Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2018-008 September 1, 2007

Unsupervised Learning and Recognition of Physical Activity Plans

Shuonan Dong

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Unsupervised Learning and Recognition of Physical Activity Plans

by

Shuonan Dong

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

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2007

© Massachusetts Institute of Technology 2007. All rights reserved.

Author .....
Department of Aeronautics and Astronautics
August 23, 2007

Certified by .....
Brian C. Williams
Professor
Thesis Supervisor

Accepted by .....
David L. Darmofal
Associate Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Unsupervised Learning and Recognition of Physical Activity Plans

by

Shuonan Dong

Submitted to the Department of Aeronautics and Astronautics
on August 23, 2007, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This thesis desires to enable a new kind of interaction between humans and computational agents, such as robots or computers, by allowing the agent to anticipate and adapt to human intent. In the future, more robots may be deployed in situations that require collaboration with humans, such as scientific exploration, search and rescue, hospital assistance, and even domestic care. These situations require robots to work together with humans, as part of a team, rather than as a stand-alone tool. The intent recognition capability is necessary for computational agents to play a more collaborative role in human-robot interactions, moving beyond the standard master-slave relationship of humans and computers today.

We provide an innovative capability for recognizing human intent, through statistical plan learning and online recognition. We approach the plan learning problem by employing unsupervised learning to automatically determine the activities in a plan based on training data. The plan activities are described by a mixture of multivariate probability densities. The number of distributions in the mixture used to describe the data is assumed to be given. The training data trajectories are fed again through the activities' density distributions to determine each possible sequence of activities that make up a plan. These activity sequences are then summarized with temporal information in a temporal plan network, which consists of a network of all possible plans. Our approach to plan recognition begins with formulating the temporal plan network as a hidden Markov model. Next, we determine the most likely path using the Viterbi algorithm. Finally, we refer back to the temporal plan network to obtain predicted future activities.

Our research presents several innovations: First, we introduce a modified representation of temporal plan networks that incorporates probabilistic information into the state space and temporal representations. Second, we learn plans from actual data, such that the notion of an activity is not arbitrarily or manually defined, but is determined by the characteristics of the data. Third, we develop a recognition algorithm that can perform recognition continuously by making probabilistic updates. Finally, our recognizer not only identifies previously executed activities, but also pre-

dicts future activities based on the plan network.

We demonstrate the capabilities of our algorithms on motion capture data. Our results show that the plan learning algorithm is able to generate reasonable temporal plan networks, depending on the dimensions of the training data and the recognition resolution used. The plan recognition algorithm is also successful in recognizing the correct activity sequences in the temporal plan network corresponding to the observed test data.

Thesis Supervisor: Brian C. Williams
Title: Professor

Acknowledgments

First, I must thank my partner in life, Thomas Coffee, for spending much of his own thesis writing time to talk through the major stumbling blocks in my research with me, and for all the times he has cooked for me so I can have more time to work. His intellectual support, unconditional devotion, and contagious smile have made the worst of my days enjoyable, and the best of my days truly blissful.

I also tremendously appreciate my research advisor Professor Brian Williams for his key insights and encouragement. Additionally, Dr. Andreas Hofmann was effectively my second advisor, and Dr. Paul Robertson has been an excellent mentor. They have offered invaluable advice leading to the completion of this research. This thesis would not have been possible without their support.

To all my friends in the MERS lab, past and present, I say "thank you" for being such an important part of my life for the past couple of years. In no particular order, *thank you*, Shen Qu, for all our heart-to-heart conversations; Lars Blackmore, for advice on how to succeed in life and how to be more British; Hui Li, for making our lab such a fun place; Bobby Effinger, for being such a great co-TA; Julie Shah, for all the quals help and fun conversations over soldering; Seung Chung, for being a great mentor and such a fun person; Larry Bush, for providing juggling entertainment and advice on life; Paul Elliott, for being an amazing code wiz; Steve Block, for making the rovers dance; Tsoline Mikaelian, for patient help with model-based programming; Stephanie Gil, for providing smiles that brighten the day; and Thomas Léauté, for writing a good thesis that I can refer to with formatting issues.

My parents, Fengzhuo Hu and Yu Dong, deserve more applause than I can express here. They have been my motivators, my role models, and my best friends. They have made me who I am, and given me the opportunity to explore what I can become. They are the wind beneath my wings.

This research was in part funded by the National Science Foundation graduate fellowship. The data used in this thesis was obtained from mocap.cs.cmu.edu. The database was created with funding from NSF EIA-0196217.

"I am superior, sir, in many ways. But I would gladly give it up, to be human."

— Lt. Commander Data

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

This thesis desires to enable a new kind of interaction between humans and computational agents, such as robots or computers, by allowing the agent to anticipate and adapt to human intent. The intent recognition capability is necessary for agents to play a more collaborative role in human-computer interactions, moving beyond the standard master-slave relationship of humans and computers today. This thesis provides an enabling capability for recognizing human intent, through statistical plan learning and online recognition.

In this chapter, we will discuss the motivations for our research in Section 1.1 and provide a problem description in Section 1.2. Next, we will review previous literature related to intent recognition in Section 1.3, and present our approach to the problem in Section 1.4. Finally, we outline the roadmap for the rest of the thesis in Section 1.5.

1.1 Motivation

In the future, more computational agents such as robots or embedded computers may be deployed in situations that require interactions with humans, such as scientific exploration, search and rescue, hospital assistance, and even domestic care. These situations require robots to work collaboratively with humans, as part of a team, rather than as a stand-alone tool. For example, researchers at NASA Johnson Space Center are developing Robonaut, a humanoid robot, to assist astronauts during extra-

vehicular activities [4], like the camera changeout task executed during the Hubble Space Telescope repair missions. Robonaut will be able to handle lower-skilled and higher-risk tasks, enabling astronauts to work on more important and less dangerous problems. Currently, Robonaut is teleoperated by humans off-site, which may be unfavorable or infeasible for missions that are farther away or have long durations. Ultimately, we desire robots like Robonaut to be intelligent enough to autonomously interact with humans, so that they can play an independent role in a collaborative task.

A down to Earth example of robots that interact closely with humans is in the nursing home situation. Pineau et al. [42] have developed a mobile robot named Pearl to assist elderly individuals with mild cognitive and physical impairments, as well as support nurses in their daily activities. This robot specializes in reminding people of events and guiding them through their environments, both of which are particularly useful capabilities for a nursing home robot. Currently, Pearl uses synthesized speech and a speech recognizer to query and identify a person's status, such as whether he or she has taken medication yet. However, many elderly have difficulty understanding the robots synthesized speech and have trouble articulating a response that the robot can decipher. Therefore, we suspect that the robot's performance can be greatly enhanced by the addition of a physical activity plan recognition capability as described by this thesis. For example, plan recognition can enable the robot to identify a person's motion of raising a pill to his or her mouth.

Although humans use a combination of verbal and non-verbal cues when performing collaborative tasks together, we often do not verbalize our plans. Computational agents, however, need to infer intent, which exists in the form of courses of actions, represented as plans. Thus, an agent needs to infer the collaborator's intent from observing his or her motions [1, 21]. This thesis focuses on the non-verbal, physical motion cues. When a medical assistant sees a doctor extending his or her hand toward a scalpel, he or she may infer that the doctor intends to pick it up. Upon recognizing this intent, the medical assistant may hold up the scalpel and make it readily available. Similarly, during physical human-robot collaborative tasks, it is useful for the

robot to anticipate what the human is doing based on his or her physical motions, because the human is better assisted if the robot can anticipate need.

To enable this recognition of intent, the robot must first gain contextual knowledge by learning a plan of how the task might be performed. In our example, a medical assistant can infer that when the doctor reaches for a scalpel, he or she intends to pick it up, because the assistant has seen similar situations before—the assistant has *learned* that reaching for something often reflects the intent of picking it up. Of course, an assistant may also learn that the doctor sometimes reaches for an item to move it out of the way. Thus there are multiple plan options that an assistant must keep track of. An agent in a similar situation when assisting a human needs to learn the different plans that the human may perform. The agent represents all the different possible plans together in a *plan network*.

Next, during an online collaborative situation, the agent needs to recognize likely plans in the plan network from the new observations. When the medical assistant observes the current situation and infer that the doctor will most likely intends to pick up the tool, he or she is actually performing plan recognition by comparing the observations with the previously learned plan network to determine which plan in the plan network the doctor is most likely executing. In a similar way, a robot should be able to use the plan network to recognize what a human is currently doing, in order to anticipate the human's needs.

1.2 Problem Description

There are two main problems that this thesis focuses on: plan learning and plan recognition. An overview of the plan learning and recognition problems is illustrated in Figure 1-1.

Plan learning refers to the problem of deriving a plan network that describes a set of training motions. We represent the actions of a human by the combined motions of particular points on his or her body, so the training data are sequences of sampled pose states at points of interest on the human's body throughout the duration of

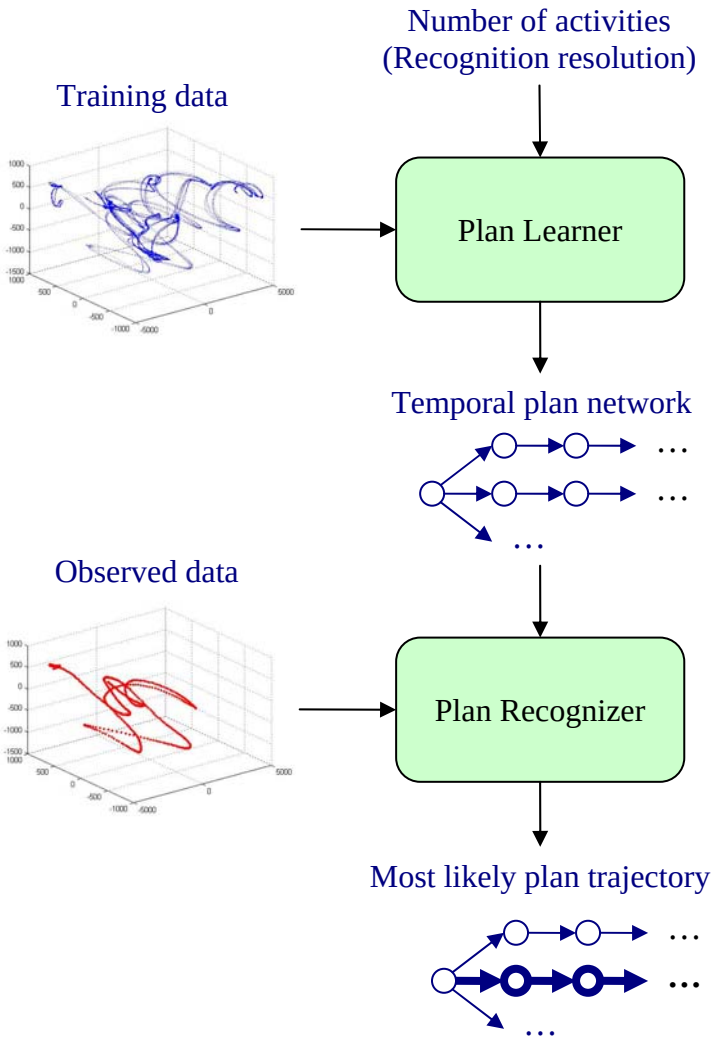


Figure 1-1: Overview of the plan learning and recognition inputs and outputs.

the motion. We want the output of the plan learning process to be a description of the space of all possible plans, or a plan network, which is comprised of smaller units called *activities*. To encode uncertainty of the motions, each activity is described by a probability density over the state space and is associated with a probabilistic activity duration.

Plan recognition refers to determining which activities the human collaborator has executed already, and predict which activities the human might perform next, given a plan network and observed data. The plan network is simply obtained from the output of the plan learning process. The observed data is a sequence of states that describe the motion of interest, similar in format to those of the training data. The key to the plan recognition problem is that the observed data does not have to be complete. By observing the human's motions for some small number of time steps, we would like the plan recognizer to identify (1) previously executed activities with corresponding schedules of when each activity began and ended, (2) the current activity and how long it has been executed, and (3) predicted future activities and most likely estimates of how long each might be executed.

1.3 Previous Research

In the past, many researchers have worked on the problem of enabling a computational agent to classify or recognize what a human is doing. Applications span from identifying gestures to recognizing handwritten letters to tracking a person's goals through dialogue, and techniques range from machine learning to Bayesian inference to plan decomposition methods. In this section, we discuss the work of other researchers and how they relate to our work.

Many researchers have focused on gesture recognition [34, 58, 11] to learn physical activities from data. Gesture recognition is the problem of extracting geometric data from visual inputs, deriving a 2-D or 3-D shape model, tracking motion segments, and then classifying the motions. Gesture recognition works very well for pre-separated data because most gesture recognition algorithms use supervised learning techniques.

For example, Bobick and Wilson's gesture recognition algorithm [11] can distinguish between segments of data capturing a hand wave versus other segments of data representing a pointing gesture. Similarly, Kadous [26] uses supervised learning to perform classification on multivariate gesture data. However, during real world tasks, pieces of data do not arrive in predetermined segments but rather a continuous stream. Therefore a plan learning algorithm not only needs to classify motions into activities, but should also automatically determine the separation between one activity and another. In contrast to the work in gesture recognition, this thesis utilizes unsupervised learning to handle automatic segmentation.

This thesis employs similar ideas as those from Barbič et al. [7], who use the unsupervised Expectation Maximization (EM) learning algorithm to identify activities from continuous physical data. We take their activity learning techniques a step further by also learning plans. Knowing a person's plan, we can not only figure out what activity a person is currently doing, but also make predictions as to what the person will do in the future. The added capability of anticipating future activities makes plan learning and recognition a more powerful tool than isolated activity learning.

Fox et al. [20] also present a learning algorithm similar to ours, used in an introspective robot behavior modeling application. They assume that the behavior of a robot when performing a task can be represented as a hidden Markov model (HMM), and they proceed to learn this model using a variance of EM. The resulting behavioral models allow the controller to reason about the robot behavior in the context of executing a task. Although the work employs similar techniques as ours, such as using EM to estimate the parameters of a stochastic model and Viterbi to determine most likely path in an HMM, its objectives are very different. Fox et al. are specifically interested in learning how a robot accomplishes a task, whereas we are interested in learning human motions, which are often more variable in execution. Furthermore, our work is based in the context of identifying a plan in order to infer intent, which requires anticipating many future activities to be executed beyond the point of current observations. This is why our work incorporates the use of a plan network. In contrast, Fox et al. are more interested in diagnosing a robot's behavior

from observations during the executing of a task.

Liao et al. [33] use learned models to infer human movements in the outdoor environment. They can detect when a person's behavior deviates from their normal pattern by evaluating the likelihood of an observed behavior in the context of a learned hierarchical Markov model. The model also allows them to predict a person's short term future movements, such as whether the person will turn left at the next street corner, and distance goals, such as if a person is going to the store. Similarly, Osentoski, Manfredi and Mahadevan [40] use a form of HMMs to model human motion through indoor environments to give a monitoring robot the capacity to predict and explain human activities. In both studies the human subject is abstracted to a point in two-dimensional space because the motion of interest is of a larger scale. Both studies use hierarchical HMMs, with the lowest level corresponding to a physical network of locations and higher levels corresponding to the tasks that are typically performed at these locations. Thus, these studies are concerned with large scale movement from place to place in a mapped environment and their emphasis is therefore different from our own.

Much plan recognition work have used qualitative representations of high and low level activities described by strings such as "clean room" or "sweep floor." Kautz and Allen's formal theory of plan recognition [27, 28] introduced a representation of plans in hierarchical constructs that can be decomposed into qualitative activities. Our plan representation differs because we keep track of the temporal ordering of activities and describe activities probabilistically. Following Kautz and Allen's plan representation, various researchers including Charniak and Goldman [14] and Pynadath and Wellman [46] formulate plans into Bayesian networks, and others like Bauer [9, 8] use Dempster-Shafer theory for recognition. Their representations are designed for activities which can be sensed through discrete inputs, such as clicking a button on a computer, whereas our representation is designed for activities with continuous data, such as physical motions. Also, when manually decomposing plans into activities, it is unclear which actions should be considered primitive and which should be considered higher-level. More importantly, during collaborative tasks, intent recogni-

tion should be performed continuously rather than only after some discrete activity has been completed. The value of intent recognition during collaboration is that it gives a collaborator the ability to estimate which activity a user is trying to execute, and then offer aid as necessary.

Another important aspect of plan recognition that has been missing in most recent literature is the consideration of temporal information. After Allen's presentation of temporal relations [3, 2], many researchers have worked on temporal planning, such as [39, 6, 57, 19]. However, few have incorporated temporal information into plan recognition. Suppose we observe a person reaching toward a lamp. If the reaching motion is swift, we may infer that she is turning on the switch, but if her hand is extended for a long period of time, we may instead conclude that she is trying to fix the light bulb. Avrahami-Zilberbrand, Kaminka and Zarosim [5] include qualitative temporal ordering in their plan recognition algorithm, but do not represent the metric temporal relations. In contrast, our research specifically encodes and relies on metric temporal relations to describe activity durations.

Avrahami-Zilberbrand et al, as well as others like Lesh, Rich and Sidner [32], use plan recipes to look up the possible relations between activities. These plan recipe libraries are pre-generated manually, which is undesirable because manually creating a recipe library is tedious and may be somewhat arbitrary, because the process is subject to the creator's opinion about what constitutes an activity. Instead, our work learns the recipe library, which in our case is the plan network, from actual data.

1.4 Approach and Innovations

This thesis provides an innovative capability for recognizing human intent, through statistical plan learning and online recognition. Our approach is to learn a probabilistic representation of possible plans, represented by a probabilistic *temporal plan network* (TPN), and then to track plans online using the TPN. A probabilistic TPN is comprised of all possible sequences of activities and corresponding activity duration distributions that represent the training data. The plan activities are described

by a mixture of multivariate probability densities over the state space to reflect the uncertainty of a human's motion. The number of distributions in the mixture used to describe the motion is assumed to be given. Activity durations are described by a probability distribution over time, since the exact timing of the activities that a human performs is not precisely known. To learn the activities, we employ unsupervised learning on the training data to automatically determine the mixture of activity distributions. To learn entire activity sequences, the training data trajectories are fed again through the activities' density distributions to determine each possible sequence of activities that make up a plan. These activity sequences are then summarized with the activity durations in a temporal plan network, which consists of a network of all possible plans.

In the medical assistant example, the plan is the motion of picking up something. The plan network consists of the different ways that the assistant has learned of how this motion can be executed. In one possible plan, one activity might be a reaching motion, described by the probabilistic region in the state space where this motion occurs. The duration associated with this activity may have an average of a few seconds, with some standard deviation. A sequence of several activities—for example, reach, grasp, pick up—may be necessary to describe one possible plan, and all the possible plans that the assistant knows form the plan network. For human assistants, the network of possible plans are learned throughout life; for robotic assistants, the plan network is learned through training. Once training is complete for some application, the assistant is ready to perform recognition.

Recognition involves tracking the most likely activity sequence through a TPN given a state observation sequence. Our approach first formulates the temporal plan network as a hidden Markov model. Then we determine the most likely path using the Viterbi algorithm. Finally, we refer back to the temporal plan network to obtain predicted future activities.

In our example, the recognition process allows the assistant to observe the doctor perform a reaching activity, determine the possible plans in the plan network that he or she may be performing, and conclude the most likely one. This recognized plan

allows the assistant to anticipate the doctor's desired future activities and help him or her perform them as needed.

This thesis presents several innovations: First, we introduce a modified representation of temporal plan networks that incorporates probabilistic information into the state space and temporal representations. Second, we learn plans from actual data, such that the notion of an activity is not manually defined, but is automatically determined by the characteristics of the data. Third, we develop a recognition algorithm that performs recognition continuously by making probabilistic updates. Finally, we not only recognize previously executed activities, but also can predict future activities based on the plan network.

1.5 Thesis Layout

We present this thesis in the following manner: In Section 2, we review some background material for the Expectation-Maximization algorithm and temporal plan networks, and in Section 3, we present the formal problem formulation. We describe the components of plan learning in Section 4, and we introduce the plan recognition algorithm in Section 5. We present the plan generation and recognition algorithms in detail in Section 6, and show results on motion capture data in Section 7. Finally, we present possible future advancements in Section 8. This concludes our introduction as we prepare to go into the details of our work.

Chapter 2

Background

This thesis assumes the knowledge of several existing algorithms and representations. Here, we review background material for data reduction using principal component analysis and unsupervised learning with the Expectation Maximization (EM) algorithm, and we discuss the existing representations of temporal plan networks.

2.1 Principal Component Analysis

The data that we deal with is highly multidimensional because we track the (x, y, z) positions of over thirty locations on the human body, producing on the order of one hundred dimensions. To prevent computational space limitations, we choose to only consider the features that most strongly distinguish the data. For example, a person's hand motions may be more important than foot motions during a reaching task. Principal component analysis (PCA) [25, 24] originally developed by Pearson [41], is a fairly well-known method of compressing data with some small but acceptable loss of accuracy. It is especially useful for data that has more dimensions than is easily analyzable graphically. Researchers have used principal component analysis in a myriad of different applications, including image processing [60, 53], economics [54], biology [13], and even atmospheric sciences [45].

