldsymbol{\mathcal{B}}}_n$ is the set of esimated foreground objects at time n.
$h_n : \boldsymbol{\mathcal{B}}_n \rightarrow \mathbb{Z}_{\geq 0}$ is time since the object was last sensed.
$h_{\max}$ is the timeout threshold for occluded objects.
τ is a probability threshold for matching.

```
 1: B^temp_{n+1} := B_{n+1}.
 2: for all b̂^i_n ∈ B̂_n
 3:    p*_i = max_{b^j_{n+1} ∈ B^temp_{n+1}} p(C b^i_{n+1} | b̂^i_n)
 4:    if p*_i ≥ τ then
 5:       b^j_{n+1} := arg max_{b^j_{n+1} ∈ B^temp_{n+1}} p(C b^i_{n+1} | b̂^i_n)
 6:       B^temp_{n+1} := B^temp_{n+1} \ {b^j_{n+1}}
 7:       h_{n+1}(b^j_{n+1}) := 0
 8:    end if
 9:    else then
10:       /* Object occluded. */
11:       b^j_{n+1} := A b̂^i_n
12:       B_{n+1} := B_{n+1} ⋃ {b^j_{n+1}}
13:       h_{n+1}(b^j_{n+1}) := h_n(b̂^i_n) + 1
14:    end else
15:    if h_n(b̂^i_n) < h_max then
16:       f(b^j_{n+1}) := b̂^i_n
17:    end if
18:    else then
19:       /* Object disappeared. */
20:    end else
21: end for all
22: for all b^j_{n+1} ∈ B^temp_{n+1}
23:    /* Any remaining objects are new. */
24:    f(b^j_{n+1}) := ∅
25:    h_{n+1}(b^j_{n+1}) := 0
26: end for all
27: return f, h_{n+1}
```

Figure 6.8: The greedy-matching algorithm for foreground-object tracking.

Figure 6.9: Bank of filters for estimating the position and velocity of users.



(a) (b)

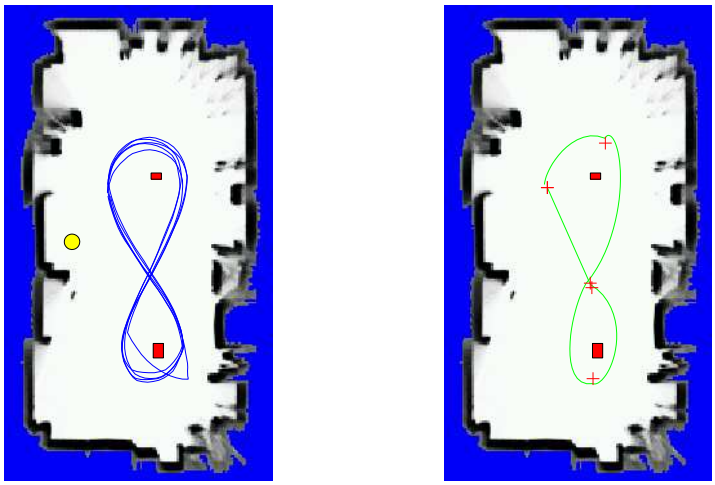
Figure 6.10: Tracking the user during five demonstrations of a "figure-eight" trajectory. The red boxes indicate the location of obstacles, and the yellow circle indicates the position of the robot while tracking the user.

while demonstrating the trajectory, the filters are able to interpolate the trajectory accurately.[3]

6.3 Describing User Trajectories

We automatically identify the subgoals in the trajectory using the sequenced Linear Dynamical System (LDS) representation from Chapter 5. Each trajectory segment between the subgoals is fitted by a single LDS that captures its salient features such as direction, curvature, and speed. This sequenced-LDS approach is particularly appealing for LBO applications since it represents trajectories in a *generative* fashion, encoding the trajectory with a stable control law induced by the LDS. That is, the generative representation provides a mapping from current state to desired state. In Figure 6.11(b) we extract subgoals and the LDS estimates from the trajectory in Figure 6.11(a). Qualitatively, this estimated trajectory appears to capture the intentions of the original trajectory: a

[3]During one demonstration the Kalman filter temporarily lost track of the user. This is manifested by the outlying line on the bottom-right side of Figure 6.10(b).

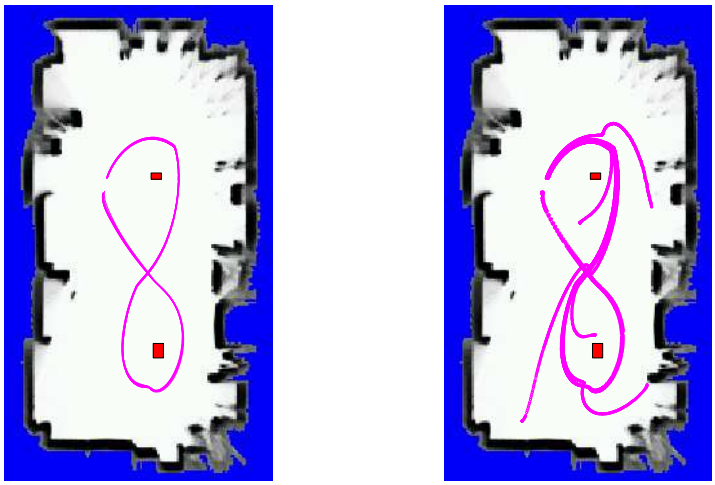


(a) (b)

Figure 6.11: Extracting the subgoals and the control law from the demonstrations in Figure 6.11(a). The green line in Figure 6.11(b) represents the control law the robot used to perform the task, based on the sequenced-LDS representation, and the crosses mark the location of subgoals.

"figure-eight" trajectory that avoids the obstacles. Quantitatively, the trajectory is represented by six LDS estimates and has an average error of 20 millimeters compared to the original trajectory. However, the LDS representation does not respect the dynamics of Agent Orange. The sequenced-LDS representation is optimal for idealized systems (cf. Theorem 5.1), but we make no guarantees for nonholonomic, underactuated, or saturating systems. In Figure 6.12(a) we plot the response of Agent Orange during ten runs of the estimated task. While it does not exactly follow the idealized trajectory in Figure 6.11(a), the robot does come fairly close to the predicted path. In Figure 6.12(b) we plot the response on five runs during which we "kidnap" Agent Orange during execution and plot its response to these perturbed conditions.[4] Because we represent the trajectory in a generative manner, using a control law induced from the LDS, Dollop automatically interpolates its response to being kidnapped. It is coincidental that none of the responses to the kidnapping resulting in Agent Orange hitting the obstacles, or laboratory walls, since the response of the control laws only depends on the current state of the robot and is independent of objects in the environment. However, the response of Agent Orange to being kidnapped seems to be a reasonable estimate of "what the user would have done."

[4]During "kidnapping," we simply stop the LBO control locus and move the robot with a joystick to a new location. This allows the MCL to keep Agent Orange globally localized. This is different from what Thrun (2001) refers to as "kidnapping," where the robot loses its localization.



(a) (b)

Figure 6.12: Figure 6.12(a) shows the result of ten runs of Agent Orange performing the estimated trajectory of Figure 6.11(a). In Figure 6.12(b) we show the response of Agent Orange after being "kidnapped" during five executions of the trajectory.

6.4 Environment Configuration

Since demonstrations may be performed in different environments, an LBO system must have the ability to determine how the actions of users are affected by these changes.

6.4.1 Determining Environment Configuration

In Dollop, the environment configuration is the set of static foreground objects in a workspace. To determine the environment configuration, Dollop extracts the set of static foreground objects from the foreground occupancy grid described in Section 6.2. During the course of its operation of tracking the user, if the laser scans a static object (e.g., a slalom cone), then the foreground occupancy grid will contain readings at the location of that object. Unless the object moves and a subsequent laser scan reveals the previous location empty, at the end of operation Dollop will determine that there is a static foreground object at its location. Dollop then places a bounding box around each static foreground object. The environment configuration is then defined to be the set of vertices of the bounding boxes enveloping each static foreground object. If there are N static foreground objects, then the environment configuration would contain $4N$ locations, one for each corner of the bounding boxes. If the ith

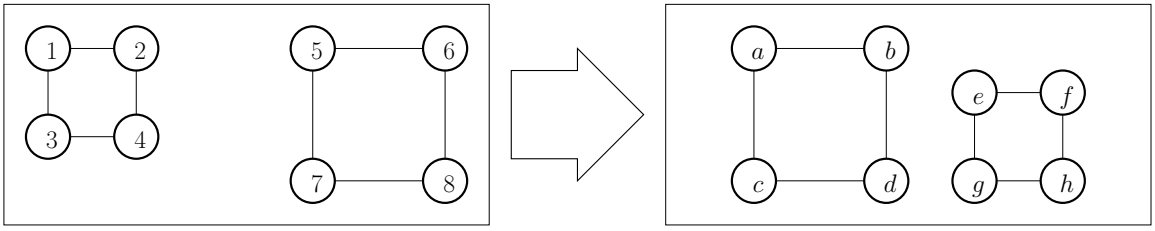


Figure 6.13: The eight objects on the left must be matched to those on the right.

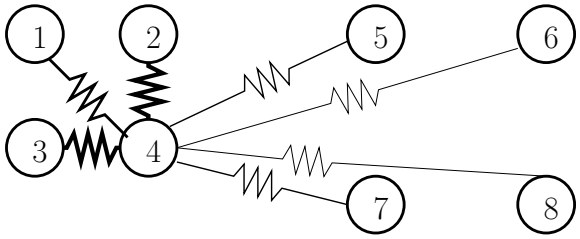


Figure 6.14: Intuitive representation of the "spring" optimization.

demonstration of a task contains N static foreground objects, then the environment configuration is

$$\boldsymbol{\Omega}^i \quad = \quad \{\boldsymbol{\omega}^i_1, \ldots, \boldsymbol{\omega}^i_{4N}\},$$

where $\boldsymbol{\omega}^i_{4(n-1)+2}$ contains the $\{x, y\}$ coordinates for the 2nd corner of the bounding box of the nth static foreground object.

6.4.2 Mapping Environment Configurations

A crucial step in learning from, and performing, tasks demonstrated in different environment configurations is determining how different environment configurations relate to each other. For example, in Figure 6.13 each object on the left, $\{1, 2, 3, 4, 5, 6, 7, 8\}$, must be matched to one on the right, $\{a, b, c, d, e, f, g, h\}$. There are many possible solutions, and any objective function giving preference to one possible mapping over another is implicitly assuming that some mappings are more likely than others. We formulate the matching problem by considering each object connected to all others by "springs," with stronger springs connected to closer objects, as in Figure 6.14. These springs result in a "force" on the object. Matching one object with another results in a change in force and we compute the matching that minimizes the *total* change in force on all objects.

Let the environment configuration of the ith demonstration be given by $\boldsymbol{\Omega}^i = \{\boldsymbol{\omega}^i_1, \ldots, \boldsymbol{\omega}^i_{4N}\}$ and let the environment configuration of the jth demonstration be given by $\boldsymbol{\Omega}^j = \{\boldsymbol{\omega}^j_1, \ldots, \boldsymbol{\omega}^j_{4N}\}$. Formally, we are computing the function $\boldsymbol{g} : \boldsymbol{\Omega}^i \rightarrow \boldsymbol{\Omega}^j$ that minimizes the cost function $c : \boldsymbol{\Omega}^i \times \boldsymbol{g}(\boldsymbol{\Omega}^i) \rightarrow \mathbb{Z}$ summed over all objects. Let the

spring constant between object $\boldsymbol{\omega}_m^i$ and object $\boldsymbol{\omega}_n^i$ be

$$k_{m,n}^i \triangleq \left\|\boldsymbol{\omega}_m^i - \boldsymbol{\omega}_n^i\right\|^{-2}.$$

The total force on object $\boldsymbol{\omega}_m^i$ is

$$\boldsymbol{f}_m^i \triangleq \sum_{\substack{\boldsymbol{\omega}_n^i \in \boldsymbol{\Omega}^i \\ \boldsymbol{\omega}_n^i \neq \boldsymbol{\omega}_m^i}} k_{m,n}^i \frac{\boldsymbol{\omega}_n^i - \boldsymbol{\omega}_m^i}{\|\boldsymbol{\omega}_n^i - \boldsymbol{\omega}_m^i\|}.$$

If object $\boldsymbol{\omega}_m^i$ is matched with object $\boldsymbol{\omega}_q^j$, then the magnitude change in force caused by this individual match is

$$c(\boldsymbol{\omega}_m^i, \boldsymbol{\omega}_q^j) \triangleq \left\lceil \alpha \left\|\boldsymbol{f}_m^i - \boldsymbol{f}_q^j\right\| \right\rceil,$$

where $\lceil b \rceil$ is the ceiling operation on b, which rounds up $b \in \mathbb{R}$ to the next greatest integer, and α is a large constant, e.g., 10^3, to alleviate numerical rounding. It is straightforward to cost this formulation as a weighted bipartite-graph matching problem known as the Hungarian Method (Papadimitriou & Steiglitz, 1998). This formulation can be solved optimally by linear programming; in practice, a large number of environment objects can matched in well under a second. However, the Hungarian Method requires that the matching function, $\boldsymbol{g}(\cdot)$, be *bijective*. This means that the number of environment objects must be the same in all demonstrations of a task. In our experiments this has not shown itself as a problem, but, in general, this is a restrictive assumption. Fortunately, there are many sophisticated, and very complex, matching formulations with more realistic assumptions (Cheriyan, 1997).

6.4.3 Associating Subgoals with Environment Objects

To learn from demonstrations performed in different environments, Dollop hypothesizes about what the user would have done had the demonstrations been performed under the same conditions. This is accomplished by associating subgoals with environment objects, a subject of previous investigations of LBO methods. Morrow and Khosla (1995), for instance, extracted the corners of objects in the workspace from camera images. In that work, users were then required to supply the location of the subgoals and manually associate them with the corners. As objects in the workspace moved, a corner-tracking algorithm updated the subgoal locations accordingly. Our method for associating subgoals is conceptually similar, but performed automatically. First, Dollop determines the closest m environment objects to a subgoal. The difference between the subgoal and these environment objects is weighted so that closer environment objects have higher weights. These weighted differences are the *subgoal associations*. Given a mapping between the environment objects in two different configurations, $\boldsymbol{g} : \boldsymbol{\Omega}^i \rightarrow \boldsymbol{\Omega}^j$, Dollop uses the mapped location of the environment objects, $\boldsymbol{g}(\boldsymbol{\omega}_l^i)$, and the environment association to reconstruct the position of the subgoal in the new environment configuration. Essentially, this forms the hypothesis about where the user

would have located the subgoals in the new environment.