In our work, the data represents the combined position information at different points on the body, sampled at some high frequency. We want to project the high

dimensional data onto a lower dimensional space defined by a set of principal components. PCA performs an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. PCA can be used to reduce the dimensionality in a data set by retaining those characteristics of the data set that contribute most to its variance, by keeping lower-order principal components and ignoring higher-order ones. Such low-order components often contain the "most important" aspects of the data.

The ordering of the principal components can be determined by the eigenvalues of the covariance matrix, and the components themselves are the eigenvectors. We can re-encode the data exactly as a function of the eigenvectors of the covariance matrix written in a feature vector V_p and the data given in principal components, which we call X_p. The encoding is given by $X = V_p^T X_p$, where the re-encoded X is assumed to have zero empirical mean. To encode the original data exactly, we would need all m eigenvectors so that the dimensionality of the data is not reduced. PCA creates an approximate function that achieves a dimensionality reduction by pruning a subset of the eigenvectors that least contribute to the error in the approximation. We can generate principal components from multidimensional data using the following method:

1. **Obtain data:** Assume $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T$ is a sequence of n data points, where each data point $\mathbf{x}_i = [x_1, x_2, \ldots, x_m]$ contains m dimensions. In other words,
$$X = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}.$$

2. **Center mean at zero:** To standardize the origin of the data, we recenter the data means at zero. Let $\mu_X = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$ be the mean of X. Then $X_{cen} =$

$[\mathbf{x}_1 - \mu_X,\ \mathbf{x}_2 - \mu_X,\ \ldots,\ \mathbf{x}_n - \mu_X]^T$. For clarity, we shall hereafter redefine X to mean X_{cen}, and $\mathbf{x}_i$ to mean $\mathbf{x}_i - \mu_X$, so that when we mention X, we actually refer the zero-centered data.

3. **Compute covariance matrix:** Intuitively, covariance tells us how the data values are correlated. The covariance has higher values when high data values are correlated. Thus the covariance can tell us the ordering of "importance" of the dimensions of the data, or which components are more "principal." We compute the covariance matrix as follows: Let $a, b \in \{1 \ldots m\}$ be dimensions of the data X, such that

$$X^a = \begin{bmatrix} x_{1a} \\ x_{2a} \\ \vdots \\ x_{na} \end{bmatrix},\ X^b = \begin{bmatrix} x_{1b} \\ x_{2b} \\ \vdots \\ x_{nb} \end{bmatrix}.$$

Let $\sigma_{ab} = \frac{1}{n-1} \sum_{i=1}^{n} \left(X_i^a - \mu_{X^a}\right) \left(X_i^b - \mu_{X^b}\right)$ be the covariance of the data for the dimensions a and b. Then the covariance matrix is

$$\Sigma_X = \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1m} \\ \vdots & & \vdots \\ \sigma_{m1} & \cdots & \sigma_{mm} \end{bmatrix}.$$

4. **Calculate the eigenvalues and eigenvectors of covariance matrix:** The set of eigenvectors $\mathbf{v}$ for Σ_X is defined as those vectors that when multiplied by Σ_X, result in a simple scaling λ of $\mathbf{v}$. Thus, $\Sigma_X \mathbf{v} = \lambda \mathbf{v}$. The scaling factors λ are the eigenvalues. To find the eigenvalues, we can solve $\det(\Sigma_X - \lambda I) = 0$, and to find the eigenvectors, we can solve $(\Sigma_X - \lambda I)\,\mathbf{v} = \mathbf{0}$. After finding the eigenvectors, we convert them into unit vectors $\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$ to unify the scaling.

5. **Form a feature vector and reduce dimensionality:** Eigenvalues indicate the importance of the dimension described by the associated eigenvector. Eigen-

vectors with larger eigenvalues describe more principal components of the data. In order to rank the eigenvectors in terms of importance, we sort the eigenvectors according to the corresponding eigenvalues in descending fashion to produce the feature vector $V = [\hat{\mathbf{v}}_{\max\lambda}, \ldots, \hat{\mathbf{v}}_{\min\lambda}]$. Now we can reduce the dimensionality by choosing only the p most important dimensions (where $p \leq m$) to be our feature vector: $V_p = [\hat{\mathbf{v}}_{\max\lambda}, \ldots, \hat{\mathbf{v}}_p]$.

6. **Deriving principal components of data:** We can now transform the data X with dimensions $n \times m$ onto the feature vector V_p with dimensions $m \times p$ to produce the principal components $X_p = XV_p$.

We have now generated the first p principal components of the data, so we have successfully compressed m dimensional data into the p most important dimensions. To retrieve the original data, we use $X_{retrieved} = V_p^T X_p + \mu_X$. Generally, the retrieved data will not match the original data exactly unless all the dimensions were included in the feature vector, i.e. $p = m$. However, usually just a few principal components are enough to produce retrieved data to within a small amount of error as the original.

2.2 Expectation Maximization for Unsupervised Learning

Our plan learning problem is to generate a plan network of temporal activities, given as input a sequence of states up to time t. Our approach is to encode the activities as a mixture of Gaussians. This section reviews an algorithm for learning a mixture of Gaussian model using unsupervised learning. The problem requires unsupervised learning because the cluster labels are not known a priori. Expectation Maximization is a well known and highly effective unsupervised learning algorithm that solves exactly this problem [48, 23, 10, 18]. The explanation of EM presented here loosely follows Russell and Norvig [48].

We will use a mixture of Gaussians to model the data because it allows us to describe multimodal distributions. This is important because we are interested in

motions that are centered around multiple locations. Assume the number of clusters, or classes, that describe the data is known to be k. Let y be a random variable denoting the class, which can have values of $1, \ldots, k$. Let $\mathbf{x}$ be the state vector at each data point with dimension p. Then the mixture distribution is given by

$$P(\mathbf{x}) = \sum_{j=1}^{k} P(y = j)\, P(\mathbf{x} \mid y = j).$$

For a mixture of Gaussian model, clusters are described by multivariate Gaussians with parameters

$$\Theta = \{\theta_1, \theta_2, \ldots, \theta_k\},$$

such that

$$\theta_j = \{w_j, \mu_j, \Sigma_j\},$$

where w_j are normalized weights of each class, μ_j are the means, and Σ_j are the covariances of each multivariate Gaussian distribution. Figure 2-1 shows an example

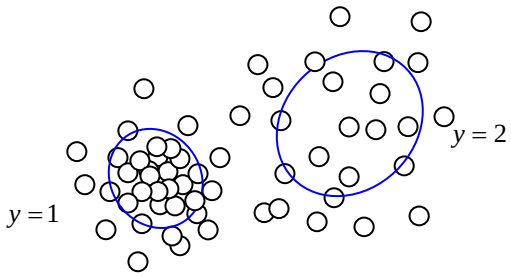


Figure 2-1: Mixture of two Gaussian clusters

of a mixture of two Gaussian clusters in 2-D. The ellipses are centered at the means of the Gaussian distribution, and the shape of the ellipses are governed by the covariances. Cluster 2 has a smaller weight than that of cluster 1, or $w_2 < w_1$ because fewer data points are part of that cluster.

The mixture of Gaussian distribution is given by

$$P(\mathbf{x} \mid \Theta) = \sum_{j=1}^{k} w_j \cdot P(\mathbf{x} \mid \mu_j, \Sigma_j) \tag{2.1}$$

$$= \sum_{j=1}^{k} w_j \cdot \frac{1}{(2\pi)^{p/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_j)^T \Sigma_j^{-1} (\mathbf{x} - \mu_j)\right), \tag{2.2}$$

where p is the dimension of $\mathbf{x}$, and $|\Sigma_j|$ is the determinant of Σ.

If we knew which cluster each data point belongs, i.e. the assignments of y, we can easily recover the Gaussian parameters of each cluster by selecting all the data with the same label $y = j$ and fitting the parameters of a Gaussian to them. On the other hand, if we knew the Gaussian parameters of each cluster Θ, we can, at least in a probabilistic sense, assign each data point y to a cluster. However, neither the assignments nor the cluster parameters are known. In this situation, the EM algorithm initially makes a guess of the parameters, then determines how good the guess is by computing the probability that each piece of data belongs to each class (Expectation). After that, the Gaussian parameters of each cluster is refitted to the the data, where each cluster is fitted to the entire data set with each point weighted by the probability that it belongs to that cluster (Maximization).

The Expectation and Maximization steps are formalized in Equations 2.3 – 2.11.

1. **E-step: Label Data.** Assume $\mathbf{X}$ is an array of state vector data of length n such that $\mathbf{X} = [\mathbf{x}_1 \ldots \mathbf{x}_n]^T$. Compute the probability that datum $\mathbf{x}_i$ belongs to class j using the update equation

$$\hat{p}(j \mid i) \leftarrow P(y_i = j \mid \mathbf{x}_i, \theta_j), \quad j = 1 \ldots k, \; i = 1 \ldots n. \tag{2.3}$$

Since $P(y_i = j \mid \mathbf{x}_i, \theta_j)$ cannot be readily determined, the actual calculation

must be performed using Bayes rule:

$$\hat{p}(j \mid i) \leftarrow P(y_i = j \mid \mathbf{x}_i, \theta_j) \tag{2.4}$$

$$= \alpha \cdot P(\mathbf{x}_i \mid y_i = j, \theta_j) \cdot P(y_i = j) \tag{2.5}$$

$$= \alpha \cdot P(\mathbf{x}_i \mid \mu_j, \Sigma_j) \cdot \hat{w}_j \tag{2.6}$$

$$= \alpha \cdot \hat{w}_j \cdot \frac{1}{(2\pi)^{p/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_j)^T \Sigma_j^{-1} (\mathbf{x} - \mu_j)\right), \tag{2.7}$$

where α is a normalization factor to enable the probabilities to sum to unity, $\hat{w}_j$ is the estimated weights derived from the Maximization step in Equation 2.10, p is the dimension of $\mathbf{x}$, and $|\Sigma_j|$ is the determinant of Σ.

2. **M-step: Update Parameters.** Compute the class distribution parameters $\hat{w}_j$, $\hat{\mu}_j$, and $\hat{\Sigma}_j$ using the update equations

$$\hat{n}_j \leftarrow \sum_{i=1}^{n} \hat{p}(j \mid i) \tag{2.8}$$

$$\hat{w}_j \leftarrow \frac{\hat{n}_j}{n} \tag{2.9}$$

$$\hat{\mu}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i=1}^{n} \hat{p}(j \mid i)\, \mathbf{x}_i \tag{2.10}$$

$$\hat{\Sigma}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i=1}^{n} \hat{p}(j \mid i)(\mathbf{x}_i - \hat{\mu}_j)(\mathbf{x}_i - \hat{\mu}_j)^T \tag{2.11}$$

The Expectation step (E-step) determines the probability that datum $\mathbf{x}_i$ belongs to class j, or $\hat{p}(j \mid i)$, while the Maximization step (M-step) uses it to find new parameters Θ by maximizing the log likelihood of the data. Wu [59] has proven that Expectation Maximization always increases or maintains the log likelihood of the data at every iteration. He has also proven that under certain conditions, the point of convergence can reach a local maximum in likelihood. In some cases, it is also possible for EM to reach a saddle point or local minimum. Figure 2-2 shows an example of Expectation Maximization working on a simple 2-D two class problem.

In the two class problem in Figure 2-2, we make an initial guess of the parameters

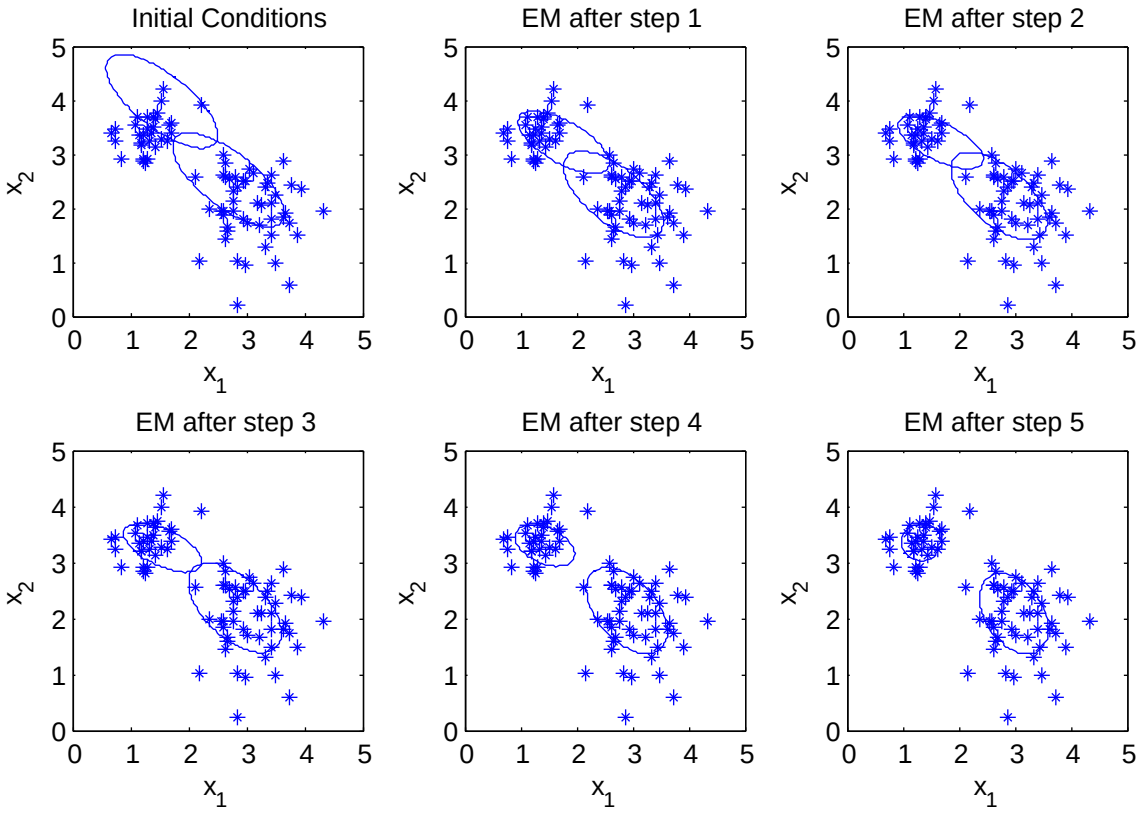


Figure 2-2: Example of EM on a 2-D two class problem.

of the two clusters. In this case, we choose two random points in the data as the Gaussian means and take the covariance of the entire data set as the covariance of each cluster. Every iteration of EM increases the log likelihood of the data so that the estimated cluster parameters (represented by the ellipses) describe the data better at each iteration.

2.3 Temporal Plan Networks

Our central problem is how to effectively encode the hypothesis space of plans that we are recognizing. Here, we review a representation of temporally flexible plans called temporal plan networks (TPNs). In terms of representing the hypothesis space, there are several features of TPNs that are important for our task. First, it can represent plans as a set of sequential activities. Second, activities are temporally extended.

Third, the start and end times of events are specified partially through qualitative and metric temporal constraints. Finally, and most importantly, a TPN is a compact encoding of a hypothesis space of possible plans, not just a representation of a single plan. The encoding is compact through the use of probabilistic choice operators.

This section reviews the representation of temporal plan networks (TPNs) as used in previous research. We also outline the concept of qualitative state plans (QSPs), which represent activities as constraints on the state space. The temporal plan networks used in this thesis are inspired by existing definitions of temporal plan networks and qualitative state plans.

2.3.1 Temporal Plan Networks

Central to plan recognition is a representation for the hypothesis space of plans. This representation should satisfy three requirements. First, the representation of plans should be expressive enough to capture the key properties of the plans that we want to represent. Second, it should be a compact encoding of all possible plans. Third, it should be effectively computable, that is, it should allow the recognizer to search efficiently through the space of possible plans. The temporal plan networks by Williams et al. [57, 29] provides these representation capabilities.

We give here an intuitive description of temporal plan networks based on Effinger [19], keeping in mind that we will be using TPNs for different purposes and different capabilities. For a formal definition of TPNs used in this thesis, please see Section 3.5.1.

Generally, a temporal plan network encompasses a simple temporal network, so it is also a directed graph with events V and edges E labeled with temporal constraints $[l, u]$, where l is the lower temporal bound and u is the upper temporal bound. In addition, an edge in a TPN can be associated with an activity, which, in planning applications, may represent a command, state assertion, state request, or timing constraint. In the recognition application of this thesis, an activity will describe a particular region of motion. In addition to parallel and sequential activities, temporal plan networks can also represent different possible activity combinations from which

one can be chosen. The event from which different possible branches of the plan stems is called a choice event. A TPN can have multiple and nested choice events. Being able to represent choices is a very important quality of TPNs for plan recognition because we can formulate the plan recognition problem as determining which choice(s) a human made when executing a task.

An example temporal plan network is shown in Figure 2-3. This example depicts the plan of someone picking up a hammer and either striking a nail or extracting a nail. The first activity of the event is picking up a hammer, which lasts between 1 and 6 time units. Following the completion of that activity at event v_1, the person simultaneously holds the nail in place while either striking it or extracting it. The choice event v_2 is illustrated by a double encirclement of the event. Edges without specific activities indicate no-ops.

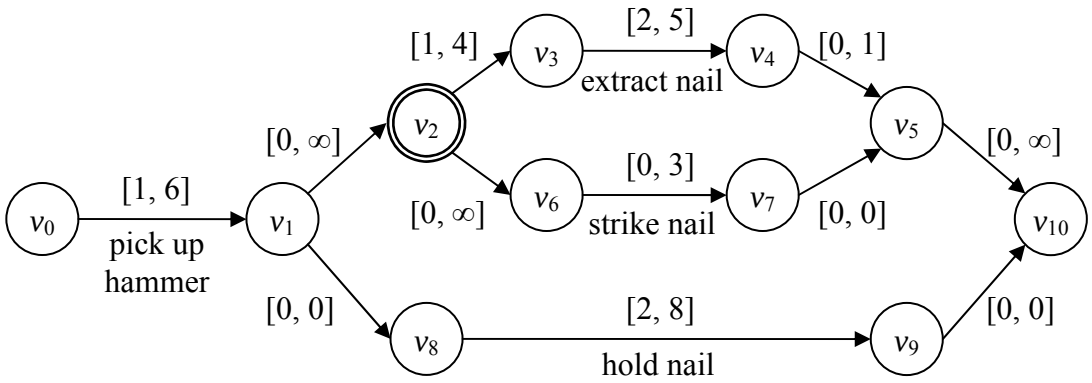


Figure 2-3: Example temporal plan network. The double circle indicates a choice event.

2.3.2 Qualitative State Plans

Our representation of activities is motivated by the activity representation in qualitative state plans (QSP). Researchers like Léauté [31] who are interested in robust control of agile systems have developed qualitative state plans (QSP) to describe the desired motion of a plant at a high level for the operator. Hofmann [22] has also utilized qualitative state plans toward controlling a simulated humanoid biped. In a QSP, state trajectories are specified qualitatively as a series of feasible state regions

rather than sequence of specific states.

Activities in a QSP specify qualitative constraints on the state of the plant. For example, a QSP activity $a_{ij} = \langle v_i, v_j, c_{ij} \rangle$ is located between events v_i and v_j in the plan, and is associated with the constraints c_{ij} on the variable $\mathbf{x}$, which is a tuple consisting of a state variable $\mathbf{s}$ and control input $\mathbf{u}$. Recall that a schedule T is an assignment $T : V \mapsto \mathbb{R}$ of an event to a specific execution time. The state constraints c_{ij} can include a start region R_S, such that $\mathbf{x}(T(v_i)) \in R_S$, an end region R_E such that $\mathbf{x}(T(v_j)) \in R_E$, a remain-in region $R_\forall$ such that $\forall t \in [T(v_i), T(v_j)], \mathbf{x}(t) \in R_\forall$, and a go-through region $R_\exists$ such that $\exists t \in [T(v_i), T(v_j)]$ for which $\mathbf{x}(t) \in R_\exists$.

The structure of activities in a QSP is the motivator for the activities in the temporal plan networks for plan learning in this thesis. In our problem, there are no control inputs $\mathbf{u}$, so $\mathbf{x}$ only consists of the state variable $\mathbf{s}$. Then the activities we eventually learn from the observed states can be described as some region of the state space, similar in concept to those in QSPs. The difference is the description of that region. For a formal definition of an activity in a plan that is used in this thesis, see Section 3.5.3.

We have now completed our review of existing techniques that we use in our work. The next chapter will present the formal statement of the problem we will address.

Chapter 3

Problem Formulation

This chapter describes the problem of plan learning and recognition more formally. We first present a description of the environment from which our data is obtained in Section 3.1 to ground the kinds of problems we focus on. Then we discuss the assumptions of the problem and present a motivating example. Next, we formally define the input data sequences in Section 3.4. The definition of a TPN is given in Section 3.5, followed by the plan learning and plan recognition problem formulations in Sections 3.6 and 3.7, respectively.

3.1 Motion Capture Setup

Our work uses the Vicon Motion Systems data obtained from Carnegie Mellon University Graphics Lab Motion Capture Database [44]. This motion capture system has eight cameras, each recording 1000×1000 pixel resolution at 120Hz sample rate. The marker set consists of 35 14mm markers placed around the whole body, the locations of which are shown in Figure 3-1. All 34 symmetric markers are used in the learning and recognition process, giving data in 102 dimensions, which is why we use principal component analysis to reduce the dimensions of the data. Each test subject has a calibrated skeleton model, and the motions can be visualized, like the dance move in Figure 3-2.

Our goal is the following: Given training data like that obtained from the motion

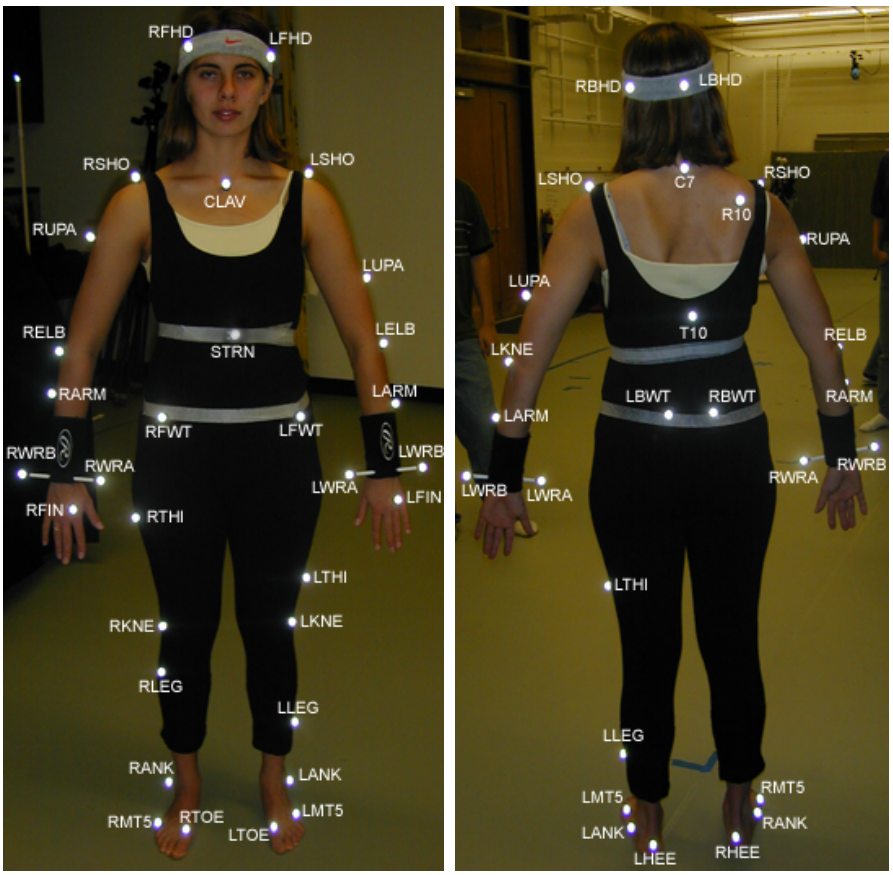


Figure 3-1: Marker placement in front and back

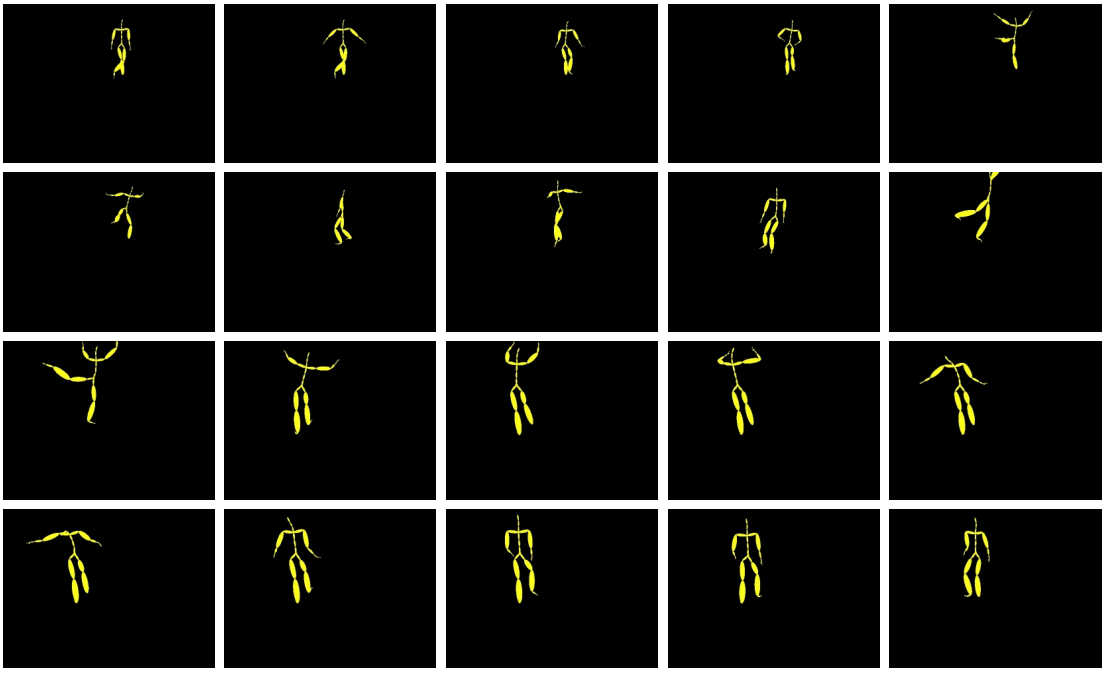


Figure 3-2: Example dance motion data (displayed in horizontal order). The motions are attitude/arabesque (frames 1-9), jete en tourant (frames 9-12), and bending back(frames 12-20).

in Figure 3-2, we would like to generate a *temporal plan network* that best describes the range of possible *activities*. Then with the temporal plan network and new observations, we want to determine the most likely sequence of activities in the plan, or *recognized activity sequence*, to which the observations correspond.

3.2 Assumptions

There are several assumptions we will make when approaching the plan learning and recognition problems to ensure that our problem has an appropriate scope. In general, we would like to focus on the problem addressed in the thesis and allow other researchers to work on interesting, related, but non-core issues relating to our problem.

We will first assume that the data provided to us is in a format readily usable by our algorithms. Specifically, we assume that the position vectors of particular points on the human subject can be obtained at some sampling frequency. This thesis uses

motion capture data which is readily available in this format. However, in practice, a robotic agent may not have the capability to obtain such high resolution motion capture data, and may need to rely on more conventional sensing techniques such as cameras or laser range finders. The image processing community has done much research to extract geometric information from image data [38, 21]. We trust that researchers like Sigal [50] will ensure that extracting 3D position information from images is not beyond the capabilities of image processing. Thus we will demonstrate the capabilities of our research only on motion capture data and assume the applicability to other input formats.

We also assume that the frequency with which we sample the motion data is faster than the rate at which a human subject can move, so that no important information is lost during the data sampling process. Furthermore, we assume that sampling is done at a constant frequency, so that the number of data points we sample directly scales as the time lapsed during sampling.

During plan learning, we assume that a recognition resolution is known a priori, meaning that we know beforehand whether we will be distinguishing motions that are very similar or relatively different. This plays a key role in the plan learning process because a higher recognition resolution is necessary for distinguishing more similar motions, while a smaller recognition resolution is more optimal for distinguishing very different motions. In plan learning, the recognition resolution is the number of unique activity clusters that will be learned, which we denote as k. Section 8.1.3 gives some insights on how the recognition resolution might be automatically learned in the future.

3.3 Astronaut Robotic Assistant Example

Now that we have stated our assumptions, we can provide a better idea of the problem we will be solving with an example.

When an astronaut performs some task in space, such as in a telescope repair mission, he or she follows a set of procedures outlined by the operations team. As an

example, Figure 3-3 shows a portion of the operations procedures for the wide-field planetary camera changeout during the Hubble Space Telescope (HST) repair and maintenance mission in 1993 [36]. Suppose we would like to design a robotic agent to inform the astronaut of which activity he or she should be completing, and to ensure that each activity is completed correctly. The agent would first need to know what the task procedures are, and in what ways they can be performed by an astronaut. We describe all the ways a person can complete a task procedure as a *plan*.

Goal: Replace the original WFPC I instrument in the HST axial bay (axis - V3) with the second generation WFPC II in the Space Shuttle cargo bay WFPC radial site.

In this exercise you will

- Deploy the forward and aft temporary parking fixtures (TPF) with attached handholds.
- Install the handhold guide studs on each of the WFPCs.
- Adjust the (-V2) scuffplate and open the FGS#3 doors to access the HST WFPC indicator lights.
- Retrieve, translate, and temporarily stow the WFPC I with the aft handhold.
- Open and access the radial site.
- Translate the WFPC II with the forward handhold from the radial site to the HST for installation.
- Install the WFPC II in the HST, close the FGS#3 doors, secure the (+V2) scuffplate, and stow the forward handhold on the forward TPF.
- Translate the WFPC I from the aft TPF to the radial site, stow for return to Earth, and close out the radial site.
- Return the aft handhold to the aft TPF and return both TPFs to their original stowed positions.

Beginning position: At the start of the WFPC tasks you will be facing the aft end of the Shuttle cargo bay at a position approximately in line with the airlock hatch.

Figure 3-3: Operations for the wide-field planetary camera changeout during Hubble Space Telescope repair and maintenance mission

Before a space mission, astronauts and ground support crew go through a series of training sessions to get familiarized with the environment and task, sometimes through immersive virtual environments like the one developed by Loftin and Kenney to train over 100 members of the ground-support flight team before the HST mission [36]. We propose that a robotic agent can learn an astronaut's plan by collecting data during this type of ground training before the mission. We would like the plan

to contain both spatial and temporal information. As mentioned, we assume the data consists of position vectors of certain points on the astronaut recorded at some constant and fast sampling frequency. The first problem we address in this thesis is how to learn a plan from training data.

Now suppose the agent has already learned the task procedure plan, and is assisting an astronaut during a mission. To effectively monitor the progress of the astronaut, the robotic agent must be able to identify which method the astronaut is applying, what activities he or she has completed already, which activity he or she is in the process of performing, and very importantly, what activities the astronaut will need to complete in the future. The second problem we address in this thesis is how to recognize the correct activity sequence that the human has executed, and using the information in the plan, predict future activities that the human should perform.

3.4 Definition of a Data Sequence

A learning task begins with data. The motion capture data used in this research consists of (x, y, z) position data of l different markers on the body, giving a total of $m = 3 \cdot l$ states at each time step. We can now clearly state the input data sequence to the problem in Definition 1.

Definition 1 *A data sequence* $\mathbf{X}$ *is a sequence of* n *poses* $\mathbf{x}$*, where* n *is the number of data samples taken, and each pose state* $\mathbf{x} = [x_1, \ldots, x_m]$ *is a vector of the* m *position data. We represent* $\mathbf{X}$ *as an* $n \times m$ *matrix of position data over time, or* $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T$.

By combining the three position states of each marker of interest into one state vector, the training data can represent combined motions. For example, a person might be holding something with the left hand while reaching for something with the right. These parallel activities can be derived either by learning over the combined states of both hands, or by learning each hand and combining the resulting activities. The choice depends on whether the motion of one marker is considered independent

of the motion of another marker or not. In our application, we assume a person uses all parts of their body together to perform a task, so the data from all markers of interest are simultaneously considered together in one state vector. This approach limits our representation of the temporal plan network because parallel activities will not be considered separately. However, when learning the activities of one person performing a specific task, such as a dance move, this approach is acceptable and appropriate because a person generally will not be multitasking. See Section 8.1.1 on future advancements for a discussion of potentially representing parallel activities separately.

3.5 Temporal Plan Network Redefined

We gave a brief description of the current notion of a temporal plan network (TPN) in Section 2.3.1. This thesis does not use all the current capabilities of a TPN, an example of which is the lack of parallel activities mentioned in Section 3.4, but on the other hand expands on certain aspects of it. One major addition is that the activities in a TPN are represented as probabilistic regions in the state space, motivated in concept by previous work on qualitative state plans discussed in Section 2.3.2. Another change is that instead of representing the durations of activities as bounded temporal constraints as they currently are in a TPN, the activity durations are also represented by probabilistic distributions.

With the additions, the output of plan learning is a particular kind of TPN in which one choice event expands into all different plan trajectories. In the implementation presented in this thesis, the only choice event is in the beginning, and plan trajectories that are different even in the slightest are represented separately. Although this is a limitation in our implementation, the TPN is still valid and the results of plan recognition are the same. For a discussion of how we might combine the representation of similar partial trajectories of a TPN in the future, see Section 8.1.2.

Another capability of the current TPN representation that is not used in this thesis is the ability to represent temporal information between any two events. In

the plan recognition application, temporal information need only be associated with activities, between adjacent events.

3.5.1 General definition of a TPN used in this thesis

We now provide the definition of a temporal plan network that will be used throughout this thesis in Definition 2.

Definition 2 *A **temporal plan network** $P = \langle \mathcal{E}, \mathcal{E}_{ch}, \mathcal{A}, \mathcal{D} \rangle$ specifies an evolution of the observed states over time, and is defined by a set $\mathcal{E}$ of all discrete events, a set $\mathcal{E}_{ch} \subseteq \mathcal{E}$ of choice events, a set $\mathcal{A}$ of activities defining probabilistic regions over the observed states, and a set $\mathcal{D}$ of temporal distributions between two events corresponding to an activity.*

We illustrate a temporal plan network diagrammatically by an acyclic directed graph in which the discrete events in $\mathcal{E}$ are represented by nodes drawn as circles, choice events as double circles, and activities as numerals encased in angle brackets along graph edges. Activity durations are represented by the duration distributions on graph edges. An example TPN diagram is shown in Figure 4-6.

3.5.2 Definition of a schedule

Since we do not know a priori when exactly a person will reach some event in the plan, we must encode the events to be temporally flexible. To enable this, we use a schedule described in Definition 3 to represent the time when an event occurs. This is the same definition of a schedule as that for simple temporal networks [17] and qualitative state plans [31].

Definition 3 *A **schedule** T for a temporal plan network P is an assignment $T : \mathcal{E} \mapsto R$ of observed execution times to all the events in P.*

In this thesis, we assume that schedules are measured relative to the first event in the plan, which we call event e_0. Thus $T(e_0) = 0$.

3.5.3 Definition of an activity

Definition 2 describes a temporal plan network as $P = \langle \mathcal{E}, \mathcal{E}_{ch}, \mathcal{A}, \mathcal{D} \rangle$, where $\mathcal{A}$ is a set of activities that describe the region of motion in the state space probabilistically.

Definition 4 *An **activity** $a = \langle e_S, e_E, r \rangle$, where $a \in \mathcal{A}$, has an associated start event e_S, an end event e_E, and a probabilistic region in the state space r.*

As Definition 4 states, every activity is between two events and is described by some probabilistic region in the state space. The set of all activities $\mathcal{A}$ describes the mixture of densities in the state space where the motion is observed. If we use a mixture of Gaussians to describe the motion in the state space, then the activity region is described by Definition 5.

Definition 5 *An **activity region** $r = N(\mu, \Sigma)$ is a multivariate Gaussian in the state space defined by the mean μ and covariance Σ.*

For example, if an astronaut were performing the procedure to "retrieve, translate, and temporarily stow the WFPC I with the aft handhold" in Figure 3-3, he or she may reach out and move the camera to the aft handhold position. The activity regions might be along the line of the reaching motion, around the location of the camera, and back near the location of the aft handhold.

Our use of an activity is significantly different from that in TPNs of previous work, where an activity may represent a command, state assertion, state request, or timing constraint. Previous work on temporal plan networks have been motivated by planning applications, where the activities are known beforehand. In recognition applications, however, the robotic agent knows nothing about individual activities that a human is performing, so activities must be learned from the human's physical motions.

3.5.4 Definition of a duration

All temporal information in a temporal plan network is encoded in $\mathcal{D}$, which contains a set of durations described in Definition 6. Technically, a duration can span between

any two events in the TPN, although in our application they will only span adjacent events corresponding to an activity. Every activity has a duration.

Definition 6 *A **duration** $d = \langle e_S, e_E, g \rangle$, where $d \in \mathcal{D}$, has an associated start event e_S, an end event e_E, and a temporal probabilistic distribution g.*

The duration is represented as a distribution of times. This is significantly different from the temporal constraints in the TPNs used in previous work. As mentioned in Section 2.3.1, temporal constraints are represented by a time interval $[l, u]$, where l is the lower temporal bound and u is the upper temporal bound. In the recognition application, however, we cannot use absolute intervals to represent possible activity durations observed in a human's motion. Instead, we represent the durations as distributions over time, as stated in Definition 7. This representation of duration is not as restricting as the temporal constraints in previous TPNs because it allows an activity to have any arbitrary duration, although certain durations are more likely than others.

Definition 7 *A **duration distribution** $g = \Gamma(k, \theta)$ is a gamma distribution of durations governed by the shape k and scale θ of the distribution.*

We chose a gamma distribution to represent durations instead of a Gaussian distribution to ensure that all durations have non-negative values.

3.5.5 Definition of a choice event

A temporal plan network is different from a predetermined list of activities because it encodes all the different ways a task can be completed. This is especially important in recognition applications because the execution is done by a human subject who may choose to perform a task in one of a myriad of different ways. To encode the general methods of executing a plan, we employ a choice event as described in Definition 8 to represent the point at which multiple methods methods branch.

Definition 8 *A **choice event** $e_{ch} \in \mathcal{E}_{ch}$ is an event such that of all activities $a_j = \langle e_{Sj}, e_{Ej}, r_j, d_j \rangle$ for which the start event is the choice event, or $e_{Sj} = e_{ch}$, only one of these activities will be recognized.*

3.6 Definition of a Plan Learning Problem

The problem of plan learning is presented in Definition 9. It is in part motivated by our desire to avoid manually generating a plan recipe, as in the case of [5, 32]. Instead, we would like to automatically learn the plan network from training data. The plan learning algorithm will perform unsupervised learning on training data to eventually generate a temporal plan network. The detailed approach of plan learning will be discussed in Chapter 4.

Definition 9 *Given a set of C training data sequences $X = \{\mathbf{X}_1, \ldots, \mathbf{X}_C\}$ and the recognition resolution k, the **plan learning problem** outputs a temporal plan network $P = \langle \mathcal{E}, \mathcal{E}_{ch}, \mathcal{A}, \mathcal{D} \rangle$, where each activity $a \in \mathcal{A}$ encompasses one of k unique activity regions that describe the set of training data sequences to a convergence factor of δ.*

In practice, the only choice event is the first event, or $\mathcal{E}_{ch} = \{e_0\}$, in the learned temporal plan network because all paths, even if only slightly different, are treated independently, as mentioned in Section 3.5. Furthermore, the set of durations $\mathcal{D}$ has a one-to-one correspondence to the set of activities $\mathcal{A}$ because the plan learner does not record time lapses between any two arbitrary events, only ones corresponding to an activity.

3.7 Definition of a Plan Recognition Problem

After obtaining a temporal plan network, a robotic agent can now perform recognition on newly observed data. The plan recognition problem is stated in Definition 10. The detailed approach of plan recognition will be discussed in Chapter 5.

Definition 10 *Given a temporal plan network* $P = \langle \mathcal{E}, \mathcal{E}_{ch}, \mathcal{A}, \mathcal{D} \rangle$ *and a newly observed data sequence* $\mathbf{X}_{obs} = [\mathbf{x}_1, \ldots, \mathbf{x}_{n_{obs}}]^T$*, the* ***plan recognition problem*** *produces a recognized activity sequence* R *and a predicted activity sequence* B *that describes the most likely past and future activities corresponding to* $\mathbf{X}_{obs}$*.*

The key to the plan recognition problem is that the observed data does not have to be complete. With data observed to some small number of time steps, n_{obs}, we would like to be able to identify (1) previously executed activities with corresponding schedules of when each activity began and ended, (2) the current activity and how long it has been executed, and (3) predicted future activities and most likely estimates of how long each might be executed. We represent this information by a *recognized activity sequence* R described in Definition 11 and a *predicted activity sequence* B described in Definition 12.

Definition 11 *A* ***recognized activity sequence*** *is a tuple* $R = \langle \mathcal{E}_R, \mathcal{A}_R, T_R \rangle$*, where* $\mathcal{A}_R$ *are the recognized activities with corresponding events* $\mathcal{E}_R$*, and a schedule* T_R *that determines the recognized execution times of each event. The schedule of the last recognized event,* $T(e_{last})$ *is equal to the time lapse between the first and last data points in the observed data sequence.*

The recognized activity sequence corresponds to a partial trajectory of the TPN when the observed data is not complete. The schedule of the recognized activity sequence is obtained from the observed data sequence, given the duration distributions of the activities in the TPN. Since the observed data may not be complete, it can be terminated in the middle of some activity. The recognized activity sequence encodes the schedule of the last activity's end event as the time of the last observed datum. Thus the recognized activity sequence provides (1) the previously executed activities with corresponding event schedules, and (2) the current activity and how long it has been executed.