Let the closest m environment objects to the subgoal $\boldsymbol{x}_h^i$ in the environment configuration $\boldsymbol{\Omega}^i$ be $\boldsymbol{A}_{\boldsymbol{x}_h^i}^{\boldsymbol{\Omega}^i}$. For each environment object in the set $\boldsymbol{\omega}_l^i \in \boldsymbol{A}_{\boldsymbol{x}_h^i}^{\boldsymbol{\Omega}^i}$, compute the difference $\boldsymbol{\varpi}_l = \boldsymbol{x}_h^i - \boldsymbol{\omega}_l^i$. Let the weight of the association be $w_l = \|\boldsymbol{\varpi}_l\|^{-1} / \sum_k \|\boldsymbol{\varpi}_k\|^{-1}$. Dollop reconstructs the position of subgoal $\boldsymbol{x}_h^i$ in the new environment configuration as

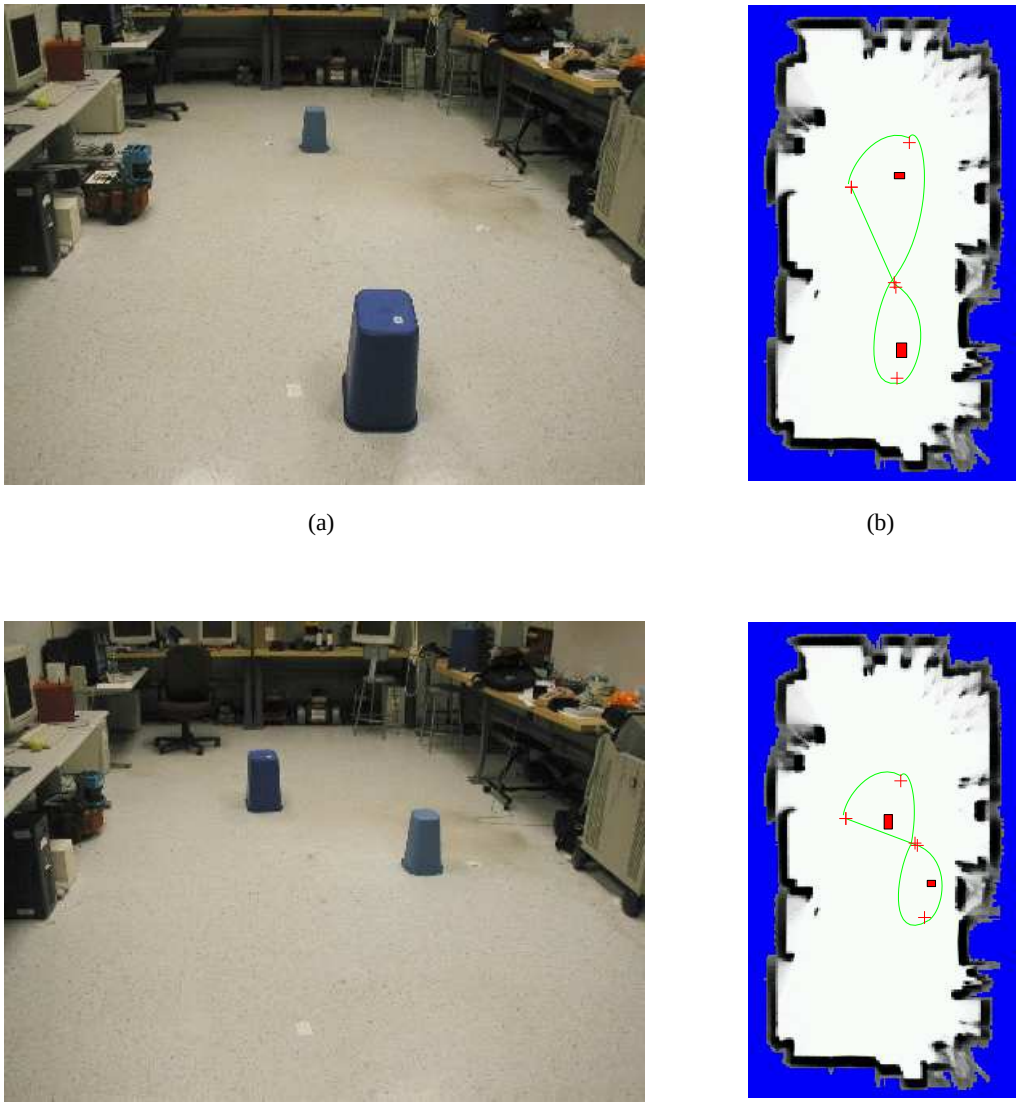
$$\boldsymbol{x}_h^j \;=\; \sum_{\boldsymbol{\omega}_l^i \in \boldsymbol{A}_{\boldsymbol{x}_h^i}^{\boldsymbol{\Omega}^i}} w_l \left(\boldsymbol{\varpi}_l + \boldsymbol{g}(\boldsymbol{\omega}_l^i) \right).$$

6.4.4 Environment Mapping Example

In Figure 6.15 we moved the obstacles from the original "figure-eight" demonstration (Figure 6.11(b)) to different locations in the laboratory. Because the subgoals are associated with the obstacles, their locations update accordingly. Furthermore, the subgoals make modifying the task trivial: a trajectory can be scaled or shifted by simply moving the subgoals and allowing the LDS to interpolate its response. The locations of the subgoals in the modified environment and the corresponding response of each LDS form the hypothesis of user intent. Independently of the hypothesis, we asked the user to demonstrate another figure-eight trajectory in the modified environment, shown in Figure 6.16. Quantitatively, the average error of the hypothesized trajectory to what the user actually performed was about 200 millimeters. Qualitatively, Dollop succeeded in determining what the user would have done: creating a "figure-eight" trajectory that avoids the obstacles.

6.5 Learning

As mentioned in previous chapters, we model the user as generating subgoals according to a Continuous-Density Hidden Markov Model (CDHMM). Since we do not know the set of tasks that the user will demonstrate *a priori*, we must estimate the *structure* of the CDHMM from user demonstrations alone. To accomplish this, we use the learning algorithm derived in Chapter 3. When applied to Dollop, this algorithm analyzes the sequences of subgoals from different task demonstrations. If the algorithm finds similarities between the subgoals of different demonstrations, then the CDHMM is simplified by combining the similarities. This merging process gives an increasingly accurate estimate about where the user intended the subgoals to be located. Since the sequenced-LDS approach is based on least-squares estimation, multiple demonstrations of a trajectory can be incorporated to compute a more accurate generalization of the indented actions of the user. Consequently, upon finding similarities in different demonstrations, Dollop estimates a single LDS that describes *both* segments, which improves the generalization ability of the estimate, as in Section 5.5. Both of these attributes mean that Dollop is able to reproduce



(a) (b)

(c) (d)

Figure 6.15: Moving the slalom cones causes the subgoals to modify their locations automatically due to the subgoal associations. The control laws for each trajectory segment automatically adapt to "what the user would have done."

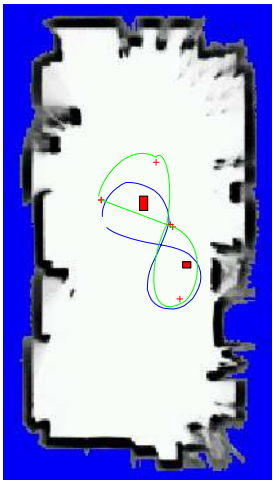


Figure 6.16: We asked the user to perform a figure-eight path in the modified environment, and the resulting trajectory observed by the robot, with our hypothesized trajectory in green. Quantitatively, the average error of the hypothesized trajectory was about 200 *millimeters.*

and generalize user actions if it has access to multiple demonstrations performed under different environment configurations. This gives Dollop a better idea of how the user would perform the task in varying conditions.

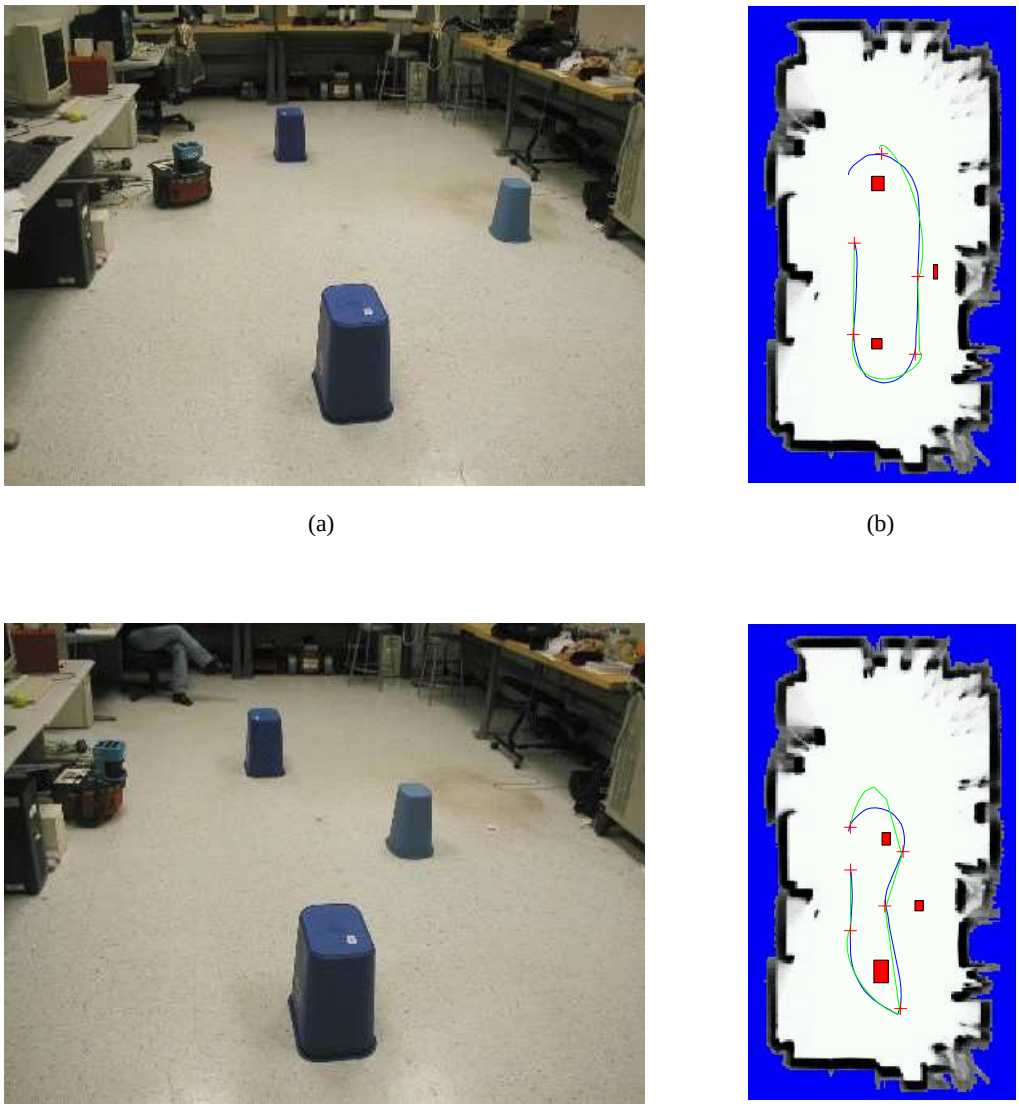
In Figure 6.17 we show two demonstrations of a task performed in different environments, along with the subgoals and trajectories estimated by Dollop. Before learning occurs, the subgoals from both demonstrations are mapped to the same environment configuration. Dollop then estimates a CDHMM that describes the subgoals in the demonstrations. We compute the most likely sequence of subgoals needed to complete the task, using the unconditional Viterbi path. The Viterbi path is typically defined as the maximum-likelihood sequence of states conditioned on an observation sequence (Rabiner & Juang, 1993). However, we are computing the unconditional Viterbi path, the most likely sequence of states through the CDHMM, which is equivalent to finding the conditional Viterbi path averaged over *all* possible observation sequences. Define the unconditional Viterbi variable as

$$\delta_n(q_j) \triangleq \begin{cases} \max_{q_i \in \mathcal{Q}} a_{j|i} \delta_{n-1}(q_i), & n > 0 \\ \pi_j, & n = 0 \end{cases}.$$

We keep track of the state sequences using back-pointers,

$$\psi_n(q_j) \triangleq \arg\max_{q_i \in \mathcal{Q}} a_{j|i} \delta_{n-1}(q_i),$$

when $n > 0$. We slightly modify our previous definition of a CDHMM (Section 3.4) to include termination states,

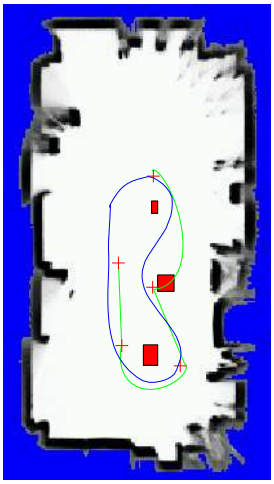


(a) (b)

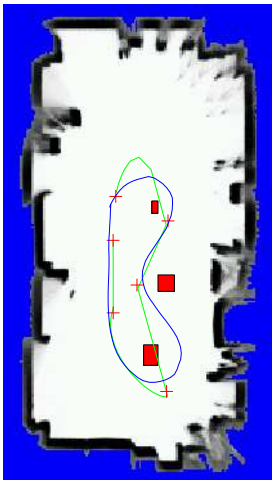
(c) (d)

Figure 6.17: Two demonstrations of a task in modified environments and the corresponding estimated trajectories.

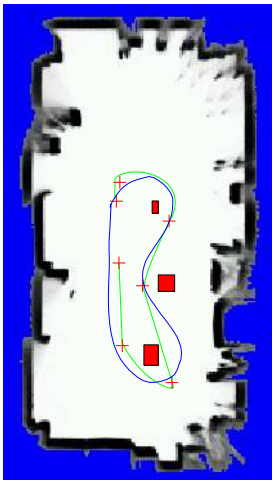
(a)



(b)



(c)



(d)

Figure 6.18: Mapping the demonstrations from Figure 6.17(b) and Figure 6.17(d) to the modified environment in Figure 6.18(a). Individually, the average error is 364 *(Figure 6.18(b)) and* 236 *millimeters (Figure 6.18(c)) respectively. When Dollop learns from both demonstrations, the average error is* 189 *millimeters (Figure 6.18(d)).*

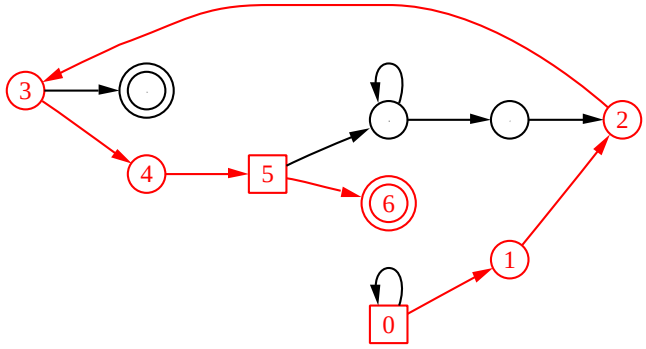


Figure 6.19: CDHMM estimated from the two trajectories in Figure 6.17, with the most likely sequence of subgoals shown in red.

and this set is written as $\mathcal{Q}_\lambda$. We then search for the most likely state sequence finishing at a termination state within a specified time window, $0 < N_{\min} \leq N_{\max}$,

$$(q_*, N_*) \quad = \quad \arg \max_{\substack{q_i \in \mathcal{Q}_\lambda \\ n \in \{N_{\min}, \cdots, N_{\max}\}}} \delta_n(i).$$

We use the back-pointers to reconstruct the Viterbi path from the initial conditions $n = N_*$ and $q_{N_*} = q_*$,

$$q_{n-1} \quad = \quad \psi_n(q_n),$$

for $n = N_*, \ldots, 1$. For the CDHMM estimated from the demonstrations in Figure 6.17 we show the Viterbi path in Figure 6.19.