Definition 12 *A* ***predicted activity sequence*** *is a tuple* $B = \langle \mathcal{E}_B, \mathcal{A}_B, T_B \rangle$*, where* $\mathcal{A}_B$ *are the recognized and predicted activities with corresponding events* $\mathcal{E}_B$*, and a*

schedule T_B that determines the recognized and predicted execution times of each event. The schedule of the events in the predicted activity sequence is determined by the schedule of the recognized activity sequence T_R and the temporal plan network.

The predicted activity sequence encompasses all the information in the recognized activity sequence, in addition to which it uses the corresponding trajectory in the TPN to predict future activities. The schedules of previously executed activities are the same as those in the recognized activity sequence. Schedules of future activities reflect the most likely durations of those activities in the TPN. The predicted activity sequence allows a robotic agent to anticipate a human's future motions.

Chapter 4

Statistical Plan Learning

To perform plan recognition, a robot needs to acquire a knowledge base of plans that can be compared against data. To acquire this data, we must either directly tell a robot the information we know, or it will need to learn the plans itself. Some researchers have suggested providing a list of hundreds of thousands of common sense actions to help computers to become smarter [35]. However, it would be extremely tedious for a user to manually generate all task plans. Hence, we prefer the robot to learn the knowledge of these plan from training data.

4.1 Overview

This chapter presents our method for plan learning. An overview of the plan learning process is illustrated in Figure 4-1. In the beginning, unsupervised learning is used to cluster the training data into the number of activities defined by the recognition resolution. Each activity is described by a multivariate Gaussian $N(\mu_j, \Sigma_j)$ in the state space. The training data sets are then used to determine the correct sequences of these activities. Finally, the activity sequences are augmented with temporal information in a temporal plan network. This approach is different from past gesture recognition work in that we use unsupervised learning, whereas gesture recognition generally utilizes supervised learning, as in [26]. We also go beyond Barbič et al.'s work [7] because we not only learn activities, but we also record the sequences and

durations of activities.

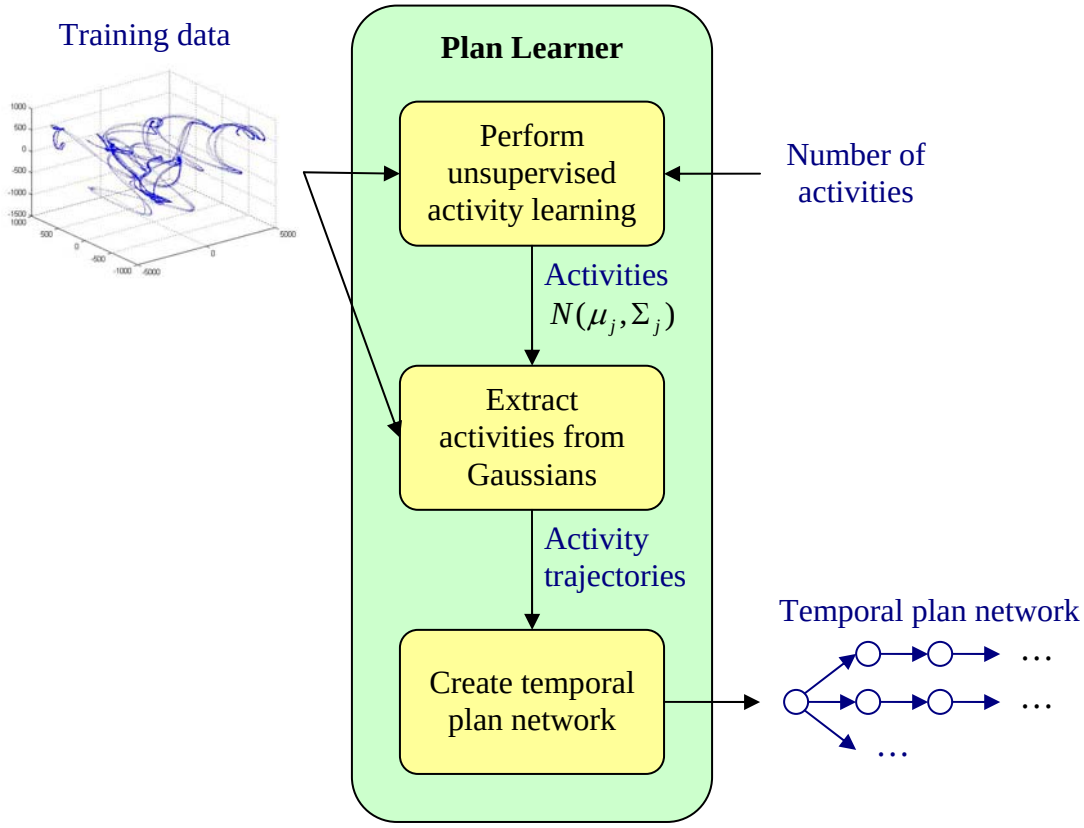


Figure 4-1: Overview of the plan learning process

We will describe the input training data in Section 4.2 and present the principal component analysis to reduce the dimensionality of the data. Section 4.3 then discusses how the EM algorithm is used to learn activities from data, and Section 4.4 describes the method by which activity sequences are extracted with associated duration distributions. Finally, Section 4.5 discusses how a TPN is created from activity trajectories.

4.2 Formatting the Training Data to Be Used in Learning

4.2.1 Motion Capture Data

Training data for the plan learner consists of multidimensional continuous state observations over time sampled at some constant frequency. This data can be obtained by a variety of sensors. In this thesis, we use Vicon Motion Systems data obtained from Carnegie Mellon University Graphics Lab Motion Capture Database, the same database used and described in [44]. Motion capture data measures the (x, y, z) position states of various markers on the body. The motion capture system used for the CMU database has eight cameras, each recording 1000×1000 pixel resolution at 120Hz sampling rate. The marker set consists of 35 14mm markers placed around the whole body, of which 34 symmetric marker placements are used in this thesis.

We combine the three position states of each marker of interest into one state vector. For example, the state vector for two markers is $\mathbf{x} = [x_1, y_1, z_1, x_2, y_2, z_2]$. We describe a state vector as $\mathbf{x}_i = [x_1, x_2, \ldots, x_m]$, where m is the dimension of the state vector. A training data sequence contains n state vectors measured at some sampling frequency, and is represented as $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T$. The dimension m of the state vector must be the same in each training data sequence, but the lengths n of each training sequence do not have to be the same.

Certain adjustments may need to be applied to the raw data to prepare it for use. Specifically, we ensure that the training data sequences are all properly scaled and co-originated. Proper scaling refers to the (x, y, z) position measurements being of the same units. Proper co-origination refers to aligning the initial positions in each training sequence to be the same. We make these adjustments without loss of generality, since they do not change important qualities of the data.

4.2.2 Dealing with Data Scarcity

In certain circumstances, we may not have many training data sequences available. When data is scarce, unsupervised learning still applies, but may be compromised in terms of accuracy. Although having more real data is always a better solution, we choose to use splines with noise to introduce a few new data sequences to bolster the original data set when the data is scarce. Essentially the splines are a crude dynamics model of the human's motion that captures smoothness. The process of creating splines is as follows: First, some number of data points are sampled evenly from an available training data sequence. Next, we add some noise to the sampled data points and create a spline with it. Finally, we resample from the splines to interpolate the new data sequence. For a more detailed discussion of splines, see Appendix A. This process, illustrated in Figure 4-2, can be repeated to create more new data sequences.

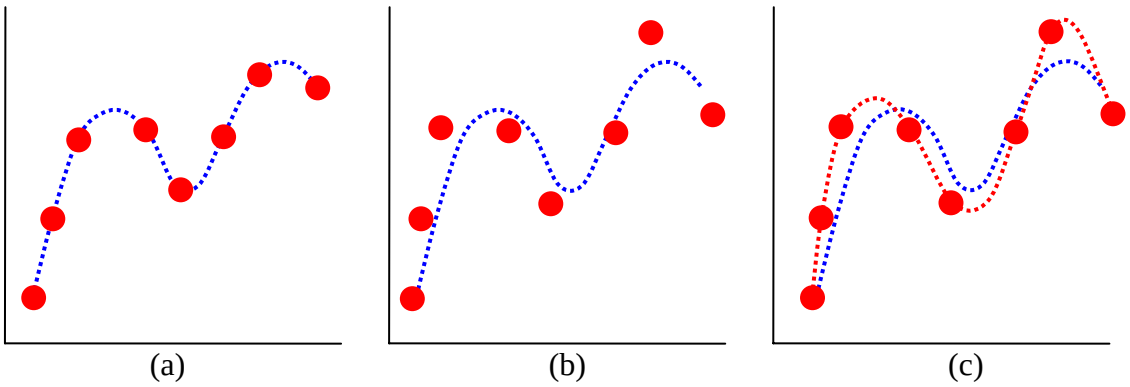


Figure 4-2: We use splines to generate new data when data is scarce. The process is as follows: (a) Sample data evenly. (b) Add noise to sampled data. (c) Create spline and interpolate new data sequence.

4.2.3 Dealing with High Dimensionality

The high dimensionality m of the state vectors makes the training data hard to work with. For example, classifiers do not scale well to high dimensions. Furthermore, more computational power is required to run algorithms on such multidimensional data, and they are difficult to represent graphically for user analysis. Therefore,

we use principal component analysis (PCA) to project to a lower dimensional space that captures the most important features. Using the PCA method presented in Section 2.1 to compute the principle components, we take a training data sequence

$$X = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix},$$

center the mean at zero by replacing X with $X - \boldsymbol{\mu}_X$, compute the covariance matrix Σ_X, obtain the eigenvalues $\{\lambda_1, \ldots, \lambda_m\}$ and eigenvectors $\{\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_m\}$ of the covariance matrix, form a feature vector based on the sorted eigenvalues $V = [\hat{\mathbf{v}}_{\max\lambda}, \ldots, \hat{\mathbf{v}}_{\min\lambda}]$, extract the $p : p \leq m$ most important dimensions of the feature vector $V_p = [\hat{\mathbf{v}}_{\max\lambda}, \ldots, \hat{\mathbf{v}}_p]$, and transform the data onto the feature vector to generate the principal components $X_p = XV_p$ for training sequence X. Given C training sequences and $c \in \{1, \ldots, C\}$, each training data sequence, after going through PCA, is represented by

$$X_c = \begin{bmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & & \vdots \\ x_{n_c1} & \cdots & x_{n_cp} \end{bmatrix}.$$

The set $\{X_1, \ldots, X_c, \ldots, X_C\}$ describes the principal components (PC) of all the training data. This set of data will be used throughout the plan learning process.

4.3 Unsupervised Activity Learning

To learn a plan from the PC data, we employ unsupervised machine learning on the data. Specifically, we use the Expectation Maximization (EM) algorithm discussed in Section 2.2 to cluster the training data into a mixture of Gaussians, similar to Barbič's method [7]. The number of Gaussians, or recognition resolution, is supplied by the user. We label this value k. Recall from 2.2 that the parameters of the j^{th}

Gaussian model include w_j, which is a normalized weight on the cluster, μ_j, which is the mean of the Gaussian, and Σ_j, which is the covariance.

Given the set of training data $\{X_1, \ldots, X_c, \ldots, X_C\}$, we perform learning over all the data sets together. To represent all training data together, we define an all-encompassing state vector sequence $\mathbf{X}$ that vertically concatenates all of the data sequences into one long sequence given by

$$\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_C \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_N \end{bmatrix},$$

where $N = \sum_{c=1}^{C} n_c$ is the total number of state vectors in all the training data sequences.

Given the concatenated training data described as $\mathbf{X}$ and number of clusters k, we initialize the model parameters for all $j \in \{1, \ldots, k\}$ with

$$\begin{aligned} w_j &= \frac{1}{k} \\ \mu_j &= \mathbf{x}_q, \ q = \left\lfloor \frac{N}{k+1} \cdot j \right\rfloor \\ \Sigma_j &= \Sigma\left(\mathbf{X}\right). \end{aligned}$$

In the initialization, the cluster weight parameters w_j are set to a uniform likelihood for all clusters. The means μ_j are initialized to evenly sampled state vectors $\mathbf{x}_q$ in $\mathbf{X}$. Initializing the means to state vectors already in the data ensures that later when we calculate probabilities of the state vectors from the probability density function governed by μ_j and Σ_j, there is at least one data point that does not have a near zero probability. This is important because our data has p dimensions, and Gaussian probability density functions evaluate to small values for large dimensions. Often, if a data point does not lie near the center, or mean state vector of a cluster, the probability density function at that data point will evaluate to a near-zero probability that

is beyond machine precision. If we initiate the cluster means μ_j completely randomly, it is often the case that all probability density functions for all clusters evaluate to machine zero for all the data points, in which case unsupervised learning ceases to work. Thus the means are initialized in such a way to prevent this problem. Finally, the covariances for each cluster are all conservatively initialized to the covariance of all training data.

We define $Y = [y_1, \ldots, y_N]^T$ as a vector containing the class labels corresponding to each data point in $\mathbf{X}$. Each label y_i is a variable with domain $\{1, \ldots, k\}$. Now we can perform the EM learning algorithm as explained in Section 2.2:

1. **E-step: Label Data.** Given all the training data $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_N]^T$ and the number of clusters k, compute the probability that datum $\mathbf{x}_i$ belongs to class j using the update equation

$$\hat{p}\left(j \mid i\right) \leftarrow P\left(y_i = j \mid \mathbf{x}_i, \theta_j\right), \quad j = 1 \ldots k, \; i = 1 \ldots N. \tag{4.1}$$

   where

$$P\left(y_i = j \mid \mathbf{x}_i, \theta_j\right) = \alpha \cdot P\left(\mathbf{x}_i \mid \mu_j, \Sigma_j\right) \cdot \hat{w}_j \tag{4.2}$$

$$= \alpha \cdot \hat{w}_j \cdot \frac{1}{(2\pi)^{p/2} \left|\Sigma_j\right|^{1/2}} \exp\left(-\frac{1}{2}\left(\mathbf{x} - \mu_j\right)^T \Sigma_j^{-1} \left(\mathbf{x} - \mu_j\right)\right) \tag{4.3}$$

   by Bayes rule. The α is a normalization factor.

2. **M-step: Update Parameters.** Compute the class distribution parameters

$\hat{w}_j$, $\hat{\mu}_j$, and $\hat{\Sigma}_j$ using the update equations

$$\hat{n}_j \leftarrow \sum_{i=1}^{N} \hat{p}\left(j \mid i\right) \tag{4.4}$$

$$\hat{w}_j \leftarrow \frac{\hat{n}_j}{N} \tag{4.5}$$

$$\hat{\mu}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i=1}^{N} \hat{p}\left(j \mid i\right) \mathbf{x}_i \tag{4.6}$$

$$\hat{\Sigma}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i=1}^{N} \hat{p}\left(j \mid i\right) \left(\mathbf{x}_i - \hat{\mu}_j\right) \left(\mathbf{x}_i - \hat{\mu}_j\right)^T \tag{4.7}$$

We define a small value δ such that the EM algorithm iterates until

$$\left\| \boldsymbol{\mu}^{(t)} - \boldsymbol{\mu}^{(t-1)} \right\| \leq \delta,$$

where $\boldsymbol{\mu} = \left[\mu_1, \ldots, \mu_k\right]^T$ is the matrix containing all cluster means, and the parenthesized superscript (t) indicates the iteration step. This allows the EM iteration to stop when the model parameters have converged. It is optional to also define an absolute maximum number of iteration steps to stop the iteration even if convergence is not reached. This is only useful if run time or computation power is limited and we are willing to compromise some accuracy.

As an example, we ran the EM algorithm on the arbitrary data sequences presented in 4.2.2. Allowing the recognition resolution k to be 2, EM outputs the two classes, or activities, shown in Figure 4-3. The ellipses represent the covariances of the two clusters, and the cluster means are located at the geometric center of the ellipses. Each activity is labeled with an activity number from $1, \ldots, k$.

This simple example is shown here only to give an idea of how the EM algorithm works. More extensive results using real motion capture data is presented in Chapter 7.

Now we have learned the distinct activities. In the next section, we learn how these activities combine into simple activity sequences.

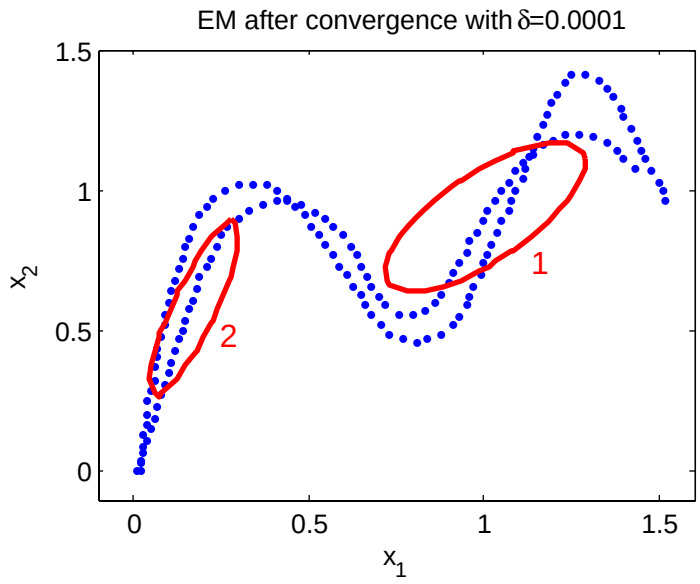


Figure 4-3: Example of running EM learning on some 2-D data sequences with two clusters

4.4 Extracting Activity Sequences

This section abstracts the data sequences into activity sequences and learns expected activity durations for each unique sequence to prepare for representation as a TPN. To do this, we first find the probability that a data point belongs to a particular activity. Next, we find the most probable activity at each data point for each training trajectory. Finally, we compress this information as a sequence of activities with durations. The key is that we now need to consider each training data sequence independently, i.e. we consider the trajectories in the set $\{X_1, \ldots, X_c, \ldots, X_C\}$ separately instead of all together as $\mathbf{X}$ before. Previously, we were learning what the activities are based on all training data; now we are determining what sequences of activities exist in the training data.

For each trajectory X_c, we use the learned classifier to label each principal component data point with its most likely activity. To accomplish this, we first find the probability of activity $y_i = j$ given a single data point $\mathbf{x}_i$, where $i = 1, \ldots, n_c$, by

using Bayes rule:

$$P(y_i = j \mid \mathbf{x}_i) = \frac{P(\mathbf{x}_i \mid y_i = j)\, P(y_i = j)}{\sum_{j=1}^{k} P(\mathbf{x}_i \mid y_i = j)\, P(y_i = j)} = \frac{P(\mathbf{x}_i \mid \mu_j, \Sigma_j)\, w_j}{\sum_{j=1}^{k} P(\mathbf{x}_i \mid \mu_j, \Sigma_j)\, w_j}.$$

Next, to find the most probable activity at a particular data point, we take the argmax over all the activities j:

$$y_i^* = \underset{j}{\operatorname{argmax}} \left(P(y_i = j \mid \mathbf{x}_i)\right) = \underset{j}{\operatorname{argmax}} \left(\frac{P(\mathbf{x}_i \mid \mu_j, \Sigma_j)\, w_j}{\sum_{j=1}^{k} P(\mathbf{x}_i \mid \mu_j, \Sigma_j)\, w_j} \right).$$

Now we find the most probable activity at every data point in a training trajectory sequence, that is, we map

$$X_c = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{n_c} \end{bmatrix} \implies Q_c = \begin{bmatrix} y_1^* \\ y_2^* \\ \vdots \\ y_{n_c}^* \end{bmatrix}.$$

Q_c is a sequence of labeled activities, which we call an *activity labeling*, where

$$Q_c = \underset{j}{\operatorname{argmax}} \left[\frac{P(X_c \mid \theta_j)}{P(X_c \mid \Theta)} \right].$$

We observe that an activity label will typically be repeated for several successive data points. Hence, we encode this more compactly by collapsing the repeated sequences using a *run-length encoding* [56]. We introduce H_c as a run-length encoding of Q_c, so that $H_c = [\mathbf{a}_c, \mathbf{d}_c]$, where $\mathbf{a}_c$ is a sequence of activities and $\mathbf{d}_c$ is its corresponding durations, such that successive activities are distinct. For example, if a

particular trajectory gives $Q_c = [3\ 3\ 3\ 3\ 3\ 5\ 5\ 2\ 2\ 2]^T$, then

$$H_c = \begin{bmatrix} 3 & 5 \\ 5 & 2 \\ 2 & 3 \end{bmatrix}$$

because there are five 3's followed by two 5's followed by three 2's. Since Q_c is an activity trajectory, $\mathbf{a}_c$ is the *activity sequence* in a trajectory, and $\mathbf{d}_c$ is the corresponding *activity duration sequence*.