In Figure 6.18 we show how the hypotheses estimated from the demonstrations in Figure 6.17 are modified when presented with the new environment configuration, in Figure 6.18(a). Independently, we also asked the user to demonstrate a slalom trajectory through the three obstacles. The average errors of the hypotheses are 364 and 236 millimeters, respectively, when learning only from a single demonstration. However, when Dollop learns from *both* demonstrations and maps its hypothesis to the modified environment, the average error is 189 millimeters. This implies that giving Dollop the opportunity to learn from both demonstrations allowed it to form a more accurate hypothesis of user intent than either demonstration individually.

6.6 Summary

We have presented a system that pursues a computational approach to Learning By Observation, called Dollop. Our approach allows Dollop to learn motor-skill tasks by observing user demonstrations with its scanning laser

range finder. From these demonstrations, subgoals are extracted and associated with objects in the environment, so that, as these objects move, the subgoals update accordingly. We described how Dollop can learn from multiple demonstrations in different environments to infer the intentions of the user. We also showed examples of Dollop learning from demonstrations in a laboratory setting with a mobile robot.

Chapter 7

Conclusions

In this chapter, we summarize our work, restate the contributions, and point out limitations and possible future directions of this work.

7.1 Summary

In this dissertation, we have presented an approach to inferring user intent for learning motor-skill tasks by observation. We began by decomposing the complex LBO problem and analyzing the simpler components. We have modeled the process by which users create task subgoals by a Continuous-Density Hidden Markov Model. Since the LBO system does not know the tasks users will perform *a priori*, we derived a learning algorithm that estimates the structure of CDHMMs from observations of user demonstrations alone. From a theoretical analysis, our learning algorithm produces CDHMMs with a locally minimal number of states, with worst-case computational complexity quadratic in the number of observations. Assuming users behave as a CDHMM, we showed that the probability of correctly estimating the states in the target CDHMM increases asymptotically as more demonstrations are incorporated. We analyzed the performance of the learning algorithm on complex, real-world tasks with an application of predictive robot programming. By isolating the performance from sensor noise and environment considerations, we showed that our model is able to predict task subgoals with a high degree of accuracy.

We then presented a method for hypothesizing about the response of users to different conditions with sequenced linear dynamical systems. This method encodes user actions by first segmenting a demonstration at important points in the trajectory, the subgoals; each segment is represented by a single LDS that induces a control law forming a hypothesis about the response of users as the subgoals or conditions change. Multiple demonstrations can be incorporated to improve the generalization of the system and infer user intent. We also provided proofs of optimality and stability for the sequenced-LDS representation.

Finally, we described our LBO system, Dollop, which learns to automate motor-skill tasks by observing users with a laser while they demonstrate tasks. User demonstrations are segmented using the sequenced-LDS approach and the resulting subgoals are automatically associated with objects in the environment. As these objects move, the subgoals are updated accordingly, and each LDS automatically adapts its response to these new conditions. Multiple demonstrations of a task can be incorporated even demonstrations performed in different environment configurations allowing Dollop to infer user intent. To learn from tasks demonstrated in different environments, demonstrations are first mapped to a common environment; then Dollop hypothesizes about what the user would have done had the demonstrations been performed under the same conditions. Finally, Dollop estimates a CDHMM that describes the subgoals from the various demonstrations and computes the most likely sequence of subgoals needed to complete the task.

7.2 Contributions

The main contributions of this dissertation are: [1]

- A **CDHMM structure-estimation algorithm** (Dixon et al., 2004). This algorithm is designed to cope with the characteristic difficulties of LBO: scarce training data and real-time operation. The algorithm can operate in an online fashion, as observations become available, or as batch processing, on a complete data set. We provide a theoretical analysis showing the strengths and weaknesses of the algorithm.

- Design and analysis of a **Predictive Robot Programming** system (Dixon & Khosla, 2003). In addition to isolating variables to study LBO performance, the application of PRP is worthwhile in its own right to decrease manipulator-robot programming time. We show that the PRP system has a median prediction error less than 0.5% of the distance traveled during prediction on a set of data from complex, real-world robotic tasks. We also present laboratory experiments showing that the PRP system results in a significant reduction in programming time, with users completing simple robot-programming tasks over 30% faster when allowing the PRP system to compute predictions of future positions.

- Trajectory representation using **Sequenced Linear Dynamical Systems** (Dixon & Khosla, 2004b). We have derived a novel method to represent hypotheses of user actions by estimating a sequence of linear dynamical systems that describes a demonstration. We show optimality of the formulation and provide a proof of stability of the induced supervisory control law.

- Development of a computational approach to **Learning By Observation** (Dixon & Khosla, 2004a). We

[1] Repeated from Section 1.4

leverage the analyses from the previous contributions to develop an LBO system that learns motor-skill tasks from user demonstrations. Multiple demonstrations can be incorporated to improve system performance, and demonstrations can be performed in different environment configurations. We present laboratory experiments showing that the LBO system accurately infers user intent.

7.3 Conclusions

7.3.1 Structure Estimation

Limitations

Our algorithm was created for instances where there exist short-sequence similarities, which prompted us to derive a memoryless merging criterion. A byproduct of our derivation is that the learning algorithm will not represent finite-looping tasks well. In domains with long-sequences similarities or finite looping, such as speech and gesture recognition or some types of assembly tasks, then we suspect that our algorithm will not perform as well as other algorithms derived for those applications, e.g., Ron et al. (1998) and Stolcke and Omohundro (1994b).

Our application domains require real-time use and we do not typically have large amounts of training data. This means that our algorithm must provide an answer as quickly as possible from a heavily biased space. For practical purposes, this prevents us from deriving an asymptotic convergence guarantee, such as those described by Ephraim and Merhav (2002).

Future Directions

We would like to derive a proof guaranteeing a negative correlation between prediction confidence and prediction accuracy, although we have been unsuccessful so far. One possibility for future use of graph-based structure estimation is in speculative precomputation (Collins et al., 2001) in computer-processor design. These systems attempt to predict memory-page faults based on the previous behavior of a computer program.

Given the severe theoretical limitations for structure estimation in graph-based models (Section 2.4), it appears unlikely that any one-size-fits-all algorithm will emerge in the near future. One possibility to broaden the appeal of a particular structure-estimation approach is by using anytime algorithms (Zilberstein & Russell, 1996). These algorithms improve their performance as their alloted time increases.

Currently, system designers needing structure-estimation algorithms in their applications should first delineate the requirements. For example, are acyclic models acceptable or are cyclic models needed? What computational complexity should the algorithm be allowed? How much training data will the algorithm be given? Is it possible

to discretize observations? Answers to these questions should lead future researchers to existing algorithms, or shed light on the attributes needed in the structure-estimation algorithm.

7.3.2 Predictive Robot Programming

Limitations

Because the PRP system does not consider the environment when computing a prediction, collisions between the robot and various objects do occur. While these collisions are not problematic for most industrial manipulators, there may be fragile objects in the workspace, including the user, that might be damaged in a collision. As such, any PRP system should avoid suggesting waypoints that result in a collision, and, for safety reasons, the motion of the PRP system must be easily stopped by the user.

Currently, the PRP system does not indicate the position of a predicted waypoint to users before moving the robot. There does not seem to be an efficient solution that allows users to visualize *a priori* where the PRP system intends to move the robot. One potential solution is to show users with a virtual-reality model (i.e., offline system) where the predicted waypoint is, allowing users to determine if the prediction is appropriate. But this implementation may cause users to spend more time determining if the waypoint is acceptable than if they simply allowed the PRP system to move the robot automatically and (potentially) undid it.

Because PRP does not have a well-defined domain, such as a dictionary for word-completion tasks, it may require a substantial amount of effort to build a sufficient repertoire of examples to predict waypoints reliably. It would be helpful to users to "seed" the CDHMM with generic movements that tend to be found in various tasks. This might help the PRP system predict waypoints sooner.

In our PRP implementation, all parameters are fixed *a priori*, such as the CDHMM complexity, δ, and the prediction-confidence threshold, $\phi_{\min}$. These parameters should certainly be adapted online, based on feedback from the user.

Future Directions

Currently, the PRP system identifies similarity to previous subtasks using a single, monolithic CDHMM. It may be helpful to allow users to provide the PRP system with labeled tasks. The PRP system could then create many smaller CDHMMs that describe these patterns. In this scenario, users would demonstrate a task and indicate that the PRP system should create a new CDHMM describing that task. In the future, the PRP system would then decide which CDHMM best describes recently created waypoints and use that model to predict the next waypoint. After completing the task, the PRP system could improve its ability to recognize and predict these patterns by

incorporating semi-supervised learning techniques (Seeger, 2000). This would give users more input about how the PRP system assists them.

7.3.3 Hypotheses of User Actions

Limitations

The proof of stability for the sequenced-LDS approach requires that the observations not be collinear or, in the high-dimension case, that the observations do not lie on a hyperplane. As the observation dimension is increased, it becomes more likely that there will be a hyperplane on which all the observations lie. For example, suppose the observation vector includes a velocity magnitude. In this case, we do not guarantee the stability of LDS estimates from constant-velocity trajectories, regardless of the position components.

We assume that observations are generated according to a type of linear, time-invariant, constant-input dynamical system. We fit a parametric model to the given observations that minimizes the squared one-step prediction error. This results in two artifacts. The first is that a discrete-time LDS assumes a geometric progression of the state variables. Thus, if the user demonstrates a trajectory with equally spaced observations, our approach will still attempt to fit a least-squares geometric series to the data, which may be undesirable. The second artifact from our formulation is that we only guarantee optimal one-step predictions. A user of the sequenced-LDS approach may be more interested in minimizing the error between the demonstrated trajectory and the estimated trajectory, which appears to be a significantly more difficult formulation.

Unlike cubic-spline interpolation, our sequenced-LDS approach does not have a smoothness criterion between segments, though one does seem possible. Consequently, at the boundary of an LDS segment, the supervisory control law could send large reference-error commands to the low-level controller, potentially resulting in high-jerk motion.

Future Directions

We would like to formulate an offline-segmentation scheme where the input is an allowable error and the output is a segmentation that achieves no worse than that error, while using the fewest parameters possible. Also, we would like to derive a smoothness criterion between LDS segments, for minimum-jerk or minimum-acceleration profiles.

Other potential applications of this approach include modeling how efficient a new room design may be. By observing how a user moves between, e.g., a computer, desk, and book case, we can design more efficient offices for people by hypothesizing about their movements to a novel configuration. It should be simple to extend this idea to design more efficient factories and office buildings prior to investing time and money in physically rearranging

the space.

7.3.4 Learning By Observation

Limitations

The representation of the environment is probably the weakest point in Dollop. While we can compute environment matchings quickly and consistently, it is an inherently ill-posed problem. The geometric information obtained from laser readings disregards many cues that could be helpful in disambiguating the matching process. As such, it is not difficult to contrive environments that result in counter-intuitive matchings. Coupling laser readings with a more expressive sensor, such as a camera, could result in improved performance. Before being used in more realistic environments, Dollop will have to incorporate a more sophisticated environment-matching algorithm, since the current formulation, the Hungarian Method, requires that the same objects be present in each demonstration. We would expect this robust matching procedure to be iterative and heuristic in nature, based on the EM algorithm.

The supervisory control laws from the sequenced-LDS approach do not respect environment factors, such as object collisions. As such, it is possible that certain environment configurations will result in the estimated supervisory control law directing the robot into a collision.

Also, the CDHMM learning procedure is sensitive to the segmentation derived from the sequenced-LDS approach. If dramatically different segmentations are produced for different demonstrations, then the learning algorithm will not find similarities in the different demonstrations. This could result in unpredictable generalizations to novel environment configurations. In our experiments, this has not shown itself to be a problem, since the LDS segmentation appears to identify important locations in trajectories reliably.[2]

Future Directions

Similar to PRP, the parameters of the LBO system should be adapted based on user feedback. Currently, all parameters are fixed *a priori*, such as the CDHMM complexity, δ, and the number of environment objects associated with a subgoal. In particular, we would like to formulate the environment-object association problem adaptively by incorporating information from multiple demonstrations. This would allow Dollop to learn about which environment objects are important to a particular task.

In the future, we plan on extending Dollop to encode higher-level knowledge, giving it the ability to learn

[2] In fact, in results not included in this dissertation, the automatic segmentation was more consistent – and performed better – than a manual segmentation.

more complex tasks. Dynamical systems can represent motor-skill tasks efficiently, but they may not be the best approach for learning more sophisticated skills. We are interested in pursuing a hierarchical representation, with the motor-skill-learning ability at the lowest level, and some as-yet-undefined method encoding high-level knowledge.

7.4 List of Parameters

This section lists the parameters, as well as the typical values used, in this work.

Parameter	**Range**	**Typical Value**	**Description**	**Described in**
$\phi_{\min}$	$[0, 1]$	0.7	Prediction-confidence threshold	Section 3.11
δ	$(0, 1]$	0.7	Complexity of the CDHMM estimated by the learning algorithm	Section 3.12
ζ	$(0, \infty)$	0.2meters	Stopping criterion for sequenced LDS estimates	Section 5.4
$\kappa_{\max}$	$[1, \infty)$	10	Maximum condition number of LDS matrix estimate $\widehat{\boldsymbol{R}}$ before declaring trajectory collinear	Section 5.6
$\varepsilon_{\max}$	$[0, \infty)$	0.7	Normalized-prediction-error threshold for trajectory segmentation	Section 5.7
$h_{\max}$	$[0, \infty)$	5sec	Timeout threshold for occluded objects	Section 6.2.2
τ	$[0, 1]$	0.5	Probability for foreground-object matching	Section 6.2.2
$\lvert \boldsymbol{A}^{\boldsymbol{\Omega}^i}_{\boldsymbol{x}^i_h} \rvert$	$[0, \infty)$	5	Number of environment objects to associate with a sub-goal	Section 6.4.3

Appendix A

Technical Details

This chapter contains the proofs of various propositions and lengthy derivations.