Let's apply activity sequence extraction to the data used in our previous example of Figure 4-3. There are two data trajectories and two labeled activities. Figure 4-4 shows the resulting activity trajectories (Q_1 and Q_2). In this example, the run-length

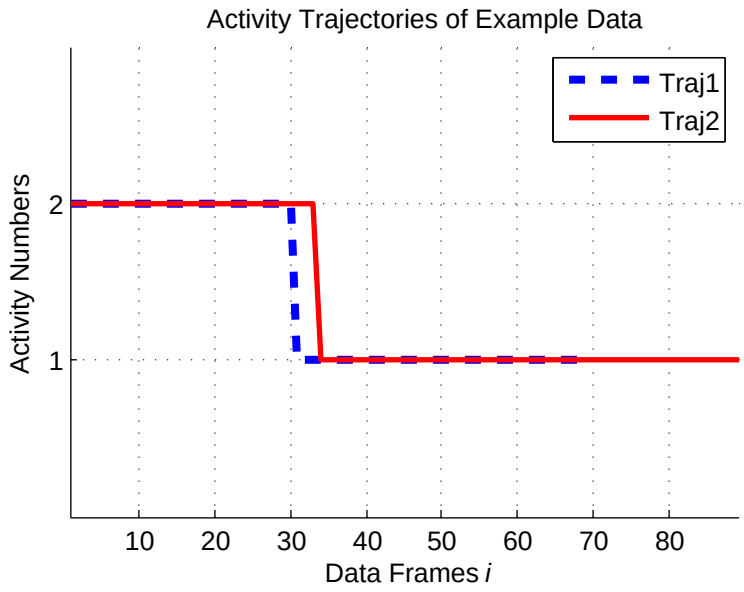


Figure 4-4: Activity trajectories corresponding to the data shown in Figure 4-3

encodings of the activity trajectories are

$$H_1 = \begin{bmatrix} 2 & 30 \\ 1 & 38 \end{bmatrix} \text{ and } H_2 = \begin{bmatrix} 2 & 33 \\ 1 & 55 \end{bmatrix}.$$

We notice that the activity sequences of the two trajectories are the same, just with different ranges of durations. Eventually, we want to summarize this information into

a plan so that all trajectories with the same activity sequences can be represented together as a *plan trajectory* with associated durations. The process of creating a plan is discussed in the next Section.

4.5 Creating a Probabilistic Temporal Plan Network (TPN)

Creating a temporal plan network involves learning a temporal activity sequence by first learning the probability distribution of durations for each activity, and then learning the prior probabilities of each plan.

Assume we have the activity sequences $\{\mathbf{a}_1, \ldots, \mathbf{a}_C\}$ and corresponding activity duration sequences $\{\mathbf{d}_1, \ldots, \mathbf{d}_C\}$ for all the PC data sets $\{1, \ldots, C\}$. We first combine duplicated activity sequences and summarize the duration information for them. Let $A_c = |\mathbf{a}_c|$ denote the number of activities in activity sequence $\mathbf{a}_c$. Let $s \in \{1, \ldots, S\}$ denote each distinct activity sequence generated from all the training data C training trajectories, and let . Thus we can find a set of *unique* activity sequences $\{\mathbf{A}_1, \ldots, \mathbf{A}_S\} \subseteq \{\mathbf{a}_1, \ldots, \mathbf{a}_C\}$. For each unique activity sequence $\mathbf{A}_s$, we define $D_{\mathcal{C}_s} = \{\mathbf{d}_c\}_{c \in \mathcal{C}_s}$ as the set of corresponding duration sequences, where $\mathcal{C}_s = \{c \mid \mathbf{a}_c = \mathbf{A}_s\}$ denotes the training sequences in C correspond to the unique sequence s.

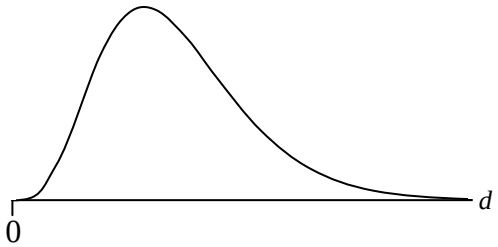


Figure 4-5: A gamma distribution is unimodal and non-negative.

Next, we learn the duration of each activity in each sequence. We model the duration data $D_{\mathcal{C}_s}$ as a two-parameter gamma distribution [15] $\Gamma(k, \theta)$ as shown in Figure 4-5. We choose a gamma distribution instead of a Gaussian because the

durations sampled from a gamma distribution is guaranteed to be non-negative. The gamma parameters k and θ are related to the mean μ and variance σ^2 by

$$\begin{aligned}\mu =& k\theta \\ \sigma^2 =& k\theta^2.\end{aligned}$$

We find the mean and variance of the set $D_{\mathcal{C}_s}$:

$$\begin{aligned}\boldsymbol{\mu}\left(D_{\mathcal{C}_s}\right) &= \frac{\sum_{\mathcal{C}_s} \mathbf{d}_c}{\left|D_{\mathcal{C}_s}\right|} \\ \boldsymbol{\sigma}^2\left(D_{\mathcal{C}_s}\right) &= \frac{\sum_{\mathcal{C}_s}\left(\mathbf{d}_c - \boldsymbol{\mu}\left(D_{\mathcal{C}_s}\right)\right)^2}{\left|D_{\mathcal{C}_s}\right| - 1},\end{aligned}$$

where the element-wise product of two vectors $\mathbf{ab}$ or $\mathbf{a}^2$ is a vector containing the products of the corresponding elements of the vectors. To find the parameters $\mathbf{k}\left(D_{\mathcal{C}_s}\right)$ and $\boldsymbol{\theta}\left(D_{\mathcal{C}_s}\right)$, we use

$$\begin{aligned}\mathbf{k}\left(D_{\mathcal{C}_s}\right) =& \frac{\left(\boldsymbol{\mu}\left(D_{\mathcal{C}_s}\right)\right)^2}{\boldsymbol{\sigma}^2\left(D_{\mathcal{C}_s}\right)} \\ \boldsymbol{\theta}\left(D_{\mathcal{C}_s}\right) =& \frac{\boldsymbol{\sigma}^2\left(D_{\mathcal{C}_s}\right)}{\boldsymbol{\mu}\left(D_{\mathcal{C}_s}\right)}.\end{aligned}$$

The special case of $\Gamma\left(0,0\right)$ is equivalent to a $[0,0]$ time bound, and $\Gamma\left(0,\infty\right)$ is equivalent to a $[0,\infty]$ time bound.

Finally, we define a duration summary matrix

$$\mathbf{D}_s = \left[\mathbf{k}\left(D_{\mathcal{C}_s}\right), \boldsymbol{\theta}\left(D_{\mathcal{C}_s}\right)\right]$$

to represent the abstracted duration information using the gamma distributions, enabling us to compactly represent the activity durations. The set of unique activity sequences $\{\mathbf{A}_1, \ldots, \mathbf{A}_S\}$ and corresponding durations $\{\mathbf{D}_1, \ldots, \mathbf{D}_S\}$ comprise two major components of a TPN. The third component is the set of prior probabilities of each plan, which we will discuss after presenting an example of creating the first two components.

To illustrate the process of creating the activities and durations in a TPN, suppose we have the following small number of $C = 6$ training trajectories:

$$H_1 = \begin{bmatrix} 3 & 15 \\ 5 & 32 \\ 2 & 14 \end{bmatrix}, \; H_2 = \begin{bmatrix} 3 & 18 \\ 5 & 32 \\ 2 & 21 \end{bmatrix}, \; H_3 = \begin{bmatrix} 3 & 9 \\ 1 & 22 \\ 4 & 6 \\ 1 & 42 \end{bmatrix},$$

$$H_4 = \begin{bmatrix} 3 & 19 \\ 5 & 28 \\ 2 & 12 \end{bmatrix}, \; H_5 = \begin{bmatrix} 3 & 7 \\ 1 & 29 \\ 4 & 10 \\ 1 & 43 \end{bmatrix}, \; H_6 = \begin{bmatrix} 3 & 11 \\ 5 & 24 \\ 2 & 17 \end{bmatrix}.$$

We identify that there are $S = 2$ unique activity sequences

$$\mathbf{A}_1 = \begin{bmatrix} 3 \\ 5 \\ 2 \end{bmatrix}, \; \mathbf{A}_2 = \begin{bmatrix} 3 \\ 1 \\ 4 \\ 1 \end{bmatrix},$$

with corresponding

$$D_{\mathcal{C}_1} = \left\{ \begin{bmatrix} 15 \\ 32 \\ 14 \end{bmatrix}, \begin{bmatrix} 18 \\ 32 \\ 21 \end{bmatrix}, \begin{bmatrix} 19 \\ 28 \\ 12 \end{bmatrix}, \begin{bmatrix} 11 \\ 24 \\ 17 \end{bmatrix} \right\}, \; D_{\mathcal{C}_2} = \left\{ \begin{bmatrix} 9 \\ 22 \\ 6 \\ 42 \end{bmatrix}, \begin{bmatrix} 7 \\ 29 \\ 10 \\ 43 \end{bmatrix} \right\},$$

where the mean and variance vectors are

$$\boldsymbol{\mu}\left(D_{\mathcal{C}_1}\right) = \begin{bmatrix} 15.75 \\ 29.00 \\ 16.00 \end{bmatrix}, \quad \boldsymbol{\sigma}^2\left(D_{\mathcal{C}_1}\right) = \begin{bmatrix} 12.89 \\ 14.67 \\ 15.37 \end{bmatrix},$$

$$\boldsymbol{\mu}\left(D_{\mathcal{C}_2}\right) = \begin{bmatrix} 8.00 \\ 25.50 \\ 8.00 \\ 42.50 \end{bmatrix}, \quad \boldsymbol{\sigma}^2\left(D_{\mathcal{C}_2}\right) = \begin{bmatrix} 1.99 \\ 24.50 \\ 8.01 \\ 0.50 \end{bmatrix}.$$

Hence, the activity duration sequence summaries $\mathbf{D}_s = [\mathbf{k}, \boldsymbol{\theta}]$ are

$$\mathbf{D}_1 = \begin{bmatrix} 19.25 & 0.82 \\ 57.33 & 0.51 \\ 16.66 & 0.96 \end{bmatrix}, \; \mathbf{D}_2 = \begin{bmatrix} 32.19 & 0.25 \\ 26.54 & 0.96 \\ 7.99 & 1.00 \\ 3583.12 & 0.01 \end{bmatrix}.$$

The final component of plan learning is to compute the prior probability at which each sequence occurs in training. These probabilities are used as prior for the recognizer. We denote the set of trajectory probabilities as $\{p_1, \ldots, p_S\}$. Each trajectory probability is defined as the ratio of number of training sequences that correspond to a particular trajectory to the total number of training sequences:

$$p_s = \frac{|\mathcal{C}_s|}{C}. \tag{4.8}$$

Given the learned activity sequences with duration distributions for each trajectory in the plan and prior probabilities for each trajectory, we have all the components of a temporal plan network. Each trajectory in the TPN is a possible plan to recognize. Thus we can describe the plan network as S possible trajectories spawning from one choice event. The plan network for the example given above is shown in Figure 4-6. Durations are labeled as gamma distributions of the form $\Gamma\left(k, \theta\right)$, where

$\Gamma(0,0)$ is equivalent to a $[0,0]$ time bound, and $\Gamma(0,\infty)$ is equivalent to a $[0,\infty]$ time bound.

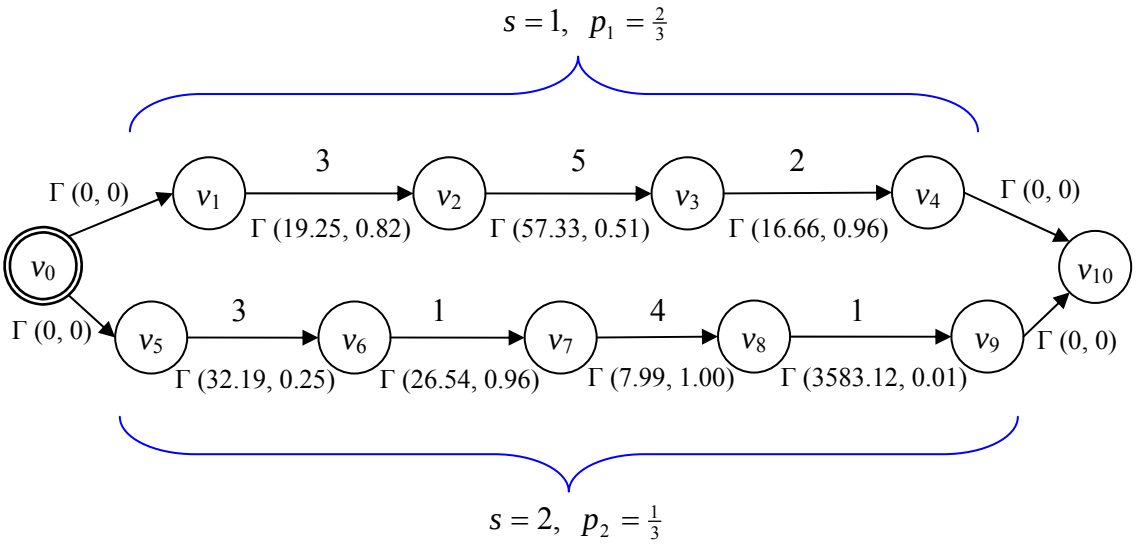


Figure 4-6: The temporal plan network derived from example activity and duration sequences

Finally, the temporal plan network is recorded in XML format. An example snippet of XML code used to describe a small part of the TPN in Figure 4-6 is shown in Appendix B.

Chapter 5

Probabilistic Plan Recognition

The problem of intent recognition is to infer the sequence of activities a human has been performing and, most importantly, to predict which activities will occur in the future. Thus the recognizer needs to do more than estimating activities matches the current data, as was done in [7]; it must determine the most likely *sequence* of activities that explain the observed data. This is the task of the plan recognizer. The plan recognizer does not require the observed data to be a full sequence, i.e. upon observing the first few data points, the plan recognizer is able identify a corresponding most likely activity sequence in the plan. This requirement enables the recognizer to not only determine which activities have been observed, but also predict activities likely to occur beyond current observations.

5.1 Overview

This chapter presents our method of plan recognition. An overview of the plan recognition process is illustrated in Figure 5-1. First, the TPN learned in Chapter 4 is represented as a Hidden Markov Model (HMM) by assigning transition probabilities that capture the distribution of each activity's duration. Next, the Viterbi algorithm is applied to the HMM to obtain the most likely sequence of activities. Finally, we refer back to the TPN to determine the predicted activity sequence.

We first discuss how the test data is processed to be comparable to the training

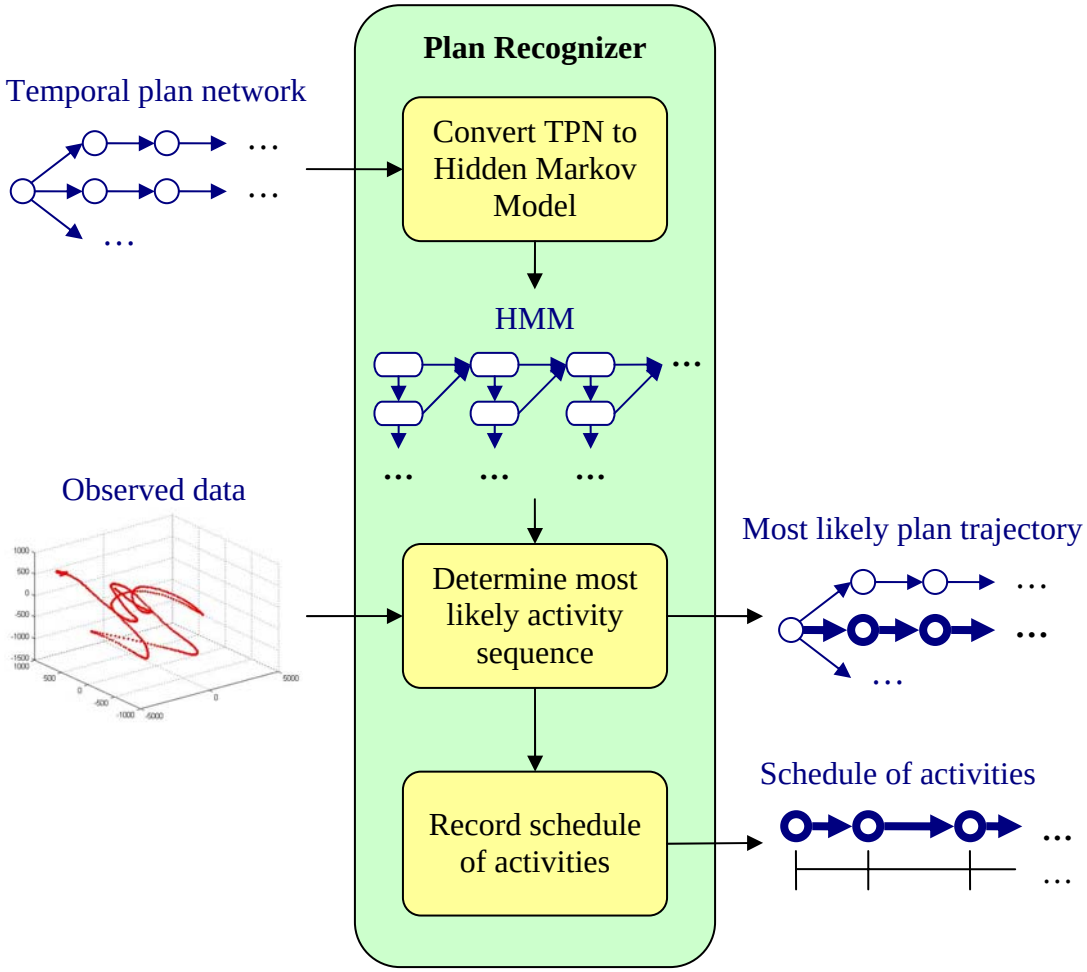


Figure 5-1: Overview of the plan recognition process

data in Section 5.2, then we set up the nomenclature of the problem in Section 5.3. We discuss the approach of representing the temporal plan network learned in Chapter 4 as an HMM in Section 5.4, and proceed to finding the most likely sequence of activities using the Viterbi algorithm in Section 5.5. We trace the steps to determining the predicted activity sequence in Section 5.6. Finally, we demonstrate the plan recognition process on a simple example in Section 5.7.

5.2 Formatting the Observed Testing Data to Be Used in Recognition

The observed data is presumed to be of the same format as the training data: in the application used in this thesis, they are motion capture data of m dimensions. The observed data is of the form

$$X_{obs} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{n_{obs}} \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \\ \vdots & & \vdots \\ x_{n_{obs}1} & \cdots & x_{n_{obs}m} \end{bmatrix},$$

where the length of the observed data sequence n_{obs} can be much shorter than those of the training data, since the idea is to recognize the activity sequence based on only the first few motions.

We reduce the dimensions of the test data using the same feature vectors identified through PCA during learning. The test data only is re-centered according to the mean of the training data and then transformed using the existing feature vector. More specifically, following Steps 2 and 6 in Section 2.1, we center the test data so that X_{obs} is replaced by $[\mathbf{x}_1 - \mu_X,\ \mathbf{x}_2 - \mu_X,\ \ldots,\ \mathbf{x}_{n_{obs}} - \mu_X]^T$, where μ_X is the mean of the training data, as obtained before. Then using the feature vector V_p derived in Section 4.2.3, we obtain the principal components of the test data $X_{p_{obs}} = X_{obs}V_p$, which has dimensions $n_{obs} \times p$.

5.3 Preliminaries: Notation and Setup for Plan Recognition

Before delving into the details of plan recognition, we first introduce some notation of the inputs. Specifically, we describe our representation of the activity labels of an observed sequence, the indexes for referencing activities in a plan network, and the

details of an activity duration distribution.

We first introduce the notation used to describe the observation data labels. The observed data $X_{p_{obs}}$ is sampled at a constant rate γ, so the time steps $t \in \{1, \ldots, n_{obs}\}$ reflect the relative temporal measurement of the observed data. Therefore, we can measure the duration between two different data points $t = a$ and $t = b$ by calculating $(b - a) \times \gamma$. Since γ is a constant, we ignore it when discussing the recognition algorithm. The vector $Y_{obs} = [y_1, \ldots, y_{n_{obs}}]^T$ denotes the activity labels corresponding to each data point in $X_{p_{obs}}$. There are k different activities. Each label y_t is a single valued variable with domain $\{1, \ldots, k\}$ that describes the appropriate activity for data point $\mathbf{x}_t$. Part of the plan recognition task is to determine the values of the labels in Y_{obs} given the observed data $X_{p_{obs}}$.

Next, we present the notation for referencing activities in a TPN. We label the trajectories and activities in a plan network with indexes s and r, as shown in Figure 5-2, where $s \in \{1, \ldots, S\}$ are indexes of trajectories, and $r \in \{1, \ldots, R_s\}$ are indexes of activities within a trajectory. We refer to a particular activity in the plan as $a_{s,r}$, where $r \leq R_s$, and $a_{s,r}$ can have values $\{1, \ldots, k\}$.