A.1 Proofs

A.1.1 Proof of Lemma 3.1

Proof. Equation 3.1, μ, was defined using any symmetric PD precision matrix $\boldsymbol{C} \in \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}}$ and any reference vector $\boldsymbol{u} \in \boldsymbol{\mathcal{X}}$. A measure on the multiset, $\boldsymbol{\mathcal{A}}$, is defined by the properties *nonnegativity* and *countable subadditivity*,

$$\begin{aligned} \mu_{\boldsymbol{C}}(\widetilde{\boldsymbol{\mathcal{A}}}, \boldsymbol{u}) &\geq 0, \forall \widetilde{\boldsymbol{\mathcal{A}}} \subseteq \boldsymbol{\mathcal{A}} && \text{(A.1)} \\ \mu_{\boldsymbol{C}}(\bigcup_n \widetilde{\boldsymbol{\mathcal{A}}}_n, \boldsymbol{u}) &= \sum_n \mu_{\boldsymbol{C}}(\widetilde{\boldsymbol{\mathcal{A}}}_n, \boldsymbol{u}), \forall \widetilde{\boldsymbol{\mathcal{A}}}_n \subseteq \boldsymbol{\mathcal{A}} \text{ and finite } n. && \text{(A.2)} \end{aligned}$$

We begin by showing the nonnegativity (Equation A.1) of our function. For any $\widetilde{\boldsymbol{\mathcal{A}}} \subseteq \boldsymbol{\mathcal{A}}$,

$$\begin{aligned} \mu_{\boldsymbol{C}}(\widetilde{\boldsymbol{\mathcal{A}}}, \boldsymbol{u}) &\triangleq \sum_{\boldsymbol{x} \in \widetilde{\boldsymbol{\mathcal{A}}}} \|\boldsymbol{x} - \boldsymbol{u}\|_{\boldsymbol{C}}^2 \\ &= \sum_{\boldsymbol{x} \in \widetilde{\boldsymbol{\mathcal{A}}}} (\boldsymbol{x} - \boldsymbol{u})^{\mathsf{T}} \boldsymbol{C} (\boldsymbol{x} - \boldsymbol{u}) \\ &\geq 0, \end{aligned}$$

by definition of PD $\boldsymbol{C}$. Note that $\|\boldsymbol{x} - \boldsymbol{u}\|_{\boldsymbol{C}}^2 = 0$ if and only if $\boldsymbol{x} = \boldsymbol{u}$. For any finite multisets, $\widetilde{\boldsymbol{\mathcal{A}}} = \boldsymbol{\mathcal{B}} \cup \boldsymbol{\mathcal{C}}$ implies

$\widetilde{\boldsymbol{\mathcal{A}}} \setminus \boldsymbol{\mathcal{B}} = \boldsymbol{\mathcal{C}}$ and $\widetilde{\boldsymbol{\mathcal{A}}} \setminus \boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{B}}$.

$$\begin{aligned}
\mu_{\boldsymbol{C}}(\widetilde{\boldsymbol{\mathcal{A}}}, \boldsymbol{u}) &\triangleq \sum_{\boldsymbol{x} \in \widetilde{\boldsymbol{\mathcal{A}}}} \|\boldsymbol{x} - \boldsymbol{u}\|_{\boldsymbol{C}}^2 \\
&= \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{B}} \cup \boldsymbol{\mathcal{C}}} \|\boldsymbol{x} - \boldsymbol{u}\|_{\boldsymbol{C}}^2 \\
&= \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{B}}} \|\boldsymbol{x} - \boldsymbol{u}\|_{\boldsymbol{C}}^2 + \sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{C}}} \|\boldsymbol{y} - \boldsymbol{u}\|_{\boldsymbol{C}}^2,
\end{aligned}$$

so the function also obeys countable subadditivity (Equation A.2). Therefore the function μ is a measure. Note that the proof would be unchanged if the domain of the measure were a set, instead of a multiset. This is because countable subadditivity is defined only over mutually disjoint subsets.

When $\widetilde{\boldsymbol{\mathcal{A}}} = \emptyset$, $\mu_{\boldsymbol{C}}(\emptyset, \boldsymbol{u}) = 0$ and any $\boldsymbol{u}$ is a minimum. When $\widetilde{\boldsymbol{\mathcal{A}}} \neq \emptyset$, we start by differentiating the measure

$$\begin{aligned}
\boldsymbol{0} &= \frac{\partial}{\partial \boldsymbol{u}} \mu_{\boldsymbol{C}}(\widetilde{\boldsymbol{\mathcal{A}}}, \boldsymbol{u}) \\
&= \frac{\partial}{\partial \boldsymbol{u}} \sum_{\boldsymbol{x} \in \widetilde{\boldsymbol{\mathcal{A}}}} \|\boldsymbol{x} - \boldsymbol{u}\|_{\boldsymbol{C}}^2 \\
&= \sum_{\boldsymbol{x} \in \widetilde{\boldsymbol{\mathcal{A}}}} \frac{\partial}{\partial \boldsymbol{u}} (\boldsymbol{x} - \boldsymbol{u})^{\mathsf{T}} \boldsymbol{C} (\boldsymbol{x} - \boldsymbol{u}) \\
&= \sum_{\boldsymbol{x} \in \widetilde{\boldsymbol{\mathcal{A}}}} \frac{\partial}{\partial \boldsymbol{u}} \left(\boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} - 2\boldsymbol{u}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} + \boldsymbol{u}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{u} \right) \\
&= \sum_{\boldsymbol{x} \in \widetilde{\boldsymbol{\mathcal{A}}}} (-2\boldsymbol{C}\boldsymbol{x} + 2\boldsymbol{C}\boldsymbol{u}) \\
2|\widetilde{\boldsymbol{\mathcal{A}}}| \boldsymbol{C}\boldsymbol{u} &= 2\boldsymbol{C} \sum_{\boldsymbol{x} \in \widetilde{\boldsymbol{\mathcal{A}}}} \boldsymbol{x} \\
\Rightarrow \boldsymbol{u}^* &= \frac{1}{|\widetilde{\boldsymbol{\mathcal{A}}}|} \sum_{\boldsymbol{x} \in \widetilde{\boldsymbol{\mathcal{A}}}} \boldsymbol{x}.
\end{aligned}$$

To ensure a minimum, we check the Hessian,

$$\frac{\partial^2}{\partial \boldsymbol{u} \partial \boldsymbol{u}^{\mathsf{T}}} \mu_{\boldsymbol{C}}(\widetilde{\boldsymbol{\mathcal{A}}}, \boldsymbol{u}) = 2|\widetilde{\boldsymbol{\mathcal{A}}}| \boldsymbol{C},$$

which must be a symmetric PD matrix, by definition of $\boldsymbol{C}$ and the assumption that $|\widetilde{\boldsymbol{\mathcal{A}}}| > 0$. This implies that $\langle \widetilde{\boldsymbol{\mathcal{A}}} \rangle = \arg\min_{\boldsymbol{u} \in \boldsymbol{\mathcal{X}}} \mu_{\boldsymbol{C}}(\widetilde{\boldsymbol{\mathcal{A}}}, \boldsymbol{u})$. Note that the proof is similar to the derivation to the sample mean for *iid* draws from a multivariate Gaussian using the log-ML approach. □

A.1.2 Proof of Lemma 3.2

Proof.

$$\begin{aligned}
\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{A}}, \langle \boldsymbol{\mathcal{A}} \rangle) &\leq \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{A}}, \langle \boldsymbol{\mathcal{B}} \rangle) \\
&\leq \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{A}}, \langle \boldsymbol{\mathcal{B}} \rangle) + \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{B}} \setminus \boldsymbol{\mathcal{A}}, \langle \boldsymbol{\mathcal{B}} \rangle) \\
&= \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{B}}, \langle \boldsymbol{\mathcal{B}} \rangle) \\
&\leq \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{B}}, \langle \boldsymbol{\mathcal{C}} \rangle) \\
&\leq \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{B}}, \langle \boldsymbol{\mathcal{C}} \rangle) + \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{C}} \setminus \boldsymbol{\mathcal{B}}, \langle \boldsymbol{\mathcal{C}} \rangle) \\
&= \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{C}}, \langle \boldsymbol{\mathcal{C}} \rangle),
\end{aligned}$$

which completes the claim. □

A.1.3 Proof of Lemma 3.4

Proof. The sufficient statistics are the values needed to compute a function without the underlying set of data. We expand the definition of the measure as

$$\begin{aligned}
\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle) &\triangleq \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \| \boldsymbol{x} - \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle \|_{\boldsymbol{C}}^2 \\
&= \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \Big(\boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} - 2 \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} + \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle^{\mathsf{T}} \boldsymbol{C} \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle \Big) \\
&= \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \left(\boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} - 2 \Big(\frac{1}{|\boldsymbol{\mathcal{V}}_{v_i}|} \sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{y} \Big)^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} + \Big(\frac{1}{|\boldsymbol{\mathcal{V}}_{v_i}|} \sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{y} \Big)^{\mathsf{T}} \boldsymbol{C} \Big(\frac{1}{|\boldsymbol{\mathcal{V}}_{v_i}|} \sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{y} \Big) \right) \\
&= \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} - \frac{2}{|\boldsymbol{\mathcal{V}}_{v_i}|} \Big(\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x} \Big)^{\mathsf{T}} \boldsymbol{C} \Big(\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x} \Big) + \frac{|\boldsymbol{\mathcal{V}}_{v_i}|}{|\boldsymbol{\mathcal{V}}_{v_i}|^2} \Big(\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x} \Big)^{\mathsf{T}} \boldsymbol{C} \Big(\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x} \Big) \\
&= \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} - \frac{1}{|\boldsymbol{\mathcal{V}}_{v_i}|} \Big(\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x} \Big)^{\mathsf{T}} \boldsymbol{C} \Big(\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x} \Big).
\end{aligned}$$

From here, it is clear that the sufficient statistics are the scalar number of observations, the scalar sum of Mahalanobis distances, and the vector sum of observations

$$\xi_i = |\boldsymbol{\mathcal{V}}_{v_i}|; \qquad \kappa_i = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x}; \qquad \boldsymbol{\sigma}_i = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x}.$$

Using the sufficient statistics, the measure can be computed as

$$\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle) \quad \equiv \quad \kappa_i - \frac{1}{\xi_i} \boldsymbol{\sigma}_i{}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{\sigma}_i .$$

Likewise, the sufficient statistics can be used to compute the measure for the union of two multisets ($\boldsymbol{\mathcal{V}}_{v_k} = \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}$).

$$\begin{array}{rccccccc} \xi_k = & |\boldsymbol{\mathcal{V}}_{v_k}| & = & |\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}| & = & |\boldsymbol{\mathcal{V}}_{v_i}| + |\boldsymbol{\mathcal{V}}_{v_j}| & = & \xi_i + \xi_j . \\ \kappa_k = & \displaystyle\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_k}} \boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} & = & \displaystyle\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}} \boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} & = & \displaystyle\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} + \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_j}} \boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x} & = & \kappa_i + \kappa_j . \\ \boldsymbol{\sigma}_k = & \displaystyle\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_k}} \boldsymbol{x} & = & \displaystyle\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}} \boldsymbol{x} & = & \displaystyle\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x} + \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_j}} \boldsymbol{x} & = & \boldsymbol{\sigma}_i + \boldsymbol{\sigma}_j . \end{array}$$

This update rule implies that the measure can be computed independent of the cardinality of either multiset. □

A.1.4 Proof of Corollary 3.4.1

Proof. From Lemma 3.4, Equation 3.1 can be computed using sufficient statistics as

$$\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_k}, \langle \boldsymbol{\mathcal{V}}_{v_k} \rangle) \quad \equiv \quad \kappa_k - \frac{1}{\xi_k} \boldsymbol{\sigma}_k{}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{\sigma}_k .$$

Using the update rules, and if $\boldsymbol{\mathcal{V}}_{v_k} = \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}$ then

$$\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j} \rangle) \quad = \quad (\kappa_i + \kappa_j) - \frac{1}{\xi_i + \xi_j} \underbrace{(\boldsymbol{\sigma}_i + \boldsymbol{\sigma}_j)^{\mathsf{T}} \boldsymbol{C} (\boldsymbol{\sigma}_i + \boldsymbol{\sigma}_j)}_{(\star)} .$$

The cost of computing this equation is dominated by the quadratic term $(\star)$, which is of computational complexity $O(d^2)$, where d is the dimension of $\boldsymbol{\sigma}_k$. □

A.1.5 Proof of Theorem 3.8

Proof. Let $\boldsymbol{u}_i = \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle$ be the finite mean of observations from state $q_i \in \widehat{\boldsymbol{\lambda}}$. We also assume that the variance of observations from state $q_i \in \widehat{\boldsymbol{\lambda}}$ is finite. Let $\boldsymbol{u}_*$ be the finite mean of observations from some state $q_* \in \boldsymbol{\lambda}^*$ and

$\boldsymbol{\Sigma}_* = \arg\max_{q_k \in \boldsymbol{\lambda}^*} \mathrm{tr}[\boldsymbol{C}\mathrm{Var}(\boldsymbol{x}|q_k)]$.

$$\begin{aligned}\Pr\left\{\boldsymbol{\lambda}^* \xrightarrow{\gamma} q_i \in \widehat{\boldsymbol{\lambda}}\right\} &= 1 - \Pr\left\{\boldsymbol{\lambda}^* \not\xrightarrow{\gamma} q_i \in \widehat{\boldsymbol{\lambda}}\right\} \\ &= 1 - \Pr\left\{\|\boldsymbol{u}_i - \boldsymbol{u}_*\|_{\boldsymbol{C}} \geq \gamma\right\}, \forall q_* \in \boldsymbol{\lambda}^*.\end{aligned}$$

From Theorem 3.3, $\|\boldsymbol{x} - \langle\boldsymbol{\mathcal{V}}_{v_i}\rangle\|_{\boldsymbol{C}} \leq \sqrt{\epsilon}$, for all $\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}$. Using the Multivariate Chebyshev's Inequality (Section A.1.6), the probability that these states could generate one observation is

$$\Pr\left\{\|\boldsymbol{x} - \boldsymbol{u}_*\|_{\boldsymbol{C}} \geq \gamma - \sqrt{\epsilon}\right\} \leq \frac{\mathrm{tr}[\boldsymbol{C}\boldsymbol{\Sigma}_*]}{(\gamma - \sqrt{\epsilon})^2}.$$

Since $q_i \in \widehat{\boldsymbol{\lambda}}$ was created from $|\boldsymbol{\mathcal{V}}_{v_i}|$ observations from *iid* tasks,

$$\Pr\left\{\boldsymbol{\lambda}^* \xrightarrow{\gamma} q_i \in \widehat{\boldsymbol{\lambda}}\right\} > 1 - \left(\frac{\mathrm{tr}[\boldsymbol{C}\boldsymbol{\Sigma}_*]}{(\gamma - \sqrt{\epsilon})^2}\right)^{|\boldsymbol{\mathcal{V}}_{v_i}|}.$$

□

A.1.6 Multivariate Chebyshev's Inequality

Before beginning, we need a slight generalization of Markov's inequality, called Bienaymé's Inequality (Papoulis & Pillai, 2002).

Lemma A.1 (Bienaymé's Inequality). *For any random variable $X \geq 0$ and any real $r \geq 0$ and $\epsilon > 0$,*

$$\Pr\{X \geq \epsilon\} \leq \frac{\mathrm{E}\{X^r\}}{\epsilon^r},$$

provided $\mathrm{E}\{X^r\}$ exists.

Proof. The general case of $r > 0$ is shown as

$$\begin{aligned}
\mathrm{E}\{X^r\} &\triangleq \int_0^\infty x^r \mathrm{p}(x)\,dx \\
&\geq \int_\epsilon^\infty x^r \mathrm{p}(x)\,dx \\
&\geq \int_\epsilon^\infty \epsilon^r \mathrm{p}(x)\,dx \\
&= \epsilon^r \int_\epsilon^\infty \mathrm{p}(x)\,dx \\
&= \epsilon^r \Pr\{X \geq \epsilon\}.
\end{aligned}$$

The case of $r = 0$ is trivial, since $\Pr\{X \geq \epsilon\} \leq 1$ is always true. □

We can use this result to generalize Chebyshev's Inequality to the hyperelliptical case.[1] The trace of a matrix has many interesting properties, and we will make use of the following:

$$\begin{aligned}
\mathrm{tr}[a] &= a; \\
\mathrm{tr}[a\boldsymbol{A}] &= a\mathrm{tr}[\boldsymbol{A}]; \\
\mathrm{tr}[\boldsymbol{A}+\boldsymbol{B}] &= \mathrm{tr}[\boldsymbol{A}]+\mathrm{tr}[\boldsymbol{B}]; \\
\mathrm{tr}[\boldsymbol{A}\boldsymbol{B}] &= \mathrm{tr}[\boldsymbol{B}\boldsymbol{A}]; \\
\boldsymbol{a}^\mathsf{T}\boldsymbol{A}\boldsymbol{b} &= \mathrm{tr}\left[\boldsymbol{A}\boldsymbol{b}\boldsymbol{a}^\mathsf{T}\right].
\end{aligned}$$

Lemma A.2 (Multivariate Chebyshev's Inequality). *For any real-valued* $(d \times 1)$ *random vector,* $\boldsymbol{x}$*, any real* $\boldsymbol{y} \in \mathbb{R}^d$*,* $\epsilon > 0$*, and symmetric positive definite* $\boldsymbol{C} \in \mathbb{R}^d \times \mathbb{R}^d$*,*

$$\Pr\{\|\boldsymbol{x}-\boldsymbol{y}\|_{\boldsymbol{C}} \geq \epsilon\} \leq \frac{\mathrm{tr}[\boldsymbol{C}\mathrm{Var}(\boldsymbol{x})] + \|\boldsymbol{y}-\mathrm{E}\{\boldsymbol{x}\}\|_{\boldsymbol{C}}^2}{\epsilon^2},$$

provided $\boldsymbol{x}$ *has finite first and second moments.*

[1] After a fairly exhaustive search, we cannot find a similar inequality in reference books or the literature. Although, we still suspect that this inequality, or a similar one, has been derived before, it is outside our purview.

Proof. Let $\boldsymbol{\mu} = \mathrm{E}\{\boldsymbol{x}\}$ and $\boldsymbol{\Sigma} = \mathrm{Var}(\boldsymbol{x})$. From Bienaymé's Inequality, with $r = 2$, we get

$$\begin{aligned}
\epsilon^2 \Pr\{\|\boldsymbol{x}-\boldsymbol{y}\|_{\boldsymbol{C}} \geq \epsilon\} &\leq \mathrm{E}\Big\{\|\boldsymbol{x}-\boldsymbol{y}\|_{\boldsymbol{C}}^2\Big\} \\
&= \mathrm{E}\Big\{(\boldsymbol{x}-\boldsymbol{y})^{\mathsf{T}}\boldsymbol{C}(\boldsymbol{x}-\boldsymbol{y})\Big\} \\
&= \mathrm{E}\Big\{\mathrm{tr}\Big[\boldsymbol{C}(\boldsymbol{x}-\boldsymbol{y})(\boldsymbol{x}-\boldsymbol{y})^{\mathsf{T}}\Big]\Big\} \\
&= \mathrm{tr}\Big[\mathrm{E}\Big\{\boldsymbol{C}(\boldsymbol{x}-\boldsymbol{y})(\boldsymbol{x}-\boldsymbol{y})^{\mathsf{T}}\Big\}\Big] \\
&= \mathrm{tr}\Big[\boldsymbol{C}\mathrm{E}\Big\{\boldsymbol{x}\boldsymbol{x}^{\mathsf{T}} - 2\boldsymbol{x}\boldsymbol{y}^{\mathsf{T}} + \boldsymbol{y}\boldsymbol{y}^{\mathsf{T}}\Big\}\Big] \\
&= \mathrm{tr}\Big[\boldsymbol{C}\left(\mathrm{E}\Big\{\boldsymbol{x}\boldsymbol{x}^{\mathsf{T}}\Big\} - 2\boldsymbol{\mu}\boldsymbol{y}^{\mathsf{T}} + \boldsymbol{y}\boldsymbol{y}^{\mathsf{T}}\right)\Big] \\
&= \mathrm{tr}\Big[\boldsymbol{C}\left((\boldsymbol{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^{\mathsf{T}}) - 2\boldsymbol{\mu}\boldsymbol{y}^{\mathsf{T}} + \boldsymbol{y}\boldsymbol{y}^{\mathsf{T}}\right)\Big] \\
&= \mathrm{tr}[\boldsymbol{C}\boldsymbol{\Sigma}] + \boldsymbol{\mu}^{\mathsf{T}}\boldsymbol{C}\boldsymbol{\mu} - 2\boldsymbol{y}^{\mathsf{T}}\boldsymbol{C}\boldsymbol{\mu} + \boldsymbol{y}^{\mathsf{T}}\boldsymbol{C}\boldsymbol{y} \\
&= \mathrm{tr}[\boldsymbol{C}\boldsymbol{\Sigma}] + (\boldsymbol{y}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{C}(\boldsymbol{y}-\boldsymbol{\mu}) \\
&\doteq \mathrm{tr}[\boldsymbol{C}\mathrm{Var}(\boldsymbol{x})] + \|\boldsymbol{y}-\mathrm{E}\{\boldsymbol{x}\}\|_{\boldsymbol{C}}^2 .
\end{aligned}$$

This completes the claim. □

There are several useful special cases of this inequality, when $\boldsymbol{y} = \boldsymbol{\mu}$, $\boldsymbol{C} = \boldsymbol{I}_{d\times d}$, or $\boldsymbol{C} = \boldsymbol{\Sigma}^{-1}$. For instance,

$$\Pr\Big\{\|\boldsymbol{x}-\boldsymbol{\mu}\|_{\boldsymbol{\Sigma}^{-1}} \geq \epsilon\Big\} \leq \frac{d}{\epsilon^2},$$

where d is the length of $\boldsymbol{x}$.

A.1.7 Proof of Theorem 3.9

Proof. The definition of the Viterbi Approximation is

$$\widehat{\alpha}_{\boldsymbol{\lambda}}^{c} \triangleq \max_{q_0,\ldots,q_N} \mathrm{p}(\boldsymbol{x}_N^c|c_N{=}q_N,\boldsymbol{\lambda})\,\mathrm{P}(c_N{=}q_N|c_{N-1}{=}q_{N-1},\boldsymbol{\lambda})\cdots\mathrm{p}(\boldsymbol{x}_0^c|c_0{=}q_0,\boldsymbol{\lambda})\,\mathrm{P}(c_0{=}q_0|\boldsymbol{\lambda}) .$$

Since we are considering the log conditional-likelihood ratio, then we need to consider only those Viterbi paths that include state q_i or state q_j in the original CDHMM $\boldsymbol{\lambda}$, or q_k in the merged CDHMM $\widetilde{\boldsymbol{\lambda}}$. The only components of the Viterbi Approximation that change involve either state q_i or state q_j (Figure A.1).

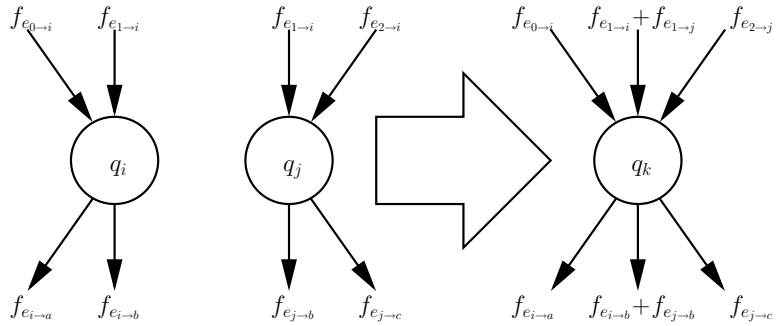


Figure A.1: Hypothetical merging of state q_i and q_j to form state q_k.

$$\log \frac{\prod_{m=1}^{M} \widehat{\alpha}_{\tilde{\boldsymbol{\lambda}}}^{m}}{\prod_{m=1}^{M} \widehat{\alpha}_{\boldsymbol{\lambda}}^{m}} \equiv \underbrace{\log \frac{\prod_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_k}} b_k(\boldsymbol{x})}{\prod_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_i}} b_i(\boldsymbol{x}) \prod_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_j}} b_j(\boldsymbol{x})}}_{(\star)} + \underbrace{\log \prod_{q_l\in\mathcal{Q}} \frac{a_{l|k}^{f_{e_{k\to l}}}}{a_{l|i}^{f_{e_{i\to l}}} a_{l|j}^{f_{e_{j\to l}}}}}_{(\diamond)} + \underbrace{\log \prod_{q_l\in\mathcal{Q}} \frac{a_{k|l}^{f_{e_{l\to k}}}}{a_{i|l}^{f_{e_{l\to i}}} a_{j|l}^{f_{e_{l\to j}}}}}_{(\dagger)},$$

where $(\star)$ is the log observation-likelihood ratio, $(\diamond)$ is the log inbound transition-probability ratio, and $(\dagger)$ is the difference in the log outbound transition-probability ratio. If we merge state q_i and state q_j to form q_k then the log observation-likelihood ratio is

$$\begin{aligned} 2\log \frac{\prod_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_k}} b_k(\boldsymbol{x})}{\prod_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_i}} b_i(\boldsymbol{x}) \prod_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_j}} b_j(\boldsymbol{x})} &= 2\sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_k}} \log b_k(\boldsymbol{x}) - 2\sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_i}} \log b_i(\boldsymbol{x}) - 2\sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_j}} \log b_j(\boldsymbol{x}) \\ &= -\sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_k}} \|\boldsymbol{x}-\langle\boldsymbol{\mathcal{V}}_{q_k}\rangle\|^2_{\boldsymbol{\Sigma}^{-1}} + \sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_i}} \|\boldsymbol{x}-\langle\boldsymbol{\mathcal{V}}_{q_i}\rangle\|^2_{\boldsymbol{\Sigma}^{-1}} + \sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{q_j}} \left\|\boldsymbol{x}-\langle\boldsymbol{\mathcal{V}}_{q_j}\rangle\right\|^2_{\boldsymbol{\Sigma}^{-1}} \\ &\dot{=} \mu_{\boldsymbol{\Sigma}^{-1}}(\boldsymbol{\mathcal{V}}_{q_i}, \langle\boldsymbol{\mathcal{V}}_{q_i}\rangle) + \mu_{\boldsymbol{\Sigma}^{-1}}(\boldsymbol{\mathcal{V}}_{q_j}, \langle\boldsymbol{\mathcal{V}}_{q_j}\rangle) - \mu_{\boldsymbol{\Sigma}^{-1}}(\boldsymbol{\mathcal{V}}_{q_k}, \langle\boldsymbol{\mathcal{V}}_{q_k}\rangle). \end{aligned} \tag{A.3}$$

Using the sufficient statistics derived in Lemma 3.4,

$$\xi_k = \xi_i + \xi_j; \qquad \kappa_k = \kappa_i + \kappa_j; \qquad \boldsymbol{\sigma}_k = \boldsymbol{\sigma}_i + \boldsymbol{\sigma}_j.$$

We can rewrite Equation A.3 as

$$\begin{aligned}
&\mu_{\boldsymbol{\Sigma}^{-1}}(\boldsymbol{\mathcal{V}}_{q_i}, \langle \boldsymbol{\mathcal{V}}_{q_i} \rangle) + \mu_{\boldsymbol{\Sigma}^{-1}}(\boldsymbol{\mathcal{V}}_{q_j}, \langle \boldsymbol{\mathcal{V}}_{q_j} \rangle) - \mu_{\boldsymbol{\Sigma}^{-1}}(\boldsymbol{\mathcal{V}}_{q_k}, \langle \boldsymbol{\mathcal{V}}_{q_k} \rangle) \\
&\doteq \kappa_i - \frac{\boldsymbol{\sigma}_i^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{\sigma}_i}{\xi_i} + \kappa_j - \frac{\boldsymbol{\sigma}_j^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{\sigma}_j}{\xi_j} - \kappa_k + \frac{\boldsymbol{\sigma}_k^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{\sigma}_k}{\xi_k} \\
&= \frac{(\boldsymbol{\sigma}_i + \boldsymbol{\sigma}_j)^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} (\boldsymbol{\sigma}_i + \boldsymbol{\sigma}_j)}{\xi_i + \xi_j} - \frac{\boldsymbol{\sigma}_i^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{\sigma}_i}{\xi_i} - \frac{\boldsymbol{\sigma}_j^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{\sigma}_j}{\xi_j} \\
&= \frac{\boldsymbol{\sigma}_i^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{\sigma}_i}{\xi_i + \xi_j} + \frac{\boldsymbol{\sigma}_j^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{\sigma}_j}{\xi_i + \xi_j} + 2\frac{\boldsymbol{\sigma}_i^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{\sigma}_j}{\xi_i + \xi_j} - \frac{\boldsymbol{\sigma}_i^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{\sigma}_i}{\xi_i} - \frac{\boldsymbol{\sigma}_j^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{\sigma}_j}{\xi_j} \\
&= \frac{-1}{\xi_i + \xi_j} \left(\frac{\xi_j}{\xi_i} \|\boldsymbol{\sigma}_i\|^2_{\boldsymbol{\Sigma}^{-1}} + \frac{\xi_i}{\xi_j} \|\boldsymbol{\sigma}_j\|^2_{\boldsymbol{\Sigma}^{-1}} - 2\boldsymbol{\sigma}_i^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{\sigma}_j \right) \\
&= \frac{-1}{\xi_i + \xi_j} \left\| \boldsymbol{\sigma}_i \sqrt{\frac{\xi_j}{\xi_i}} - \boldsymbol{\sigma}_j \sqrt{\frac{\xi_i}{\xi_j}} \right\|^2_{\boldsymbol{\Sigma}^{-1}}.
\end{aligned}$$

If we use the convention that $0 \log 0 = 0$, noting that the edge counts from state q_k equal the sum from the two merged nodes, the log inbound transition-probability ratio is

$$\begin{aligned}
\log \prod_{q_l \in \mathcal{Q}} \frac{a_{k|l}^{f_{e_{l\to k}}}}{a_{i|l}^{f_{e_{l\to i}}} a_{j|l}^{f_{e_{l\to j}}}} &= \sum_{q_l \in \mathcal{Q}} \left(f_{e_{l\to k}} \log a_{k|l} - f_{e_{l\to i}} \log a_{i|l} - f_{e_{l\to j}} \log a_{j|l} \right) \\
&= \sum_{q_l \in \mathcal{Q}} \left((f_{e_{l\to i}} + f_{e_{l\to j}}) \log \frac{f_{e_{l\to i}} + f_{e_{l\to j}}}{\xi_l} - f_{e_{l\to i}} \log \frac{f_{e_{l\to i}}}{\xi_l} - f_{e_{l\to j}} \log \frac{f_{e_{l\to j}}}{\xi_l} \right) \\
&= \sum_{q_l \in \mathcal{Q}} \left(f_{e_{l\to i}} \log \frac{f_{e_{l\to i}} + f_{e_{l\to j}}}{f_{e_{l\to i}}} + f_{e_{l\to j}} \log \frac{f_{e_{l\to i}} + f_{e_{l\to j}}}{f_{e_{l\to j}}} \right).
\end{aligned}$$

The log outbound transition-probability ratio is

$$\begin{aligned}
\log \prod_{q_l \in \mathcal{Q}} \frac{a_{l|k}^{f_{e_{k\to l}}}}{a_{l|i}^{f_{e_{i\to l}}} a_{l|j}^{f_{e_{j\to l}}}} &= \sum_{q_l \in \mathcal{Q}} \left(f_{e_{k\to l}} \log a_{l|k} - f_{e_{i\to l}} \log a_{l|i} - f_{e_{j\to l}} \log a_{l|j} \right) \\
&= \sum_{q_l \in \mathcal{Q}} \left((f_{e_{i\to l}} + f_{e_{j\to l}}) \log \frac{f_{e_{i\to l}} + f_{e_{j\to l}}}{\xi_i + \xi_j} - f_{e_{i\to l}} \log \frac{f_{e_{i\to l}}}{\xi_i} - f_{e_{j\to l}} \log \frac{f_{e_{j\to l}}}{\xi_j} \right) \\
&= \sum_{q_l \in \mathcal{Q}} \left(f_{e_{i\to l}} \log \frac{\xi_i}{\xi_i + \xi_j} \frac{f_{e_{i\to l}} + f_{e_{j\to l}}}{f_{e_{i\to l}}} + f_{e_{j\to l}} \log \frac{\xi_j}{\xi_i + \xi_j} \frac{f_{e_{i\to l}} + f_{e_{j\to l}}}{f_{e_{j\to l}}} \right).
\end{aligned}$$