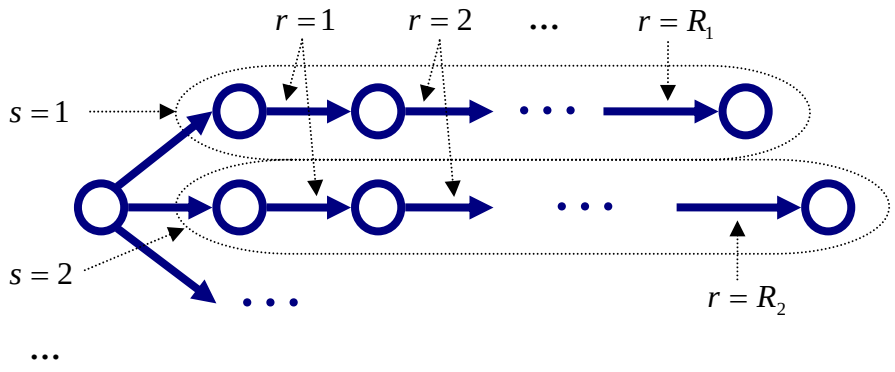


Figure 5-2: Labels for trajectories and activities in a plan. Trajectories are labeled s, while activities are labeled r. For each trajectory s, there are R_s activities.

Finally, we discuss the details of representing a duration distribution to support estimation and prediction. We define variable $d_{s,r}$ as the best current estimate (at time step t) of the duration of the activity $a_{s,r}$ corresponding to y_t. We define t_m as the time step when the current activity first began, that is, $t_m = \sum_{k=1}^{r-1} d_{s,k}$. As a

shorthand, we use $\tau = t - t_m + 1$ to denote the time elapsed since the beginning of the current activity. If at time step t, the activity does not transition to a new activity, that is, time step t is sometime in the middle of the activity, then we know that $d_{s,r}$ should be at least τ, since $d_{s,r}$ anticipates the entire duration of the current activity.

The temporal plan network learned in Chapter 4 allows us to predict the distributions over the durations of each activity $a_{s,r}$. An example duration distribution is shown in Figure 5-3. The shaded region is the probability that the current ac-

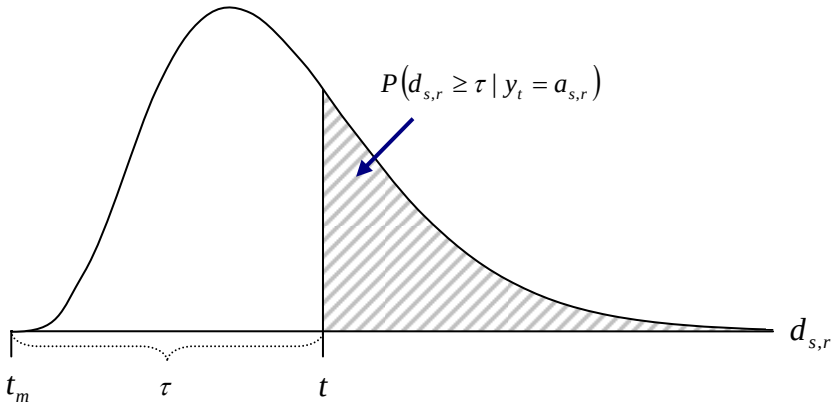


Figure 5-3: Duration distribution

tivity's duration $d_{s,r}$ is longer than the time elapsed since the activity began, or $P(d_{s,r} \geq \tau \mid y_t = a_{s,r})$. In the example shown in the figure, a large amount of time has elapsed since the beginning of the activity, so there is a low probability that the activity duration is longer than the time elapsed so far. If elapsed time is short, we would expect to take longer to reach the activity's duration and transition to a new activity, so the probability that the activity's duration exceeds the time elapsed is large.

We have presented the notation for an observed data sequence and a TPN, which are inputs to the plan recognizer. The next section discusses the first step of the plan recognition process.

5.4 Represent a TPN as a Non-stationary Hidden Markov Model (HMM)

A temporal plan network is shown in Figure 5-4. We want to represent the TPN as a Hidden Markov Model (HMM) in order to apply appropriate estimation techniques.

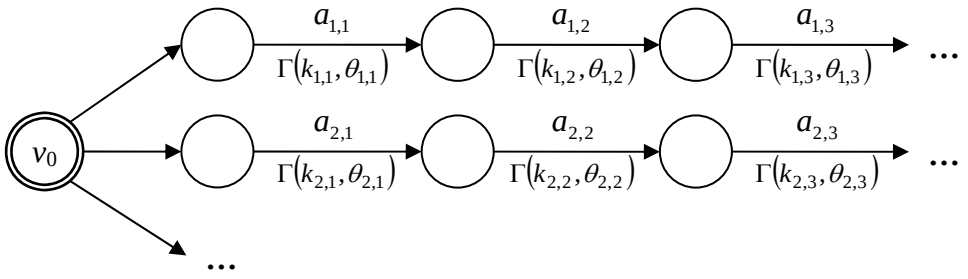


Figure 5-4: Temporal Plan Network

Our activities in the TPN are time-dependent and have specific duration distributions. A standard HMM, however, cannot model arbitrary state duration distributions. Instead, we use a non-stationary representation of HMM based on Sin and Kim's work [51] and Rabiner's discussion on inclusion of explicit state duration densities in HMMs [47]. We expand the representation of a standard HMM such that the transitions of a state to itself are described explicitly, as shown in Figure 5-5. By considering activities at different time steps as different states, we can treat the non-stationary HMM as a standard HMM, and hence can apply standard filtering techniques.

Recall from 5.3 that we defined the time that an activity starts as $t_m = \sum_{k=1}^{r-1} d_{s,k}$, and the time elapsed from the beginning of the activity to the current time is $\tau = t - t_m + 1$. We represent an activity at time step t as $a_{s,r}^{(\tau)}$. These activities at particular time steps are the states of the HMM. The observed test data are the observations, and the prior probabilities are those derived in Chapter 4.

The sequence of events in a non-stationary HMM is described as follows: At time t_m, we enter into state $a_{s,r}^{(1)}$. For the next τ time units, we make transitions from that state to itself with probabilities derived from the activity's duration distribution, as

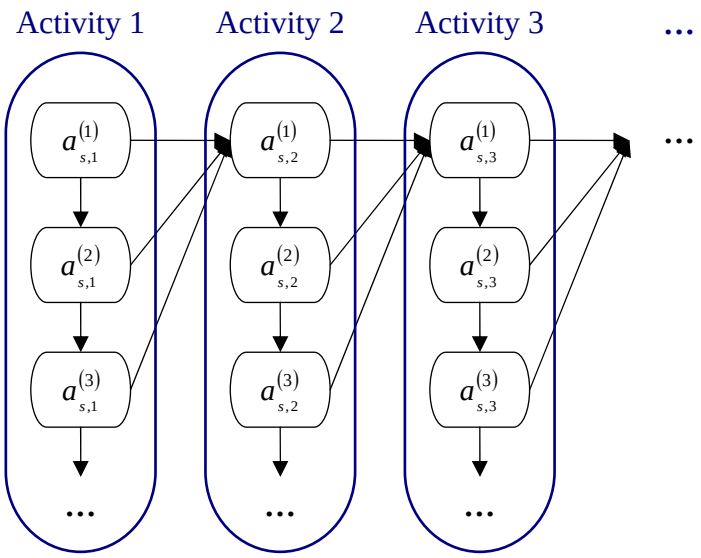


Figure 5-5: Markov model of a particular trajectory in the TPN. Each activity time slice $a^{(\tau)}_{s,r}$ represents the r^{th} activity in trajectory s at time step τ since the beginning of the activity. Transition probabilities are determined by the duration distribution of each activity.

will be shown in Section 5.4.1. The observations $X_{obs} = [\mathbf{x}_1, \ldots, \mathbf{x}_{n_{obs}}]^T$ are assumed to be independent. After all the self-transitions, we transition to the next state $a^{(1)}_{s,r+1}$ with the transition probability derived in Section 5.4.2.

Next, we discuss the method of obtaining the transition and observation probabilities.

5.4.1 Staying in an Activity

At each t, an activity $a^{(\tau)}_{s,r}$ can transition either to itself $a^{(\tau+1)}_{s,r}$, corresponding to the downward arrows in Figure 5-5, or to the next activity $a^{(1)}_{s,r+1}$, corresponding to the rightward moving arrows in Figure 5-5. Associated with each activity $a_{s,r}$, is a duration distribution similar to the one shown in Figure 5-3.

The probability of transitioning from an activity to itself at a particular time step is dependent on the distribution of that activity's duration. For example, at time steps soon after the beginning of an activity, i.e. when τ is small, the probability of staying in the same activity should be larger than for time lapses well beyond the mean duration of the current activity. The time dependency prevents us from using

a simple self-cycling automaton with constant probability to represent an activity transitioning to itself. The probability that an activity at time t transitions to itself at time $t+1$ should be the area of the region under the gamma duration distribution curve [15] above $t+1$, similar to the shaded region in Figure 5-3. It is governed by

$$\begin{aligned} P\left(a_{s,r}^{(\tau+1)} \mid a_{s,r}^{(\tau)}\right) =& P\left(d_{s,r} \geq \tau+1 \mid y_t = a_{s,r}\right) && (5.1)\\ =& \int_{\tau+1}^{\infty} \Gamma\left(k_{s,r}, \theta_{s,r}\right)\, du && (5.2)\\ =& \frac{1}{\theta_{s,r}^{k_{s,r}} \Gamma\left(k_{s,r}\right)} \int_{\tau+1}^{\infty} u^{k_{s,r}-1} e^{-\frac{u}{\theta_{s,r}}}\, du, && (5.3) \end{aligned}$$

where the gamma function [16] in Equation 5.3 is given by $\Gamma(z) = \int_0^{\infty} \zeta^{z-1} e^{-\zeta}\, d\zeta$, which for positive integers z, reduces to $\Gamma(z) = (z-1)!$. We define a small value ϵ such that any transition probability smaller than ϵ is regarded as zero. This ensures that the HMM will be bounded and finite.

5.4.2 Moving to the Next Activity

The transition probability of an activity $a_{s,r}$ at time t to a new activity $a_{s,r+1}$ at time $t+1$ is just the complement of the probability of staying in the same activity, or

$$\begin{aligned} P\left(a_{s,r+1}^{(1)} \mid a_{s,r}^{(\tau)}\right) =& 1 - P\left(a_{s,r}^{(\tau+1)} \mid a_{s,r}^{(\tau)}\right) && (5.4)\\ =& 1 - \frac{1}{\theta_{s,r}^{k_{s,r}} \Gamma\left(k_{s,r}\right)} \int_{\tau+1}^{\infty} u^{k_{s,r}-1} e^{-\frac{u}{\theta_{s,r}}}\, du. && (5.5) \end{aligned}$$

5.4.3 Observation Model

The observation probabilities of the Hidden Markov Model are given by the multi-dimensional activity distributions in the state space. It is the probability that we observe data point $\mathbf{x}_t$, given that we are in activity state $a_{s,r}$. This is simply the evaluation of the multivariate Gaussian at the observed data point:

$$P\left(\mathbf{x}_t \mid y_t = a_{s,r}\right) = P\left(\mathbf{x}_t \mid \mu_{a_{s,r}}, \Sigma_{a_{s,r}}\right). \quad (5.6)$$

5.4.4 Initial Probabilities

The initial probabilities of all the states in the Hidden Markov Model are such that for each trajectory s,

$$P\left(y_1 = a_{s,1}^{(1)}\right) = p_s \tag{5.7}$$

$$P\left(y_1 = a_{s,r}^{(\tau)} \mid r > 1, \tau > 1\right) = 0, \tag{5.8}$$

where p_s is obtained from Equation 4.8. Prior probabilities only exist for the first activity at the first time step in each trajectory because they are the initial start states. Hence, the priors on all other activities at all other time steps are zeros as stated in Equation 5.8.

5.5 HMM Model Evaluation to Recognize Most Likely Path

Part of the intent recognition problem is to determine the sequence of activities that a human has performed from an observed sequence. Given a hidden Markov model that models the transitions of activity states, we are interested in finding the most likely path of hidden activity states that generated the observed sequence. We accomplish this by applying the Viterbi algorithm on the HMM.

First, we define some shorthand notation. We denote the transition probability from activity i to activity j as $p_{ij} = P\left(y_{t+1} = j \mid y_t = i\right)$, where i and j are distinct activity states $a_{s,r}^{(\tau)}$ in the HMM. We denote the probability of observing $\mathbf{x}_t$ during activity i as $o_i\left(\mathbf{x}_t\right) = P\left(\mathbf{x}_t \mid y_t = i\right)$. The initial state probability is notated as $\pi_i = P\left(y_1 = i\right)$.

Since we formulated the non-stationary HMM as a standard HMM in Section 5.4, we can apply standard filtering techniques for HMMs. Given a Hidden Markov Model λ, the probability that the activity state is some j at the current time step t, given all the observed data up to the current time step is called the forward probability $f_t\left(j\right)$.

After using Bayes' rule and the Markov property that observations at time t depend only on the state at t, the unnormalized forward probability $f_t(j)$ is evaluated as [49, 51, 12]:

$$f_t(y_t = j) = P(y_t = j \mid \mathbf{x}_{1:t}) \tag{5.9}$$

$$= P(\mathbf{x}_t \mid y_t = j) \sum_i P(y_t = j \mid y_{t-1} = i)\, f_{t-1}(y_{t-1} = i). \tag{5.10}$$

We write the forward probability in shorthand as

$$f_t(j) = o_j(\mathbf{x}_t) \sum_i p_{ij}\, f_{t-1}(i), \tag{5.11}$$

where the forward probability is initialized to $f_1(j) = \pi_j\, o_j(\mathbf{x}_1)$.

To recognize the hidden activity sequence that a human has performed, we find the most likely sequence of states through the hidden Markov model. We use a technique similar to filtering to compute the probability of the most likely path that reaches each state in the HMM, called the Viterbi algorithm [12]. To identify the most likely state sequence, we keep pointers from each state back to the most likely state that leads to it, and the sequence is identified by following the pointers back from the most likely final state. We denote $m_{1:t}(y_t = j)$ as the highest probability of any single path ending in state j, at time t, given the observations from $1, \ldots, t$. This probability is given by

$$m_{1:t}(y_t = j) = \max_{y_{1:t-1}} P(y_{1:t-1}, y_t = j \mid \mathbf{x}_{1:t}) \tag{5.12}$$

$$= \alpha\, P(\mathbf{x}_t \mid y_t)\ \max_i (P(y_t = j \mid y_{t-1} = i)\ m_{1:t-1}(y_{t-1} = j)), \tag{5.13}$$

which in shorthand is

$$m_{1:t}(j) = \alpha\, o_j(\mathbf{x}_t)\ \max_i (p_{ij}\, m_{1:t-1}(j)), \tag{5.14}$$

where the path probability is initialized to $m_{1:1}(j) = \pi_j\, o_j(\mathbf{x}_1)$.

Now that we know the probability of the most likely path, we need to extract that path to determine the most likely sequence. We start with the state j that gave the largest path probability at the last time step t. Having calculated the most likely path probability, we can determine which preceding state was the one to generate $m_{1:t}(j)$, that is, what state we were in at time $t-1$ if we arrive optimally at state j at time t. We create a back pointer $\phi_t(j)$ for each state j that points to the most likely preceding state i leading to the current state. The back pointer is defined as

$$\phi_t(j) = \underset{i}{\operatorname{argmax}} \left(p_{ij}\, m_{1:t-1}(j) \right). \tag{5.15}$$

To determine the most likely path, we trace the back pointer back to $t = 1$, so that $i_{t-1} = \phi_t(i_t)$. We call the resulting most likely activity sequence $\hat{\mathbf{A}} = [i_1, \ldots, i_t]^T$.

The Viterbi algorithm is a computationally efficient way of determining the most likely sequence of states in a Hidden Markov Model. It uses recursion to maximize computational efficiency by avoiding looking at every possible path in the model. The Viterbi algorithm has a time complexity linear in the number of time steps t, or, and a space complexity also linear in t. The Viterbi algorithm complexity with respect to the number of states N in the HMM depends on the implementation of the matrix product operation, since we implement transition probabilities and Viterbi messages in matrix form. The complexity of matrix products is $O(N^3)$ in the worst case assuming a naïve implementation so that the worst case Viterbi complexity is $o(t \cdot N^3)$. Fortunately, the transition probability matrix is extremely sparse, for which matrix product operations scale linearly with the number of non-zero elements. Our encoding of the transition probability matrix has on the order of N non-zero elements, so our Viterbi complexity is $O(t \cdot N)$.

5.6 Recognized and Predicted Activity Sequences

The final step in plan recognition is to associate the recognized activity sequence with corresponding durations, and generate a prediction on future activities. We ac-

complish this by performing run-length encoding on the most likely activity sequence obtained from the HMM, and referring back to the TPN for possible future activities.

Recall that the labels i in the most likely activity sequence $\hat{\mathbf{A}}$ denote activities at particular time steps $a_{s,r}^{(\tau)}$. Similar to the method used in Section 4.4, we define $\hat{\mathbf{H}}$ as a run-length encoding of $\hat{\mathbf{A}}$, where $\hat{\mathbf{a}} = [\hat{a}_1, \hat{a}_2, \ldots]$ are the recognized activities and $\hat{\mathbf{d}} = \left[\hat{d}_1, \hat{d}_2, \ldots\right]$ are the durations. In addition to the relative time durations $\hat{\mathbf{d}}$, we also determine the absolute transition times $\hat{\mathbf{T}} = \left[\hat{T}_1, \hat{T}_2, \ldots\right]$ (relative to the beginning of the observations) by cumulatively summing over $\hat{\mathbf{d}}$: $\hat{T}_i = \sum_{j=1}^{i} \hat{d}_j$.

The combination of the recognized activity sequence $\hat{\mathbf{a}}$ in addition to either the durations $\hat{\mathbf{d}}$ or the schedule $\hat{\mathbf{T}}$ form one of the outputs of the plan recognizer.

Next, we determine the other output of the plan recognizer, the *predicted activity sequence*, which we denote as $\tilde{\mathbf{a}}$, and corresponding durations $\tilde{\mathbf{d}}$. We first refer back to the TPN to determine the plan that corresponds to the recognized activity sequence. The activities of this plan are encoded as $\mathbf{A}_s^*$, and corresponding mean durations are encoded as $\mathbf{D}_s^*$. The predicted activity sequence $\tilde{\mathbf{a}}$ is the same as the corresponding TPN plan $\mathbf{A}_s^*$, which encompasses all the activities in the recognized activity sequence $\hat{\mathbf{a}}$. The schedules of previously executed activities are the same as those in the recognized activity sequence. Specifically, if $h = \left|\hat{\mathbf{d}}\right|$ is the number of encoded durations in the recognized activity sequence and $n^* = |\mathbf{D}_s^*|$ is the number of encoded durations in the corresponding TPN path, then $\tilde{\mathbf{d}}_{1:h-1} = \hat{\mathbf{d}}_{1:h-1}$. Durations of future activities reflect the mean durations of those activities in the TPN, or $\tilde{\mathbf{d}}_{h+1:n^*} = (\mathbf{D}_s^*)_{h+1:n^*}$. The predicted duration of the current activity takes the max of either the current recognized activity duration or the corresponding activity duration in the TPN, or $\tilde{\mathbf{d}}_h = \max\left(\hat{\mathbf{d}}_h, (\mathbf{D}_s^*)_h\right)$. Finally, we can determine the schedule of the predicted activity sequence by cumulating over the durations: $\tilde{T}_i = \sum_{j=1}^{i} \tilde{d}_j$.

The combination of the predicted activity sequence $\tilde{\mathbf{a}}$ in addition to either the durations $\tilde{\mathbf{d}}$ or the schedule $\tilde{\mathbf{T}}$ form the final output of the plan recognizer.

5.7 Simple Example of the Plan Recognition Process

We now demonstrate the plan recognition process on a simple example.