When we put it all together, the log conditional-likelihood ratio from merging state q_i and q_j is

$$\begin{aligned}\log \prod_{m=1}^{M} \widehat{\alpha}_{\widetilde{\boldsymbol{\lambda}}}^{m} - \log \prod_{m=1}^{M} \widehat{\alpha}_{\boldsymbol{\lambda}}^{m} &= \frac{-1/2}{\xi_i + \xi_j} \left\| \boldsymbol{\sigma}_i \sqrt{\frac{\xi_j}{\xi_i}} - \boldsymbol{\sigma}_j \sqrt{\frac{\xi_i}{\xi_j}} \right\|_{\boldsymbol{\Sigma}^{-1}}^2 \\ &\quad + \sum_{q_l \in \mathcal{Q}} \left(f_{e_{l\to i}} \log \frac{f_{e_{l\to i}} + f_{e_{l\to j}}}{f_{e_{l\to i}}} + f_{e_{l\to j}} \log \frac{f_{e_{l\to i}} + f_{e_{l\to j}}}{f_{e_{l\to j}}} \right) \\ &\quad + \sum_{q_l \in \mathcal{Q}} \left(f_{e_{i\to l}} \log \frac{\xi_i}{\xi_i + \xi_j} \frac{f_{e_{i\to l}} + f_{e_{j\to l}}}{f_{e_{i\to l}}} + f_{e_{j\to l}} \log \frac{\xi_j}{\xi_i + \xi_j} \frac{f_{e_{i\to l}} + f_{e_{j\to l}}}{f_{e_{j\to l}}} \right).\end{aligned}$$

Furthermore, this quantity can be computed independent of the cardinality of any state in the CDHMM. □

A.1.8 Proof of Theorem 5.1

Proof. The proof is similar to Theorem 5.5.1 in Golub and Van Loan (1996). Let $\boldsymbol{A} = \boldsymbol{X}_{1:N} - \boldsymbol{X}_{0:N-1}$ and $\boldsymbol{B} = \boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N$. Furthermore, let $\boldsymbol{B} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}}$ be the SVD of $\boldsymbol{B}$, and let $\boldsymbol{B}$ have full row rank, d. Let $\widehat{\boldsymbol{R}}_{\mathrm{LS}}$ be the least-squares estimate of $\boldsymbol{R}$, i.e., the matrix that minimizes the squared Frobenius norm of one-step prediction errors (Kreyszig, 1999). Let the squared error of the least-squares estimate be

$$\begin{aligned}\varepsilon_{\mathrm{LS}}^2 &= \left\| \boldsymbol{X}_{1:N} - \left(\widehat{\boldsymbol{R}}_{\mathrm{LS}} \left(\boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N \right) + \boldsymbol{X}_{0:N-1} \right) \right\|_F^2 \\ &= \left\| \left(\boldsymbol{X}_{1:N} - \boldsymbol{X}_{0:N-1} \right) - \widehat{\boldsymbol{R}}_{\mathrm{LS}} \left(\boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N \right) \right\|_F^2 \\ &\doteq \left\| \boldsymbol{A} - \widehat{\boldsymbol{R}}_{\mathrm{LS}} \boldsymbol{B} \right\|_F^2 \\ &= \left\| \boldsymbol{A}\boldsymbol{V} - \widehat{\boldsymbol{R}}_{\mathrm{LS}} \boldsymbol{U}\boldsymbol{U}^{\mathsf{T}}\boldsymbol{B}\boldsymbol{V} \right\|_F^2 \\ &= \left\| \boldsymbol{A}\boldsymbol{V} - \widehat{\boldsymbol{R}}_{\mathrm{LS}} \boldsymbol{U}\boldsymbol{\Sigma} \right\|_F^2 \\ &= \left\| \boldsymbol{A} \left[\boldsymbol{v}_1 \cdots \boldsymbol{v}_N \right] - \widehat{\boldsymbol{R}}_{\mathrm{LS}} \left[\sigma_1^2 \boldsymbol{u}_1 \cdots \sigma_d^2 \boldsymbol{u}_d \boldsymbol{0} \cdots \boldsymbol{0} \right] \right\|_F^2 \\ &= \underbrace{\sum_{n=1}^{d} \left\| \boldsymbol{A}\boldsymbol{v}_n - \sigma_n^2 \widehat{\boldsymbol{R}}_{\mathrm{LS}} \boldsymbol{u}_n \right\|_2^2}_{(\diamond)} + \underbrace{\sum_{n=d+1}^{N} \left\| \boldsymbol{A}\boldsymbol{v}_n \right\|_2^2}_{(\star)}.\end{aligned}$$

Since the right term $(\star)$ does not depend on $\widehat{\boldsymbol{R}}_{\mathrm{LS}}$, there is no way to minimize it. The left term $(\diamond)$ can forced to zero by setting $\widehat{\boldsymbol{R}}_{\mathrm{LS}} = \boldsymbol{A}\boldsymbol{V}\mathrm{diag}\left(1/\sigma_1^2, \ldots, 1/\sigma_d^2, 0, \ldots, 0\right)\boldsymbol{U}^{\mathsf{T}}$. Then we have

$$\begin{aligned}
\widehat{\boldsymbol{R}}_{\mathrm{LS}} &= \boldsymbol{A}\boldsymbol{V}\mathrm{diag}\left(1/\sigma_1^2, \ldots, 1/\sigma_d^2, 0, \ldots, 0\right)\boldsymbol{U}^{\mathsf{T}} \\
&= \boldsymbol{A}\boldsymbol{B}^{\mathsf{R}} \\
&\dot{=} \left(\boldsymbol{X}_{1:N} - \boldsymbol{X}_{0:N-1}\right)\left(\boldsymbol{X}_{0:N-1} - \boldsymbol{\Gamma}_N\right)^{\mathsf{R}}.
\end{aligned}$$

□

A.1.9 Lemmas for Theorem 5.2

Lemma A.3. $\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{T}} = \boldsymbol{\delta}_0\boldsymbol{\delta}_0{}^{\mathsf{T}} + \boldsymbol{\Delta}_{1:N}\boldsymbol{\Delta}_{1:N}{}^{\mathsf{T}}$.

Proof.

$$\begin{aligned}
\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{T}} &= \sum_{n=0}^{N-1} \boldsymbol{\delta}_n\boldsymbol{\delta}_n{}^{\mathsf{T}} \\
&= \boldsymbol{\delta}_0\boldsymbol{\delta}_0 + \sum_{n=0}^{N-1} \boldsymbol{\delta}_{n+1}\boldsymbol{\delta}_{n+1}{}^{\mathsf{T}} - \boldsymbol{\delta}_N\boldsymbol{\delta}_N{}^{\mathsf{T}} \\
&= \boldsymbol{\delta}_0\boldsymbol{\delta}_0 + \boldsymbol{\Delta}_{1:N}\boldsymbol{\Delta}_{1:N}{}^{\mathsf{T}},
\end{aligned}$$

since by definition $\boldsymbol{\delta}_N = \mathbf{0}$. □

Lemma A.4. *Let $d < N$ be the number of rows and columns in $\boldsymbol{\Delta}_{0:N-1}$ and let $\boldsymbol{\Delta}_{0:N-1}$ have full row rank. This implies*

$$\boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{R}}\boldsymbol{\Delta}_{0:N-1} = \sum_{i=1}^{d} \boldsymbol{v}_i\boldsymbol{v}_i{}^{\mathsf{T}},$$

where $\boldsymbol{v}_i$ is the ith column of the matrix $\boldsymbol{V}$ resulting from the SVD of $\boldsymbol{\Delta}_{0:N-1}$.

Proof. Let the SVD be $\boldsymbol{\Delta}_{0:N-1} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}}$.

$$
\begin{aligned}
\boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{R}}\boldsymbol{\Delta}_{0:N-1} &= \boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{T}}\left(\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{T}}\right)^{-1}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}} \\
&= \boldsymbol{V}\boldsymbol{\Sigma}^{\mathsf{T}}\boldsymbol{U}^{\mathsf{T}}\left(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}}\boldsymbol{V}\boldsymbol{\Sigma}^{\mathsf{T}}\boldsymbol{U}^{\mathsf{T}}\right)^{-1}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}} \\
&= \boldsymbol{V}\boldsymbol{\Sigma}^{\mathsf{T}}\boldsymbol{U}^{\mathsf{T}}\left(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^{\mathsf{T}}\boldsymbol{U}^{\mathsf{T}}\right)^{-1}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}} \\
&= \boldsymbol{V}\boldsymbol{\Sigma}^{\mathsf{T}}\boldsymbol{U}^{\mathsf{T}}\boldsymbol{U}\left(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^{\mathsf{T}}\right)^{-1}\boldsymbol{U}^{\mathsf{T}}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}} \\
&= \boldsymbol{V}\boldsymbol{\Sigma}^{\mathsf{T}}\left(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^{\mathsf{T}}\right)^{-1}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}} \\
&= \boldsymbol{V}\boldsymbol{\Sigma}^{\mathsf{T}}\mathrm{diag}(1/\sigma_1^2,\ldots,1/\sigma_d^2)\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}} \\
&= [\boldsymbol{v}_1\cdots\boldsymbol{v}_d\cdots\boldsymbol{v}_N]\begin{bmatrix}\boldsymbol{I}_{(d\times d)} & \boldsymbol{0}_{(d\times d-N)}\\ \boldsymbol{0}_{(N-d\times d)} & \boldsymbol{0}_{(N-d\times N-d)}\end{bmatrix}\begin{bmatrix}\boldsymbol{v}_1^{\mathsf{T}}\\ \vdots \\ \boldsymbol{v}_d^{\mathsf{T}}\\ \vdots \\ \boldsymbol{v}_N^{\mathsf{T}}\end{bmatrix} \\
&= \sum_{i=1}^{d}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}.
\end{aligned}
$$

□

A.1.10 Proof of Theorem 5.2

Proof. Let the equilibrium point of Equation 5.4 be $\boldsymbol{x}_\infty$,

$$
\begin{aligned}
\boldsymbol{x}_\infty &= (\widehat{\boldsymbol{R}}+\boldsymbol{I})\boldsymbol{x}_\infty - \widehat{\boldsymbol{R}}\boldsymbol{x}_N \\
-\widehat{\boldsymbol{R}}\boldsymbol{x}_\infty &= -\widehat{\boldsymbol{R}}\boldsymbol{x}_N \\
\Rightarrow \boldsymbol{x}_\infty &= \boldsymbol{x}_N,
\end{aligned}
$$

provided $\widehat{\boldsymbol{R}}^{-1}$ exists.

Let d be the number of rows in $\boldsymbol{\Delta}_{0:N-1}$. We use the Discrete Algebraic Lyapunov Equation (Equation 5.6) to

show stability i.s.L. by choosing the symmetric PD matrix $\boldsymbol{P} = \boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}}$,

$$
\begin{aligned}
&\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}} - \boldsymbol{A}\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}}\boldsymbol{A}^{\mathsf{T}} \\
&\quad = \boldsymbol{\Delta}_{1:N}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{R}}\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{1:N}^{\mathsf{T}}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{R}^{\mathsf{T}}}\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} \\
&\quad = \boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}} - \boldsymbol{\Delta}_{1:N}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}}(\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}})^{-1}\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}}(\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}})^{-1}\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} \\
&\quad = \boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}} - \boldsymbol{\Delta}_{1:N}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}}(\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}})^{-1}\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} \\
&\quad = \boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{T}} - \boldsymbol{\Delta}_{1:N}\boldsymbol{\Delta}_{0:N-1}^{\mathsf{R}}\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} \\
&\quad = \boldsymbol{Q}.
\end{aligned}
$$

From Lemma A.3 and Lemma A.4 and, by definition, $\boldsymbol{\delta}_N = \boldsymbol{0}$,

$$
\begin{aligned}
\boldsymbol{Q} &= \boldsymbol{\delta}_0\boldsymbol{\delta}_0^{\mathsf{T}} + \boldsymbol{\Delta}_{1:N}\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} - \boldsymbol{\delta}_N\boldsymbol{\delta}_N^{\mathsf{T}} - \boldsymbol{\Delta}_{1:N}\left(\sum_{i=1}^{d}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}\right)\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} \\
&= \boldsymbol{\delta}_0\boldsymbol{\delta}_0^{\mathsf{T}} + \boldsymbol{\Delta}_{1:N}\boldsymbol{I}\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} - \boldsymbol{\Delta}_{1:N}\left(\sum_{i=1}^{d}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}\right)\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} \\
&= \boldsymbol{\delta}_0\boldsymbol{\delta}_0^{\mathsf{T}} + \boldsymbol{\Delta}_{1:N}\boldsymbol{V}\boldsymbol{V}^{\mathsf{T}}\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} - \boldsymbol{\Delta}_{1:N}\left(\sum_{i=1}^{d}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}\right)\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} \\
&= \boldsymbol{\delta}_0\boldsymbol{\delta}_0^{\mathsf{T}} + \boldsymbol{\Delta}_{1:N}\left(\sum_{i=1}^{N}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}\right)\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} - \boldsymbol{\Delta}_{1:N}\left(\sum_{i=1}^{d}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}\right)\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} \\
&= \boldsymbol{\delta}_0\boldsymbol{\delta}_0^{\mathsf{T}} + \boldsymbol{\Delta}_{1:N}\left(\sum_{i=d+1}^{N}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}\right)\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} \\
&\quad + \boldsymbol{\Delta}_{1:N}\left(\sum_{i=1}^{d}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}\right)\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} - \boldsymbol{\Delta}_{1:N}\left(\sum_{i=1}^{d}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}\right)\boldsymbol{\Delta}_{1:N}^{\mathsf{T}} \\
&= \boldsymbol{\delta}_0\boldsymbol{\delta}_0^{\mathsf{T}} + \boldsymbol{\Delta}_{1:N}\left(\sum_{i=d+1}^{N}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}\right)\boldsymbol{\Delta}_{1:N}^{\mathsf{T}}.
\end{aligned}
$$

Clearly, this matrix is symmetric PSD since

$$
\begin{aligned}
\boldsymbol{x}^{\mathsf{T}}\boldsymbol{Q}\boldsymbol{x} &= \boldsymbol{x}^{\mathsf{T}}\boldsymbol{\delta}_0\boldsymbol{\delta}_0^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{x}^{\mathsf{T}}\boldsymbol{\Delta}_{1:N}\left(\sum_{i=d+1}^{N}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}\right)\boldsymbol{\Delta}_{1:N}^{\mathsf{T}}\boldsymbol{x} \\
&= \boldsymbol{x}^{\mathsf{T}}\boldsymbol{\delta}_0\boldsymbol{\delta}_0^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{y}^{\mathsf{T}}\left(\sum_{i=d+1}^{N}\boldsymbol{v}_i\boldsymbol{v}_i^{\mathsf{T}}\right)\boldsymbol{y} \\
&\geq 0,
\end{aligned}
$$

for any $\boldsymbol{x}$. The matrix is PD unless $\boldsymbol{\delta}_0$ lies in the space spanned by the first d principal row components of $\boldsymbol{\Delta}_{0:N-1}$. Since the matrix $\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{T}} - \boldsymbol{A}\boldsymbol{\Delta}_{0:N-1}\boldsymbol{\Delta}_{0:N-1}{}^{\mathsf{T}}\boldsymbol{A}^{\mathsf{T}}$ is PSD, then the dynamical system in Equation 5.4 is stable i.s.L.. □

A.2 Derivations

A.2.1 Derivation of Equation 3.4

The prior is the description length of the model (Stolcke & Omohundro, 1994b):

$$\begin{aligned}
\ell\,(q_i) &\cong \log|\mathrm{Var}(\boldsymbol{x}|q_i)\,| + |E_{q_i}|\log|\mathcal{Q}| \\
\mathrm{p}(q_i) &\propto \exp\{-\ell\,(q_i)\} \\
&\cong |\mathrm{Var}(\boldsymbol{x}|q_i)\,|^{-1}|\mathcal{Q}|^{-|E_{q_i}|} \\
\mathrm{p}(\boldsymbol{\lambda}) &= \prod_{q_i\in\mathcal{Q}} \mathrm{p}(q_i) \\
&\tilde{\propto} \prod_{q_i\in\mathcal{Q}} |\mathrm{Var}(\boldsymbol{x}|q_i)\,|^{-1}|\mathcal{Q}|^{-|E_{q_i}|} \\
&= |\mathcal{Q}|^{-|E|}\prod_{q_i\in\mathcal{Q}} |\mathrm{Var}(\boldsymbol{x}|q_i)\,|^{-1}.
\end{aligned}$$

A.2.2 Derivation of Equation 3.9

The "forward variables" are defined as

$$\begin{aligned}
\alpha_n(j) &\triangleq \mathrm{p}(c_n{=}q_j, \boldsymbol{X}^c_{0:n}|\boldsymbol{\lambda}) \\
&= \sum_{q_i\in\mathcal{Q}} \mathrm{p}(\boldsymbol{x}_n|c_n{=}q_j,\boldsymbol{\lambda})\,\mathrm{p}\big(c_n{=}q_j, c_{n-1}{=}q_i, \boldsymbol{X}^c_{0:n-1}|\boldsymbol{\lambda}\big) \\
&= \mathrm{p}(\boldsymbol{x}_n|c_n{=}q_j,\boldsymbol{\lambda})\sum_{q_i\in\mathcal{Q}} \mathrm{P}(c_n{=}q_j|c_{n-1}{=}q_i,\boldsymbol{\lambda})\,\mathrm{p}\big(c_{n-1}{=}q_i, \boldsymbol{X}^c_{0:n-1}|\boldsymbol{\lambda}\big) \\
&\doteq \begin{cases} b_j(\boldsymbol{x}_n)\sum\limits_{q_i\in\mathcal{Q}} a_{j|i}\alpha_{n-1}(i), & n>0 \\ b_j(\boldsymbol{x}_0)\pi_j, & n=0 \end{cases}.
\end{aligned}$$

The next-state pmf is defined as

$$\begin{aligned}
\nu_n(j) &\triangleq \mathrm{P}\big(c_n{=}q_j|\boldsymbol{X}_{0:n-1}^c,\boldsymbol{\lambda}\big)\\
&= \frac{\mathrm{p}\big(c_n{=}q_j,\boldsymbol{X}_{0:n-1}^c|\boldsymbol{\lambda}\big)}{\mathrm{p}\big(\boldsymbol{X}_{0:n-1}^c|\boldsymbol{\lambda}\big)}\\
&= \frac{\sum\limits_{q_i\in\mathcal{Q}}\mathrm{p}\big(c_n{=}q_j,c_{n-1}{=}q_i,\boldsymbol{X}_{0:n-1}^c|\boldsymbol{\lambda}\big)}{\sum\limits_{q_k\in\mathcal{Q}}\mathrm{p}\big(c_{n-1}{=}q_k,\boldsymbol{X}_{0:n-1}^c|\boldsymbol{\lambda}\big)}\\
&= \frac{\sum\limits_{q_i\in\mathcal{Q}}\mathrm{P}(c_n{=}q_j|c_{n-1}{=}q_i,\boldsymbol{\lambda})\,\mathrm{p}\big(c_{n-1}{=}q_i,\boldsymbol{X}_{0:n-1}^c|\boldsymbol{\lambda}\big)}{\sum\limits_{q_k\in\mathcal{Q}}\mathrm{p}\big(c_{n-1}{=}q_k,\boldsymbol{X}_{0:n-1}^c|\boldsymbol{\lambda}\big)}\\
&\doteq \frac{\sum\limits_{q_i\in\mathcal{Q}}a_{j|i}\alpha_{n-1}(i)}{\sum\limits_{q_k\in\mathcal{Q}}\alpha_{n-1}(k)}.
\end{aligned}$$

If we use these definitions, the ML estimator, Equation 3.9, is computed as

$$\begin{aligned}
\widehat{\boldsymbol{x}}_n^* &= \arg\max_{\boldsymbol{x}_n}\mathrm{p}\big(\boldsymbol{x}_n|\boldsymbol{X}_{0:n-1}^c,\boldsymbol{\lambda}\big)\\
&= \arg\max_{\boldsymbol{x}_n}\sum_{q_j\in\mathcal{Q}}\mathrm{p}\big(\boldsymbol{x}_n,c_n{=}q_j|\boldsymbol{X}_{0:n-1}^c,\boldsymbol{\lambda}\big)\\
&= \arg\max_{\boldsymbol{x}_n}\sum_{q_j\in\mathcal{Q}}\mathrm{p}(\boldsymbol{x}_n|c_n{=}q_j,\boldsymbol{\lambda})\,\mathrm{P}\big(c_n{=}q_j|\boldsymbol{X}_{0:n-1}^c,\boldsymbol{\lambda}\big)\\
&\doteq \arg\max_{\boldsymbol{x}_n}\sum_{q_j\in\mathcal{Q}}b_j(\boldsymbol{x}_n)\nu_n(j).
\end{aligned}$$

A.2.3 Derivation of Equation 4.5

Because the observation pdf is Gaussian and $\boldsymbol{\Psi}$ is a diagonal (symmetric) matrix,

$$\begin{aligned}
\frac{\partial}{\partial\psi}b_{j,\psi}(\boldsymbol{x}_n) &= \frac{\partial}{\partial\psi}\mathrm{p}(\boldsymbol{x}_n|c_n{=}q_j,\psi,\boldsymbol{\lambda}) \\
&= \frac{\partial}{\partial\psi}\frac{\exp\left\{-\frac{1}{2}(\boldsymbol{\Psi}\boldsymbol{x}_n-\boldsymbol{\mu}_j)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Psi}\boldsymbol{x}_n-\boldsymbol{\mu}_j)\right\}}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}} \\
&= \mathrm{p}(\boldsymbol{x}_n|c_n{=}q_j,\psi,\boldsymbol{\lambda})\,\frac{\partial}{\partial\psi}(\boldsymbol{\Psi}\boldsymbol{x}_n-\boldsymbol{\mu}_j)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Psi}\boldsymbol{x}_n-\boldsymbol{\mu}_j) \\
&= \mathrm{p}(\boldsymbol{x}_n|c_n{=}q_j,\psi,\boldsymbol{\lambda})\left\{(\boldsymbol{\Psi}'\boldsymbol{x}_n)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j-\boldsymbol{\Psi}\boldsymbol{x}_n)\right\} \\
&\doteq \left\{(\boldsymbol{\Psi}'\boldsymbol{x}_n)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j-\boldsymbol{\Psi}\boldsymbol{x}_n)\right\}b_{j,\psi}(\boldsymbol{x}_n).
\end{aligned}$$

We will also need the following notation:

$$\begin{aligned}
\mathrm{p}\big(c_n{=}q_j,\boldsymbol{X}^c_{0:n-1}|\psi,\boldsymbol{\lambda}\big) &= \sum_{q_i\in\mathcal{Q}}\mathrm{p}\big(c_n{=}q_j,c_{n-1}{=}q_i,\boldsymbol{X}^c_{0:n-1}|\psi,\boldsymbol{\lambda}\big) \\
&= \sum_{q_i\in\mathcal{Q}}\mathrm{P}(c_n{=}q_j|c_{n-1}{=}q_i,\boldsymbol{\lambda})\,\mathrm{p}\big(c_{n-1}{=}q_i,\boldsymbol{X}^c_{0:n-1}|\psi,\boldsymbol{\lambda}\big) \\
&\doteq \sum_{q_i\in\mathcal{Q}}a_{j|i}\alpha_{n-1}(i,\psi),
\end{aligned}$$

and its corresponding partial derivative:

$$\frac{\partial}{\partial\psi}\mathrm{p}\big(c_n{=}q_j,\boldsymbol{X}^c_{0:n-1}|\psi,\boldsymbol{\lambda}\big) \doteq \sum_{q_i\in\mathcal{Q}}a_{j|i}\frac{\partial}{\partial\psi}\alpha_{n-1}(i,\psi).$$

The partial derivative of Equation 4.4 with respect to the scale factor is

$$
\begin{aligned}
\frac{\partial}{\partial\psi}\alpha_n(j,\psi) &= \frac{\partial}{\partial\psi}\mathrm{p}(c_n{=}q_j, \boldsymbol{X}^c_{0:n}|\psi,\boldsymbol{\lambda}) \\
&= \frac{\partial}{\partial\psi}\mathrm{p}(\boldsymbol{x}_n|c_n{=}q_j,\psi,\boldsymbol{\lambda})\,\mathrm{p}\big(c_n{=}q_j,\boldsymbol{X}^c_{0:n-1}|\psi,\boldsymbol{\lambda}\big) \\
&= \mathrm{p}\big(c_n{=}q_j,\boldsymbol{X}^c_{0:n-1}|\psi,\boldsymbol{\lambda}\big)\,\frac{\partial}{\partial\psi}\mathrm{p}(\boldsymbol{x}_n|c_n{=}q_j,\psi,\boldsymbol{\lambda}) + \mathrm{p}(\boldsymbol{x}_n|c_n{=}q_j,\psi,\boldsymbol{\lambda})\,\frac{\partial}{\partial\psi}\mathrm{p}\big(c_n{=}q_j,\boldsymbol{X}^c_{0:n-1}|\psi,\boldsymbol{\lambda}\big) \\
&= \mathrm{p}\big(c_n{=}q_j,\boldsymbol{X}^c_{0:n-1}|\psi,\boldsymbol{\lambda}\big)\,\frac{\partial}{\partial\psi}\mathrm{p}(\boldsymbol{x}_n|c_n{=}q_j,\psi,\boldsymbol{\lambda}) + \mathrm{p}(\boldsymbol{x}_n|c_n{=}q_j,\psi,\boldsymbol{\lambda})\,\frac{\partial}{\partial\psi}\mathrm{p}\big(c_n{=}q_j,\boldsymbol{X}^c_{0:n-1}|\psi,\boldsymbol{\lambda}\big) \\
&\doteq \Big(\sum_{q_i\in\mathcal{Q}} a_{j|i}\alpha_{n-1}(i,\psi)\Big)\Big(\frac{\partial}{\partial\psi}b_{j,\psi}(\boldsymbol{x}_n)\Big) + \Big(b_{j,\psi}(\boldsymbol{x}_n)\Big)\Big(\sum_{q_i\in\mathcal{Q}} a_{j|i}\frac{\partial}{\partial\psi}\alpha_{n-1}(i,\psi)\Big) \\
&= \Big\{(\boldsymbol{\Psi}'\boldsymbol{x}_n)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j{-}\boldsymbol{\Psi}\boldsymbol{x}_n)\Big\}\,\alpha_n(j,\psi) + b_{j,\psi}(\boldsymbol{x}_n)\sum_{q_i\in\mathcal{Q}} a_{j|i}\frac{\partial}{\partial\psi}\alpha_{n-1}(i,\psi).
\end{aligned}
$$

Bibliography

Abe, N., & Warmuth, M. K. (1992). On the computational complexity of approximating probability distributions by probabilistic automata. *Machine Learning, 9*.

Alissandrakis, A., Nehaniv, C. L., & Dautenhahn, K. (2002). Imitation with ALICE: Learning to imitate corresponding actions across dissimilar embodiments. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans, 32*.

Angluin, D. (1978). On the complexity of minimum inference of regular sets. *Information and Control, 39*, 337–350.

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation, 75*, 87–106.

Antsaklis, P. J., & Michel, A. N. (1997). *Linear systems*. McGraw-Hill.

Arkin, R. C. (1998). *Behavior-based robotics*. MIT Press.

Arkin, R. C., & Balch, T. R. (1997). AuRA: Principles and practice in review. *The International Journal of Robotics Research*, 92–112.

Asada, H., & Asari, Y. (1988). The direct teaching of tool manipulation skills via the impedance identification of human motions. *Proceedings of the IEEE International Conference on Robotics and Automation*.

Ashby, W. R. (1952). *Design for a brain*. John Wiley.

Bahlman, C., & Burkhardt, H. (2001). Measuring HMM similarity with the Bayes probability of error and its application to online handwriting recognition. *Proceedings of the 6th International Conference on Document Analysis and Recognition*.

Bentivegna, D. C., & Atkeson, C. G. (1999). Using primitives in learning from observation: A preliminary report. *Eighth AAAI Mobile Robot Competition Workshop* (pp. 40–46).

Bentivegna, D. C., Atkeson, C. G., & Cheng, G. (2003). Learning to select primitives and generate sub-goals from practice. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Bererton, C., Gordon, G., & Thrun, S. (2004). Auction mechanism design for multi-robot coordination. *Advances in Neural Information Processing Systems*.

Bertsekas, D. P. (1995). *Nonlinear programming*. Athena Scientific.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Betts, J. T., & Erb, S. O. (2003). Optimal low thrust trajectories to the moon. *SIAM Journal on Applied Dynamical Systems*, *2*, 144–170.

Billard, A., Epars, Y., Cheng, G., & Schaal, S. (2003). Discovering imitation strategies through categorization of multi-dimensional data. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Bilmes, J. A. (1997). *A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models* (Technical Report TR-97-021). Berkeley International Computer Science Institute.

Blondel, V. D., Bournezy, O., Koiranz, P., & Tsitsiklis, J. N. (2000). The stability of saturated linear dynamical systems is undecidable. *Lecture Notes in Computer Science*, *1770*.

Blondel, V. D., Theys, J., & Vladimirov, A. A. (2003). An elementary counterexample to the finiteness conjecture. *SIAM Journal on Matrix Analysis*, *24*.

Blondel, V. D., & Tsitsiklis, J. N. (2000). The boundedness of all products of a pair of matrices is undecidable. *Systems and Control Letters*, *41*.

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, *11*, 1–94.

Boyles, R. A. (1983). On the convergence of the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, *45*, 47–50.

Brand, M. (1999). An entropic estimator for structure discovery. *Advances in Neural Information Processing Systems* (pp. 723–729).