Suppose we learned the TPN given in Figure 5-6, where the activities 1 and 2 are shown with the observed test sequence in Figure 5-7. The HMM converted from

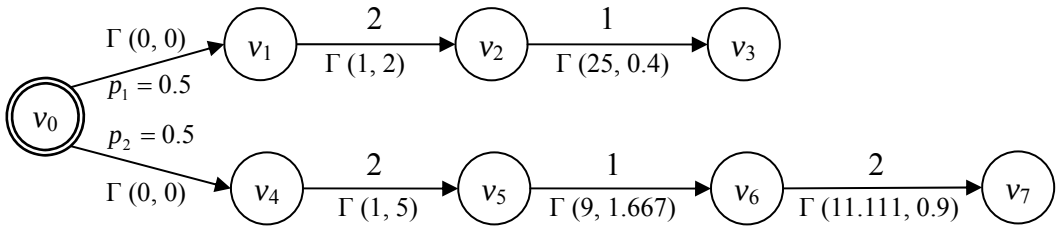


Figure 5-6: TPN of a simple example

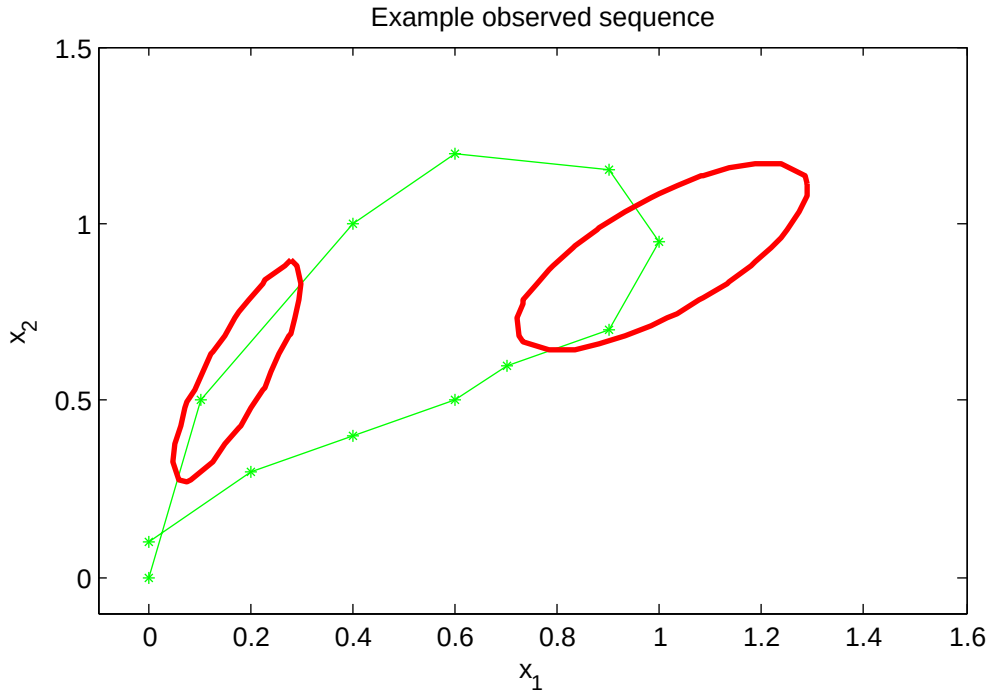


Figure 5-7: Activities shown with test sequence of example 2D data

the TPN is shown in Figure 5-8, where transition probabilities are derived from the gamma duration distributions in the TPN. After the running Viterbi algorithm on the

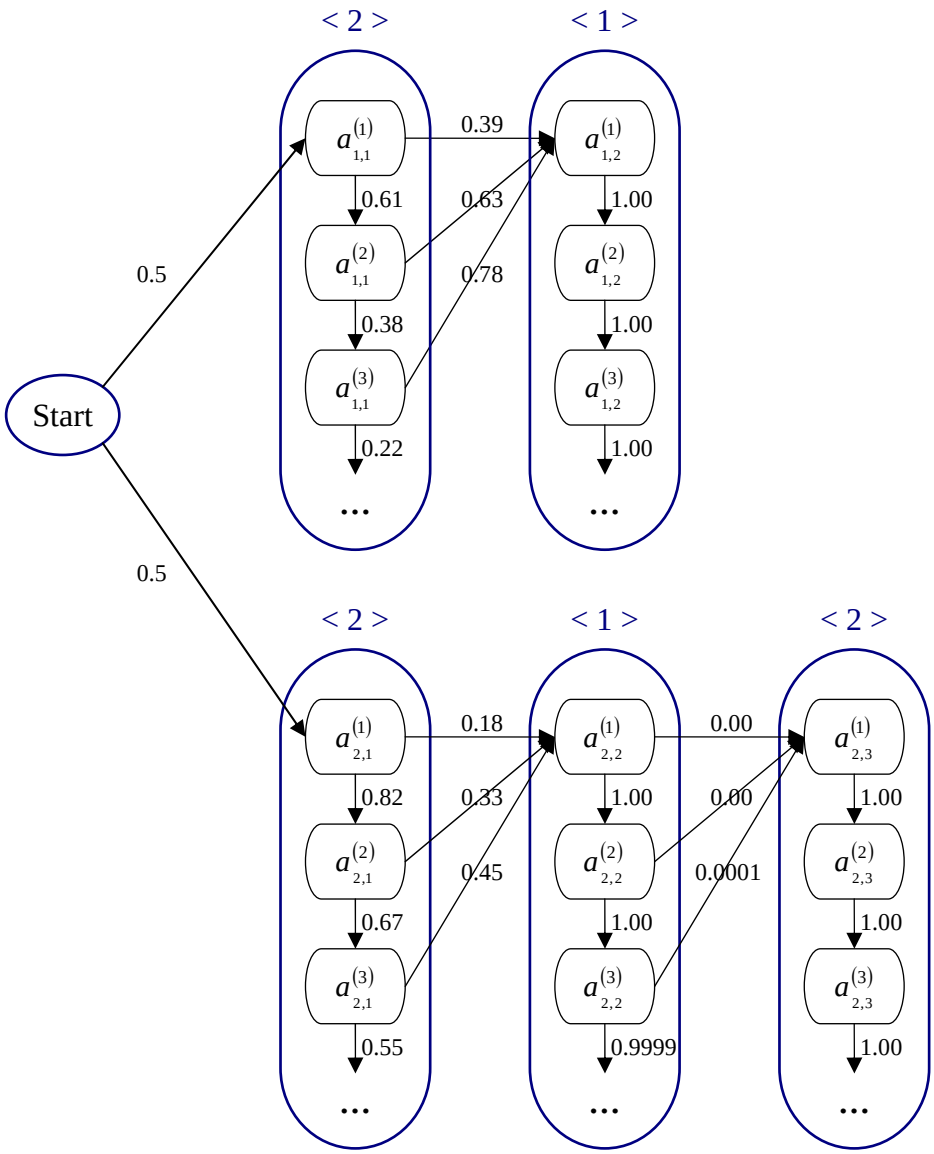


Figure 5-8: Hidden Markov model derived from the example TPN

HMM given the observed sequence, we obtain the recognized and predicted activity sequences in Figure 5-9.

Activity	Duration
2	3
1	7
2	1

(a) Recognized activity sequence

Activity	Duration
2	3
1	7
2	10

(b) Predicted activity sequence

Figure 5-9: Recognized and predicted activity sequences of example problem

Chapter 6

Implementation

This chapter presents the plan learning and plan recognition algorithms in more detail. The components of plan learning are described in Section 6.1, and include learning activity distributions with Expectation Maximization, obtaining activity and duration summaries for all training sequences, and creating a temporal plan network with this information. The components of plan recognition are discussed in Section 6.2, and include converting a TPN into an HMM, using Viterbi to recognize most likely trajectory, and referring back to the TPN to conclude the full predicted path.

Our research was implemented in a MATLAB 7.0 environment, with an additional `pointer` package for creating tree type data structures, an `xmltree` package for storing data structures as XML files, a Graph Theory Toolbox for displaying graphs, and a `C3DServer` package for reading data in `c3d` format. We will, however, present the implemented algorithms in an environment independent manner. The two parts, learning and recognition, are standalone modules that can be executed independently.

6.1 Plan Learning

Plan learning is designed to be executed offline. The plan learning pseudocode is laid out in Algorithm 6.1, and closely follows the operations in Chapter 4. We first obtain the training data using the GetData function, which outputs the training data in two ways: X, which is the set of all C training data sequences $\{X_1, \ldots, X_C\}$; and $\mathbf{X}$,

which is the concatenated vector of all the data, or $\mathbf{X} = [X_1, \ldots, X_C]^T$.

Algorithm 6.1 Plan Learning

1: $[X,\ \mathbf{X}] \leftarrow$ GetData ()
2: $[\boldsymbol{\mu},\ \boldsymbol{\sigma}] \leftarrow \mathrm{EM}\,(\mathbf{X},\ k)$
3: **for** $c = 1$ to C **do**
4: $\quad \mathbf{H}_c \leftarrow$ GetActivitySequence $(X_c,\ \boldsymbol{\mu},\ \boldsymbol{\sigma})$
5: **end for**
6: $tpn \leftarrow$ MakeTPN $(\{\mathbf{H}_1, \ldots, \mathbf{H}_C\},\ \boldsymbol{\mu},\ \boldsymbol{\sigma})$

Next, the vector of all training data $\mathbf{X}$ and the user-determined recognition resolution k denoting the number of activities are passed through the Expectation Maximization learning algorithm, which outputs a matrix $\boldsymbol{\mu}$ containing k vectors describing the multivariate mean of each activity, $[\mu_1, \ldots, \mu_k]^T$, and a matrix $\boldsymbol{\Sigma}$ containing the multivariate covariances of each activity, $[\Sigma_1, \ldots, \Sigma_k]^T$.

The EM implementation is laid out in Algorithm 6.2, and closely follows the description in Section 4.3. Although in theory the parameters of the Gaussian mixture can be initialized arbitrarily, in practice we set them to strategic values that do not cause machine precision errors. Specifically, the mixture weights w_j of each activity cluster are initialized uniformly; the multivariate activity means μ_j are initialized to k points chosen evenly out of the training data vector $\mathbf{X}$ to ensure that the Expectation step (line 17 in Algorithm 6.2) does not evaluate to values smaller than machine precision; and the covariances Σ_j of each activity are initialized to the covariance of the entire training vector. The Expectation and Maximization steps are repeated until convergence, which we define here to be the iteration at which the norm of the difference in the $\boldsymbol{\mu}$ vector between iterations is less than or equal to some small value δ.

After the multivariate activity parameters are learned, each training sequence is independently put through a GetActivitySequence function that produces activity and duration summaries for each sequence, the details of which are presented in Algorithm 6.3. This function records the most likely activity at each data point in the training sequence (lines 8 - 13) as a vector a. It then performs run-length encoding on a to generate the activity and duration summary.

Algorithm 6.2 Expectation Maximization

EM $(\mathbf{X},\ k)$

Input:

$\mathbf{X}$, all training data $[\mathbf{x}_1, \ldots, \mathbf{x}_N]^T$

k, number of activity clusters

Output:

$\boldsymbol{\mu}$, vector of multivariate activity means, $[\mu_1, \ldots, \mu_k]^T$

$\boldsymbol{\Sigma}$, vector of multivariate activity covariances, $[\Sigma_1, \ldots, \Sigma_k]^T$

Notable local variables:

p, $k \times N$ probability matrix, such that $p_{ji} = p(j \mid i)$, where j is an activity and i is a data point

{Initialize parameters for start of algorithm}
for $j = 1$ to k **do**
    $w_j \leftarrow \frac{1}{k}$
    $\mu_j \leftarrow \mathbf{x}_q,\ q = \left\lfloor \frac{N}{k+1} \cdot j \right\rfloor$
    $\Sigma_j \leftarrow \text{Cov}(\mathbf{X})$
end for

while $\|\mu - \mu_{old}\| > \delta$ **do**
    {E-step: label data}
    for $i = 1$ to N **do**
        for $j = 1$ to k **do**
            $p_{ji} \leftarrow \text{MultiNormPDF}(\mathbf{x}_i,\ \mu_j,\ \sigma_j) \cdot w_j$
        end for
    end for
    Normalize p so that the probabilities at each time step sum to unity

    {M-step: update parameters}
    for $j = 1$ to k **do**
        $n_j \leftarrow \sum_{i=1}^{N} p_{ji}$
        $w_j \leftarrow \frac{n_j}{N}$
        $\mu_j \leftarrow \frac{1}{n_j} \sum_{i=1}^{N} p_{ji} \mathbf{x}_i$
        $\Sigma_j \leftarrow \frac{1}{n_j} \sum_{i=1}^{N} p_{ji} (\mathbf{x}_i - \mu_j)(\mathbf{x}_i - \mu_j)^T$
    end for
    $\mu_{old} \leftarrow \mu$
end while

Algorithm 6.3 Get Activity Sequence

GetActivitySequence $(X_c,\ \boldsymbol{\mu},\ \boldsymbol{\Sigma})$

Input:

X_c, one sequence of training data $[\mathbf{x}_1, \ldots, \mathbf{x}_n]^T$

$\boldsymbol{\mu}$, vector of multivariate activity means, $[\mu_1, \ldots, \mu_k]^T$

$\boldsymbol{\Sigma}$, vector of multivariate activity covariances, $[\Sigma_1, \ldots, \Sigma_k]^T$

Output:

$\mathbf{H}_c$, matrix of activity and duration summary vectors, $[\mathbf{A}_c,\ \mathbf{D}_c]$

Notable local variables:

probs, $n \times k$ probability matrix of data given an activity

a, $n \times 1$ vector of most likely activity at each data point

```
for i = 1 to n do
    for j = 1 to k do
        probs_ij ← MultiNormPDF (x_i, μ_j, Σ_j)
    end for
    a_i ← argmax_j (probs_i)
end for
H_c ← RunLengthEncoding (a)
```

After finding the activity and duration summaries for each training sequence, we are ready to create a temporal plan network. We encode TPNs as a tree type data structure, where each node in the tree is an activity. The variable *tpn* is a pointer to the root node of the tree that represents the TPN. Each activity node in *tpn* contains at least the following fields:

- *start*, the start event of activity, represented by a number
- *end*, the end event of activity, represented by a number
- *name*, the activity number
- μ, the multivariate Gaussian mean of the activity
- Σ, the multivariate Gaussian covariance of the activity
- *times*, vector of all durations for this activity from training data
- $times_k$, the k gamma parameter of the *times* vector (optional)
- $times_\theta$, the θ gamma parameter of the *times* vector (optional)

- *prob*, the prior probability of the trajectory

Algorithm 6.4 Make Temporal Plan Network

MakeTPN $(H, \boldsymbol{\mu}, \boldsymbol{\sigma})$

Input:

H, the set of all training data summaries $\{\mathbf{H}_1, \ldots, \mathbf{H}_C\}$

$\boldsymbol{\mu}$, vector of multivariate activity means, $[\mu_1, \ldots, \mu_k]^T$

$\boldsymbol{\Sigma}$, vector of multivariate activity covariances, $[\Sigma_1, \ldots, \Sigma_k]^T$

Output:

tpn, pointer to root node of tree that represents the TPN, initialized **null**

for all $\mathbf{H}_c \in H$ **do**
  if $\mathbf{A}_c$ is a new activity sequence **then**
    Add $\mathbf{A}_c$ as a new trajectory in *tpn* by doing the following:
    Add $\mathbf{A}_c$ to the *name* fields at each activity node in the new trajectory
    Ensure that each activity's *start* is the previous activity's *end*
    Add $\mathbf{D}_c$ to the *times* vectors at each activity in the new trajectory
    $\mu \leftarrow \mu_{name}$ for each activity in the trajectory
    $\Sigma \leftarrow \Sigma_{name}$ for each activity in the trajectory
  else
    {$\mathbf{A}_c$ has been encountered before}
    Find the existing trajectory $\mathbf{A}_c$ in *tpn*
    Add $\mathbf{D}_c$ to the *times* vectors at each activity in that trajectory
    $[times_k,\ times_\theta] \leftarrow$ GammaFit $(times)$ for each activity in the trajectory
  end if
end for
For each existing trajectory in *tpn*, $prob \leftarrow |times| / C$

An activity that follows another will have a *start* event that is the same as the preceding activity's *end* event. The *times* vector contains all the duration values of the current activity that are part of the current trajectory in the training data. In other words, the training data that describe the same trajectory will have the same activity sequence, but perhaps not the exact same durations, so the *times* vector at each activity contains all the potentially different durations at that activity. The $times_k$ and $times_\theta$ variables are the parameters obtained from performing a gamma distribution fit to the durations in the *times* vector. The prior probability *prob* of each activity is the ratio of the number of times this trajectory appears, to the total number of training trajectories.

The MakeTPN function goes through each activity and duration summary of

the training data as shown in Algorithm 6.4. It maintains a list of existing activity trajectories. If an activity summary sequence has not been seen before, this sequence is added as a new trajectory to the TPN by performing the actions in lines 9 - 13. If the activity summary sequence is in the list of existing trajectories, then the *times* vector gets updated with the new duration value, and the gamma distribution parameters $times_k$ and $times_\theta$ are recalculated. Finally, the prior probabilities *prob* are calculated by the ratio of the number of elements in the *times* vector to the total number of data sequences, to indicate how often the current trajectory occurs in training data.

6.2 Plan Recognition

The plan recognition procedure is outlined in Algorithm 6.5, and closely follows the operations in Chapter 5. First, it obtains the observed test data $\mathbf{X}_{obs}$, which are expressed in the same principal components as the training data. Next, information from the temporal plan network is extracted to be represented in a Hidden Markov Model so that we can run the Viterbi algorithm on it to extract the most likely path.

Algorithm 6.5 Plan Recognition

1: $\mathbf{X}_{obs} \leftarrow$ GetTestData ()
2: $\mathbf{a} \leftarrow$ GetHMMStatesFromTPN (tpn)
3: $[m,\ \phi] \leftarrow$ Viterbi ($\mathbf{X}_{obs},\ \mathbf{a}$)
4: $[recognizedPathSoFar,\ predictedTraj] \leftarrow$ GetPath ($m_{n_{obs}},\ \phi$)

The function GetHMMStatesFromTPN as presented in Algorithm 6.6 returns a look up table **a** containing activity numbers, max durations, self-transitioning probabilities, prior probabilities, activity means, and activity covariances for each trajectory. The max duration is determined by the time at which the probability of sampling a duration longer than the current time is less than some small ϵ in the gamma distribution. At each time step, the probability of making a self-transition, or continuing with the same activity, is just the probability that the duration of said activity is longer than the current elapsed time in the activity. Since we deal with discrete time

Algorithm 6.6 Get HMM states from TPN

1: GetHMMStatesFromTPN (tpn)

Input:

2: tpn

Output:

3: **a**, look up table, initialized **null**

4: $\epsilon \leftarrow$ some small value (eg. 0.001)
5: $curArcs \leftarrow$ NextArcsInTPN $(tpn,\ 0)$ {the arcs in TPN with $start = 0$}
6: **a** $\leftarrow$ GetStates $(curArcs,\ \textbf{null},\ tpn,\ \epsilon)$

Subfunction:

7: GetStates $(curArcs,\ \textbf{a},\ tpn,\ \epsilon)$
8: **if** IsNull $(curArcs)$ **then**
9: **a** $\leftarrow$ **null**
10: **else**
11: {Get the activity information}
12: $curName \leftarrow curArcs_1.name$
13: $\mu \leftarrow curArcs_1.\mu$
14: $\Sigma \leftarrow curArcs_1.\Sigma$

15: {Get the activity durations up to $1-\epsilon$ from gamma CDF distribution}
16: $maxDur \leftarrow \lceil$InvGammaCDF $(1-\epsilon,\ curArcs_1.times_k,\ curArcs_1.times_\theta)\rceil$

17: {Get self transition probabilities p_{ii}, given by duration gamma}
18: $i \leftarrow [1, \ldots, maxDur]$
19: $pSelf \leftarrow 1 -$ GammaCDF $(i,\ curArcs_1.times_k,\ curArcs_1.times_\theta)$

20: {Get prior probabilities for the start of each trajectory}
21: **if** $curName = 0$ **then**
22: $prior \leftarrow curArcs_1.prob$
23: **end if**

24: {Run recursive function to get this information for whole TPN}
25: **a** $\leftarrow \begin{bmatrix} \{curName,\ maxDur,\ pSelf,\ prior,\ \mu,\ \Sigma\} \\ \text{GetStates}\left(\text{NextArcsInTPN}\left(tpn,\ curArcs_1.end\right),\ \textbf{a},\ tpn,\ \epsilon\right) \\ \text{GetStates}\left(curArcs_{2:\text{Length}(curArcs)},\ \textbf{a},\ tpn,\ \epsilon\right) \end{bmatrix}$
26: **end if**

steps, we record the self-transition probabilities at all the discrete time steps in an activity up to the max duration in a vector $pSelf$. The other items in the look up table are taken directly from the TPN. We add activity information from the TPN to the look up table recursively in a depth first manner so that all activities in one trajectory are conveniently listed in order.

The structure of the HMM is implicitly encoded in the look up table. The states of the model consist of all possible activities at all possible time steps until the max duration at each activity. In other words, there are as many states in the HMM as the sum of all the $maxDur$ values in the look up table, which we will call N_{HMM}. All transition probabilities in the HMM are implicitly given by the self-transition probabilities in the look up table because at each time step, an activity can only transition to itself or the following activity.

In preparation for running Viterbi in Algorithm 6.7 to find the most likely path, we first extract the transition probability matrix T and prior probabilities $prior$ from the look up table. The observation probabilities are obtained from the multivariate Gaussian activity parameters. The Viterbi algorithm closely follows the description given in Section 5.5. The first part of Algorithm 6.7 up to line 16 initializes the parameters in preparation for updating the Viterbi messages through all the time steps in the observed sequence in lines 17 - 26. The implementation takes advantage of the sparsity of the matrices to reduce computation by only calculating observation probabilities at non-zero values of the message. The Viterbi messages throughout all time steps are recorded in a matrix m, and the back pointer indexes at every time step are recorded in a matrix ϕ.