Carrasco, R. C., & Oncina, J. (1999). Learning deterministic regular grammars from stochastic samples in polynomial time. *Theoretical Informatics and Applications*, *33*, 1–19.

Chen, J., & Zelinsky, A. (2003). Programming by demonstration: Coping with suboptimal teaching actions. *International Journal of Robotics Research*, *22*.

Cheriyan, J. (1997). Randomized $\tilde{O}(M(|V|))$ algorithms for problems in matching theory. *SIAM Journal on Computing*, *26*.

Collins, J. D., Tullsen, D. M., Wang, H., & Shen, J. P. (2001). Dynamic speculative precomputation. *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture* (pp. 306–317).

Craig, J. J. (1989). *Introduction to robotics: Mechanics and control*. Addison Wesley. Second edition.

Dayan, P. (1992). The convergence of $TD(\lambda)$ for general λ. *Machine Learning*, *8*, 341–362.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological), 39*, 1–38.

Dixon, K. R., Dolan, J. M., & Khosla, P. K. (2004). Predictive robot programming: Theoretical and experimental analysis. *To appear in International Journal of Robotics Research*.

Dixon, K. R., & Khosla, P. K. (2003). Programming complex robot tasks by prediction: Experimental results. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Dixon, K. R., & Khosla, P. K. (2004a). Learning by observation with mobile robots: A computational approach. *To appear in Proceedings of the IEEE International Conference on Robotics and Automation*.

Dixon, K. R., & Khosla, P. K. (2004b). Trajectory representation using sequenced linear dynamical systems. *To appear in Proceedings of the IEEE International Conference on Robotics and Automation*.

Dorfman, R. (1969). An economic interpretation of optimal control theory. *American Economic Review, 59*, 817–831.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification*. Wiley-Interscience. Second edition.

Ephraim, Y., & Merhav, N. (2002). Hidden Markov processes. *IEEE Transactions on Information Theory, 48*.

Fagg, A. H., Lotspeich, D., & Bekey, G. A. (1994). A reinforcement-learning approach to reactive control policy design for autonomous robots. *Proceedings of the IEEE International Conference on Robotics and Automation*.

Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2*, 189–208.

Fod, A., Howard, A., & Matarić, M. J. (2002). A laser-based people tracker. *Proceedings of the IEEE International Conference on Robotics and Automation*.

Fraser, D. A. S. (1957). *Nonparametric methods in statistics*. John Wiley.

Friedrich, H., Münch, S., Dillmann, R., Bocionek, S., & Sassin, M. (1996). Robot programming by demonstration (RPD): Supporting the induction by human interaction. *Machine Learning, 23*, 163–189.

Friston, K. J., & Price, C. J. (2001). Dynamic representations and generative models of brain function. *Brain Research Bulletin, 54*, 275–285.

Gassiat, E., & Boucheron, S. (2003). Optimal error exponents in hidden Markov models order estimation. *IEEE Transactions on Information Theory, 48*.

Gat, E., Desai, R., Ivlev, R., Loch, J., & Miller, D. P. (1994). Behavior control for robotics exploration of planetary surfaces. *IEEE Transactions on Robotics and Automation, 10*.

Gaussier, P., Moga, S., Banquet, J.-P., & Quoy, M. (1998). From perception-action loops to imitation processes: A bottom-up approach of learning by imitation. *Applied Artificial Intelligence, 1*.

Gerkey, B. P., & Matarić, M. J. (2002). Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation, 18*.

Ghahramani, Z., & Roweis, S. T. (1999). Learning nonlinear dynamical systems using an EM algorithm. *Advances in Neural Information Processing Systems*.

Gillman, D., & Sipser, M. (1994). Inference and minimization of hidden Markov chains. *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory (COLT)*.

Gold, E. M. (1967). Language identification in the limit. *Information and Control, 10*, 447–474.

Golub, G. H., & Van Loan, C. F. (1996). *Matrix computations*. Johns Hopkins University Press. Third edition.

Hannaford, B., & Lee, P. (1991). Hidden Markov model analysis of force/torque information in telemanipulation. *International Journal of Robotics Research, 10*.

Hathaway, R. J. (1986). A constrained EM algorithm for univariate normal mixtures. *Journal of Statistical Computing and Simulation, 23*, 211–230.

Hayes, G., & Demiris, J. (1994). A robot controller using learning by imitation. *Proceedings of the 2nd International Symposium on Intelligent Robotic Systems*.

He, Q., & Shayman, M. A. (2000). Using reinforcement learning for proactive network fault management. *Proceedings of the International Conference on Communication Technologies*.

Hovland, G., Sikka, P., & McCarragher, B. (1996). Skill acquisition from human demonstration using a hidden Markov model. *Proceedings of the IEEE International Conference on Robotics and Automation*.

Hugues, L., & Drogoul, A. (2002). Synthesis of robot's behaviors from few examples. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Hundtofte, C. S., Hager, G. D., & Okamura, A. M. (2002). Building a task language for segmentation and recognition of user input to cooperative manipulation systems. *International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*.

Hwang, J.-H., Arkin, R. C., & Kwon, D.-S. (2003). Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Iba, S., Paredis, C. J., & Khosla, P. K. (2003). Intention aware interactive multi-modal robot programming. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Ijspeert, A. J., Nakanishi, J., & Schaal, S. (2001). Trajectory formation for imitation with nonlinear dynamical systems. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Ijspeert, A. J., Nakanishi, J., & Schaal, S. (2002). Movement imitation with nonlinear dynamical systems in humanoid robots. *Proceedings of the IEEE International Conference on Robotics and Automation*.

Intille, S. S., & Bobick, A. F. (1999). A framework for recognizing multi-agent action from visual evidence. *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Juang, B.-H., Levinson, S. E., & Sondhi, M. M. (1986). Maximum likelihood estimation for multivariate mixture observations of Markov chains. *IEEE Transactions on Information Theory*, *32*.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kaiser, M., & Dillman, R. (1996). Building elementary robot skills from human demonstration. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Kaiser, M., Dillman, R., Friedrich, H., Lin, I., Wallner, F., & Weckesser, P. (1996). Learning coordination skills in multi-agent systems. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Kaiser, M., Retey, A., & Dillman, R. (1995). Designing neural networks for adaptive control. *IEEE Conference on Decision and Control*.

Kaminka, G. A., Fidanboylu, M., Chang, A., & Veloso, M. M. (2002). Learning the sequential coordinated behavior of teams from observations. *Proeedings of the 2002 RoboCup Symposium*.

Kang, S. B., & Ikeuchi, K. (1995). A robot system that observes and replicates grasping tasks. *IEEE Fifth International Conference on Computer Vision* (pp. 1093–1099).

Kawamura, K., Peters, R. A., Wilkes, D. M., Alford, W. A., & Rogers, T. E. (2000). ISAC: Foundations in human-humanoid interaction. *IEEE Intelligent Systems*, *15*.

Kearns, M., & Valiant, L. (1994). Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, *41*, 67–95.

Kohonen, T. (1988). An introduction to neural computing. *Neural Networks*, *1*, 3–16.

Kreyszig, E. (1999). *Advanced engineering mathematics*. John Wiley. Eighth edition.

Krogh, A., Mian, I. S., & Haussler, D. (1994). A hidden Markov model that finds genes in *E. coli* DNA. *Nucleic Acids Research*, *22*, 4768–4778.

Kuniyoshi, Y., Inaba, M., & Inoue, H. (1994). Learning by watching: Reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, *10*.

Kurtz, R. B., & Kuper, G. H. (1986). *Toward a new era in U.S. manufacturing: The need for a national vision*. Manufacturing Studies Board and the National Research Council. Available from `http://books.nap.edu/books/0309036917/html/`.

Lenser, S., Bruce, J., & Veloso, M. (2002). A modular hierarchical behavior-based architecture. *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*.

Li, M., & Okamura, A. M. (2003). Recognition of operator motions for real-time assistance using virtual fixtures. *International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*.

Ljung, L. (1999). *System identification: Theory for the user*. Prentice Hall. Second edition.

Lusena, C., Goldsmith, J., & Mundhenk, M. (2001). Nonapproximability results for partially observable Markov decision processes. *Journal of Artificial Intelligence Research, 14*, 83–103.

Maes, P., & Brooks, R. A. (1990). Learning to coordinate behaviors. *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI)*.

Mahadevan, S., & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence, 55*, 311–365.

Mason, M. T. (2001). *Mechanics of robotic manipulation*. MIT Press.

Matarić, M. J. (1992). Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation, 8*, 304–312.

Matarić, M. J. (1997). Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence, 9*, 232–336.

Merhav, N., Gutman, M., & Ziv, J. (1989). On the estimation of the order of a Markov chain and universal data compresion. *IEEE Transactions on Information Theory, 35*.

Mitchell, T. M. (1997). *Machine learning*. McGraw Hill.

Montemerlo, M., Roy, N., & Thrun, S. (2003). Perspectives on standardization in mobile robot programming : The Carnegie Mellon navigation (CARMEN) toolkit. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Morgan, F. (2000). *Geometric measure theory: A beginner's guide*. Academic Press. Third edition.

Morrow, J. D., & Khosla, P. K. (1995). Sensorimotor primitives for robotic assembly skills. *Proceedings of the IEEE International Conference on Robotics and Automation*.

Nicolescu, M. N. (2003). *A framework for learning from demonstration, generalization and practice in human-robot domains*. Doctoral dissertation, Department of Computer Science, University of Southern California.

Nicolescu, M. N., & Matarić, M. J. (2001). Learning and interacting in human-robot domains. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans, 31*.

Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial optimization: Algorithms and complexity*. Mineola, New York: Dover Publications. Second edition.

Papoulis, A., & Pillai, S. U. (2002). *Probability, random variables and stochastic processes*. McGraw-Hill. Fourth edition.

Pineau, J., Gordon, G., & Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. *International Joint Conference on Artificial Intelligence*.

Pomerleau, D. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Copmutation, 3*, 88–97.

Pomerleau, D. (1996). Neural network vision for robot driving. *Early Visual Learning*. Oxford University Press.

Pook, P. K., & Ballard, D. H. (1992). Recognizing teleoperated manipulations. *Proceedings of the IEEE International Conference on Robotics and Automation*.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77*, 257–286.

Rabiner, L. R., & Juang, B.-H. (1986). An introduction to hidden Markov models. *IEEE ASSP Magazine, 3*, 4–16.

Rabiner, L. R., & Juang, B.-H. (1993). *Fundamentals of speech recognition*. Englewood Cliffs, NJ: Prentice Hall.

Redner, R. A., & Walker, H. F. (1984). Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review, 26*, 195–239.

Rivest, R. L., & Shapire, R. E. (1994). Diversity-based inference of finite automata. *Journal of the ACM, 41*.

Ron, D. (1995). *Automata learning and its applications*. Doctoral dissertation, The Hebrew University.

Ron, D., Singer, Y., & Tishby, N. (1998). On the learnability and usage of acyclic probabilistic finite automata. *Journal of Computer and System Sciences, 56*.

Rosch, E., Mervis, C. B., Gray, W. D., Johnson, D. M., & Boyes-Braem, P. (1976). Basic objects in natural categories. *Cognitive Psychology, 8*, 382–439.

Rudich, S. (1985). Inferring the structure of a Markov chain from its output. *IEEE Symposium on Foundations of Computer Science* (pp. 321–326).

Russell, S. J., & Norvig, P. (2002). *Artificial intelligence: A modern approach*. Prentice Hall. Second edition.

Rydén, T. (1995). Estimating the order of hidden Markov models. *Statistics, 26*, 345–354.

Schaal, S. (1997). Learning from demonstration. *Advances in Neural Information Processing Systems* (pp. 1040–1046).

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences, 3*, 233–242.

Schaal, S., Ijspeert, A., & Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transaction of the Royal Society of London: Series B, Bilogical Sciences, 358*, 537–547.

Seeger, M. (2000). *Learning with labeled and unlabeled data* (Technical Report). Institute for Adaptive and Neural Computation, University of Edinburgh.

Singh, R., Raj, B., & Stern, R. M. (2002). Automatic generation of sub-word units for speech recognition systems. *IEEE Transactions on Speech and Audio Processing, 10*, 89–99.

Skubic, M., & Volz, R. A. (2000). Acquiring robust, force-based assembly skills from human demonstration. *IEEE Transactions on Robotics and Automation, 16*.

Slotine, J.-J. E. (1984). Sliding controller design for nonlinear systems. *International Journal of Control, 40*.

Solis, J., Avizzano, C. A., & Bergamasco, M. (2002). Teaching to write Japanese characters using a haptic interface. *International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*.

Stengel, R. F. (1986). *Stochastic optimal control: Theory and applications*. Wiley-Interscience.

Stengel, R. F., Ghigliazza, R. M., Kulkarni, N. V., & Laplace, O. (2002). Optimal control of innate immune response. *Optimal Control Applications and Methods, 23*, 91–104.

Stolcke, A., & Omohundro, S. (1994a). Inducing probabilistic grammars by Bayesian model merging. *International Conference on Grammatical Inference*.

Stolcke, A., & Omohundro, S. M. (1994b). *Best-first model merging for hidden Markov model induction* (Technical Report TR-94-003). International Computer Science Institute, Berkeley, CA.

Strang, G. (1993). *Introduction to linear algebra*. Wellesley-Cambridge Press.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

Tesauro, G. S. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM, 38*, 58–68.

Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence, 99*, 21–71.

Thrun, S. (2001). Probabilistic algorithms in robotics. *AI Magazine, 21*, 93–109. Also appeared as Carnegie Mellon University School of Computer Science Technical Report CMU-CS-00-126.

Thrun, S., Fox, D., Burgard, W., & Dellaert, F. (2001). Robust Monte Carlo localization for mobile robots. *Artificial Intelligence, 128*.

Thrun, S., Langford, J. C., & Fox, D. (1999). Monte Carlo hidden Markov models: Learning non-parametric models of partially observable stochastic processes. *International Conference on Machine Learning*.

Tsitsiklis, J. N., & Van Roy, B. (1995). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control, 42*, 674–690.

Tso, S., & Liu, K. (1997). Demonstrated trajectory selection by hidden Markov model. *Proceedings of the IEEE International Conference on Robotics and Automation*.

van Nieuwstadt, M. J., & Murray, R. M. (1998). Real time trajectory generation for differentially flat systems. *International Journal of Robust and Nonlinear Control, 8*.

Watkins, C. J. C. H., & Dayan, P. (1992). Technical note: Q-learning. *Machine Learning*, *8*, 279–292.

Yang, J., Xu, Y., & Chen, C. S. (1994). Hidden Markov model approach to skill learning and its application to telerobotics. *IEEE Transactions on Robotics and Automation*, *10*, 621–631.

Yang, J., Xu, Y., & Chen, C. S. (1997). Human action learning via hidden Markov model. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, *27*.

Zentall, T., & B.G. Galef, J. (Eds.). (1988). *Social learning: Psychological and biological perspectives*. Hillsdale, NJ: Erlbaum.

Zhang, W., & Dietterich, T. G. (1996). High-perfoamce job-shop scheduling with a time-delay TD(λ) network. *Advances in Neural Information Processing Systems*.

Zilberstein, S., & Russell, S. (1996). Optimal composition of real-time systems. *Artificial Intelligence*, *82*, 181–213.

Zlot, R., Stentz, A., Dias, M. B., & Thayer, S. (2002). Multi-robot exploration controlled by a market economy. *Proceedings of the IEEE International Conference on Robotics and Automation*.