Using the back pointer matrix, we can retrace the most likely path determined by Viterbi with Algorithm 6.8. Taking a run-length encoding of this recognized path gives the $recognizedPathSoFar$, which is the currently recognized most likely partial trajectory. Knowing the temporal plan network, however gives the advantage that we can predict the entire trajectory. The function ExtractCorresponding-PathInTPN in line 17 returns the activity and duration summary for the trajectory corresponding to the recognized partial trajectory in the TPN. Finally, the true pre-

Algorithm 6.7 Viterbi

1: VITERBI $(\mathbf{X}_{obs}, \mathbf{a})$

Input:

2: $\mathbf{X}_{obs}$, observed data sequence

3: $\mathbf{a}$ look up table

Output:

4: m, $N_{HMM} \times n_{obs}$ matrix of Viterbi messages at all HMM states for each time step

5: ϕ, $N_{HMM} \times n_{obs} - 1$ matrix of back pointer indexes

Notable local variables:

6: T, $N_{HMM} \times N_{HMM}$ transition matrix: T_{ij} is probability of transitioning $i \rightarrow j$

7: $prior$, $1 \times N_{HMM}$ prior probabilities

8: o, $1 \times N_{HMM}$ observation probabilities at current time step

9: $T \leftarrow$ GETHMMTRANSITIONS $(\mathbf{a})$

10: $prior \leftarrow$ GETHMMPRIOR $(\mathbf{a})$

11: $priorIndexes \leftarrow$ non-zero indexes of $prior$

12: **for** $i \in priorIndexes$ **do**

13: $\quad j \leftarrow$ GETACTIVITYNUMBERFROMHMMINDEX $(i,\ \mathbf{a})$

14: $\quad o_i \leftarrow$ GETHMMOBSERVED $((\mathbf{X}_{obs})_1,\ \mathbf{a},\ j)$

15: **end for**

16: $m_1 \leftarrow \alpha \cdot prior \cdot o$

17: **for** $t = 2$ to n_{obs} **do**

18: $\quad mm \leftarrow \max_i (T_{ij} \cdot m_{t-1,j})$

19: $\quad \phi_{t-1} \leftarrow \text{argmax}_i (T_{ij} \cdot m_{t-1,j})$

20: $\quad mmIndexes \leftarrow$ non-zero indexes of mm

21: $\quad$ **for** $i \in mmIndexes$ **do**

22: $\quad\quad j \leftarrow$ GETACTIVITYNUMBERFROMHMMINDEX $(i,\ \mathbf{a})$

23: $\quad\quad o_i \leftarrow$ GETHMMOBSERVED $((\mathbf{X}_{obs})_t,\ \mathbf{a},\ j)$

24: $\quad$ **end for**

25: $\quad m_t \leftarrow \alpha \cdot o \cdot mm$

26: **end for**

dicted full trajectory ensures that the durations of the past activities agree with those in $recognizedPathSoFar$.

Algorithm 6.8 Get Path

GetPath $(m_{last},\ \phi,\ \mathbf{a})$

Input:

m_{last}, message at the last time step

ϕ, matrix containing history of max message indexes

$\mathbf{a}$, HMM look up table

Output:

$recognizedPathSoFar$, recognized path for observed sequence

$predictedTraj$, predicted trajectory based on observed sequence and TPN

Notable local variables:

$path$, $n_{obs} \times 1$ vector containing back pointer indexes

$apath$, $n_{obs} \times 1$ vector containing activity number corresponding to $path$

$path_{n_{obs}} \leftarrow \text{argmax}_j\,(m_{last})$

$apath_{n_{obs}} \leftarrow$ GetActivityNumberFromHMMIndex$(path_{n_{obs}},\ \mathbf{a})$

for $i = 1$ to number of time steps in ϕ **do**

  Let $j \leftarrow n_{obs} - i$

  $path_j \leftarrow \phi_{j,path_{j+1}}$

  $apath_j \leftarrow$ GetActivityNumberFromHMMIndex $(path_j,\ \mathbf{a})$

end for

$recognizedPathSoFar \leftarrow$ RunLengthEncoding $(apath)$

$predictedTraj \leftarrow$ ExtractCorrespondingPathInTPN $(path,\ \mathbf{a})$

$rLen \leftarrow$ length of $recognizedPathSoFar$

$predictedTraj_{1:rLen-1,2} \leftarrow recognizedPathSoFar_{1:rLen-1,2}$

$predictedTraj_{rLen,2} \leftarrow \max\,(predictedTraj_{rLen,2},\ recognizedPathSoFar_{rLen,2})$

Chapter 7

Results

We will now run our plan learning and recognition algorithms on the Vicon Motion Systems data discussed in Section 3.1 to test the capabilities and analyze the performance of our algorithms. We will look at the results on two different sets of data. One set contains several different ballet dance moves, and the other contains common motions during golf. We imagine that a robot may take on the role of a coach or caddie, who can remind a dancer of the next move in his or her prepared program, or prepare the correct club or ball to the golfer at the right time. Although these are mostly recreational motions, we can imagine more serious scenarios during which plan recognition can play an important role, such as a space ship monitor that can remind an astronaut of the next task in an extravehicular procedure, or a nurse who must hand a surgeon the right medical tool. One reason for choosing these particular motion capture data for testing is that there were multiple executions of different but related motions available in the database.

7.1 Evaluation of Success

We must first discuss what it means for a plan to be successfully learned. The input of the plan learning process is a set of training data that can represent a variety of different motions. For example, we might have five training trajectories for reaching and picking up an object, and three trajectories for reaching and pushing away an

object. We, the human operators, know the "true" motion of each training data beforehand. Plan learning performs learning on all the training data to try to group the similar motions together. The output of the plan learning process is a network of s possible motion plans. To evaluate the success of the plan learner, we compare what we know about the training data to the plan network given by the plan learner. The outcome may contain two undesirable conditions: multiple "true" motions are learned as the same plan in the network, which we call a "lumping error;" or the same real motion is learned as two different plans in the network, which we call a "splitting error." Since we are interested in distinguishing different motions, a lumping error is worse than a splitting error, i.e. it is particularly bad if the plan learner was unable to distinguish the different motions and lumped them all as the same plan.

We assign a value of -2 for every instance of a lumping error, and a value of -1 for every instance of a splitting error. The user can define the level of tolerable error. For example, for a given run of the plan learning algorithm, we can choose the maximum allowable number of error units to be at most twice the number of different "true" motions in the training data. Thus an execution of plan learning is considered successful if the number of error units it made is less than twice the number of different motions in the training data.

During recognition, an observed sequence is identified as one of the plans in the plan network. We, the human operators, know the "true" motion of the observed sequence. We also know the true motion(s) in the training set that generated the plan that the recognizer identified. If the true motion of the observed sequence matches at least one of the motions that generated the identified plan, then we consider the recognition process successful.

7.2 Dance Data

The first three principal components of the training data for three different dance motions is shown in Figure 7-1. The training data has been normalized so that all motions start from the same place. We used splines to fabricate additional data

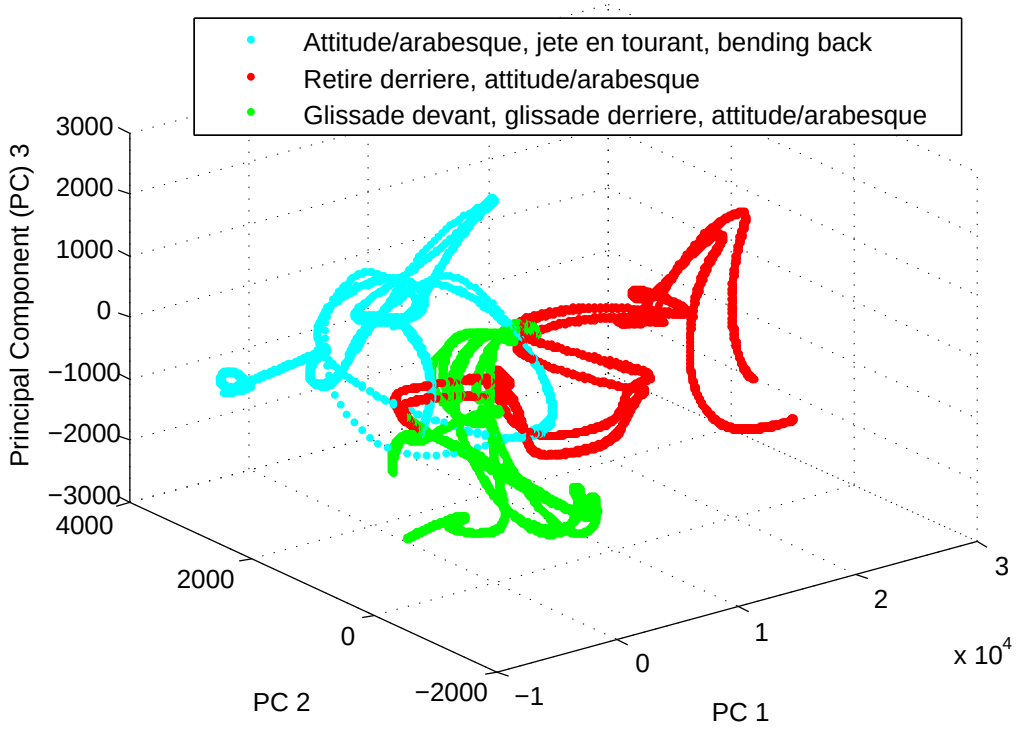


Figure 7-1: Training data for dance motions

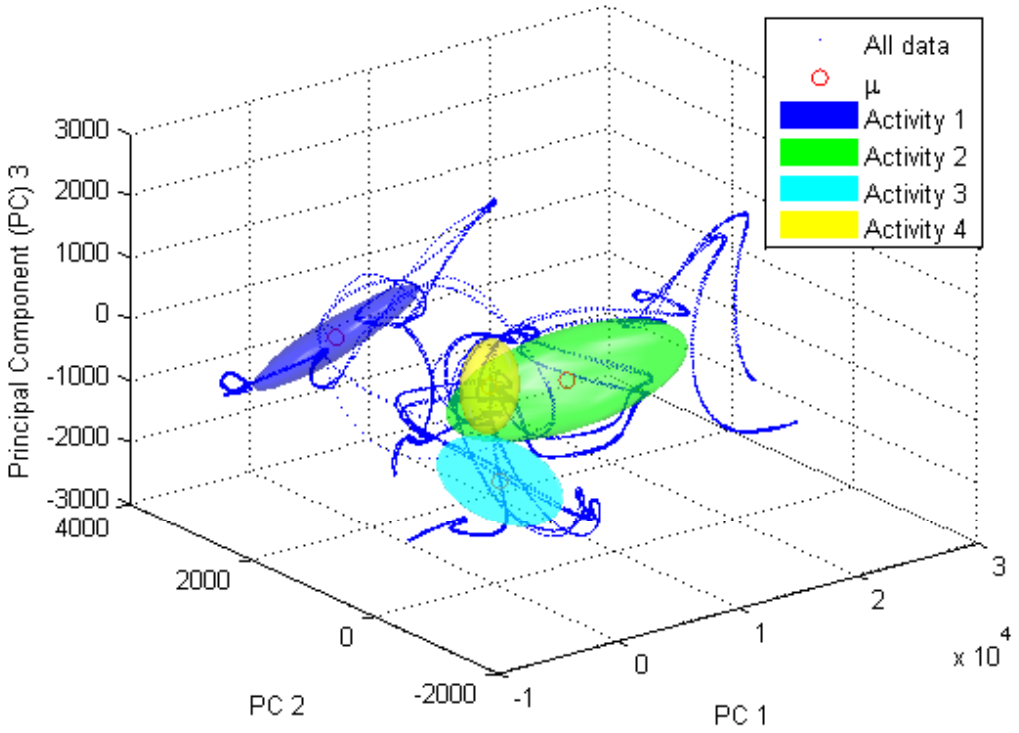


Figure 7-2: Activity clusters for dance data

to bolster the original due to data scarcity.

Figure 7-2 shows the results of running unsupervised learning with four activity clusters on the three principal components of the training data. Running the EM algorithm on a total of 9645 data points and recognition resolution of 4 took about 14 minutes on a standard PC. The plan learning is executed offline, so this is a reasonable execution time. The execution time consisted of 31 EM iterations to reach a convergence factor of $\delta = 1$.

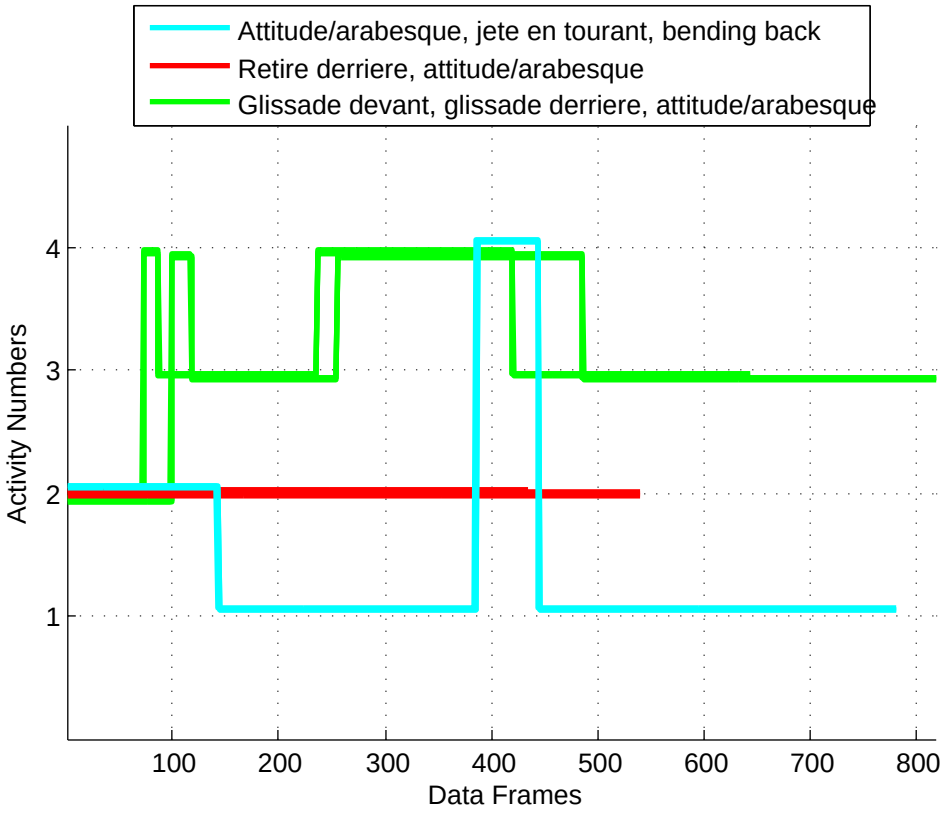


Figure 7-3: Activity trajectories for dance data with 4 activities

Sending each training sequence through Algorithm 6.3 to obtain the trajectories produces results shown in Figure 7-3. We can now see the distinctly different activity sequences and corresponding durations. Finally, we can create the temporal plan network shown in Figure 7-4. In this case, the plan learner successfully learned the plan network because it made no lumping or splitting errors, producing exactly three plans for our three different dance motions.

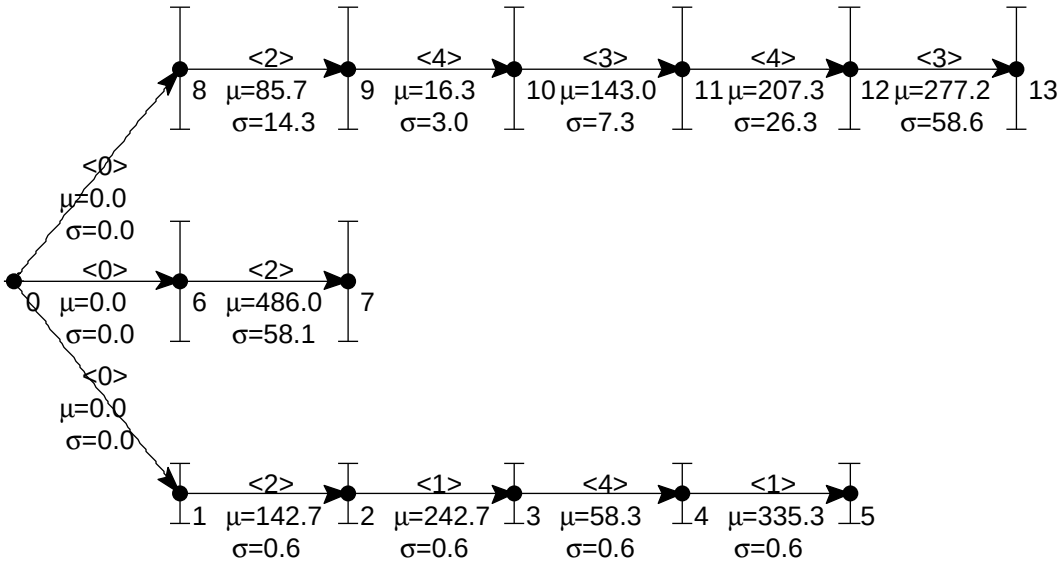


Figure 7-4: Output TPN of plan learner on dance motions. Inside angle brackets $\langle a \rangle$ are activity numbers; μ and σ are mean and standard deviations of activity duration.

We are now ready to do some testing with observed data. We run the recognizer after 150, 400 and all time steps of the observed sequence, producing the results shown in Figure 7-6. Note that initially after 150 time steps, the recognizer predicted an incorrect most likely trajectory, which is not surprising since the person does not do much during the first 150 time steps, or 1.25 seconds, that strongly differentiates the motion. After 400 time steps, or about 3.3 seconds, the recognizer is clearly predicting the correct trajectory. Finally, with the entire observed sequence, the algorithm is

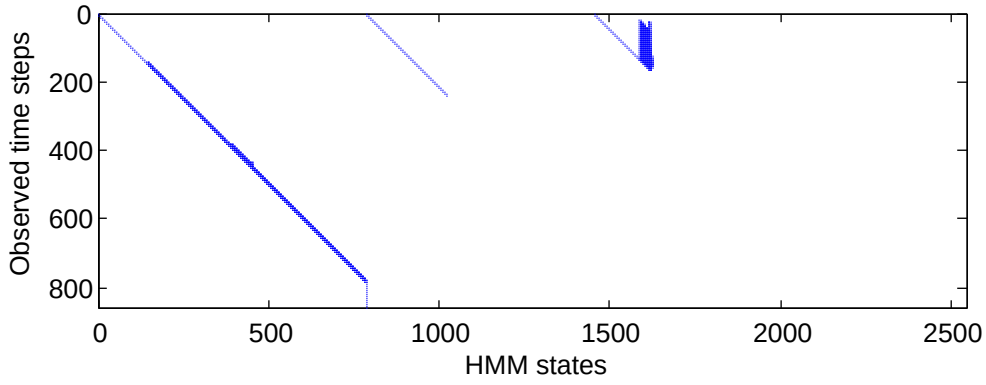
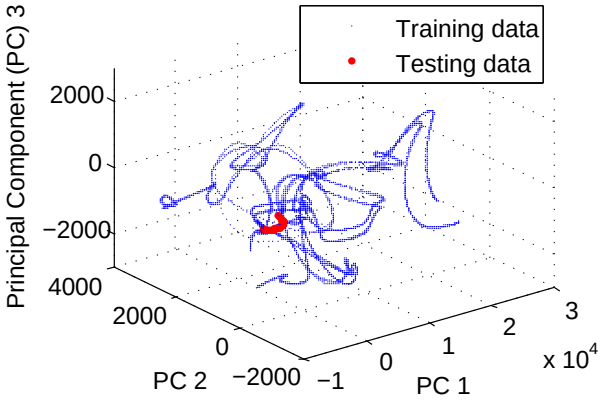


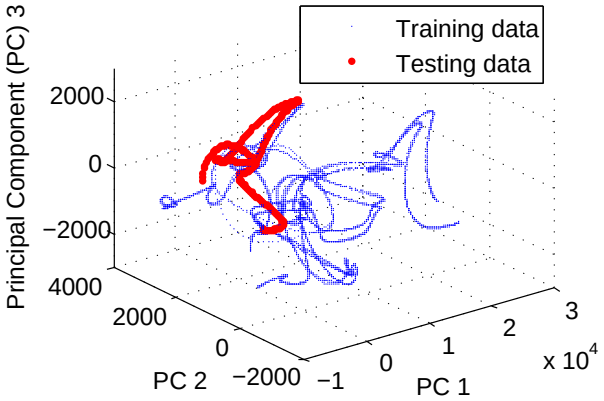
Figure 7-5: A look at the messages m over all observation time steps. The non-zero elements of the matrix are indicated by a dot.



(a) Partial dance test data with 150 time steps

Activity	Duration
2	486

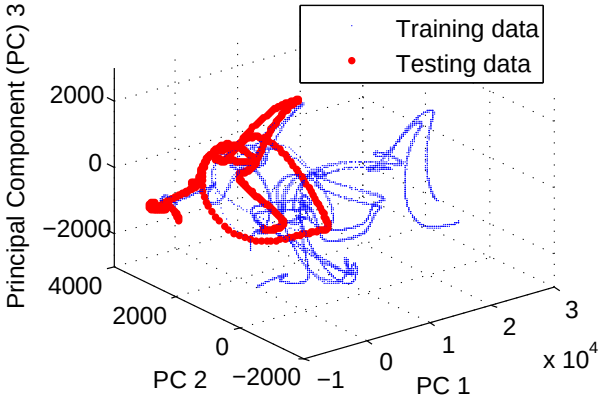
(b) Predicted activity sequence for 150 steps data



(c) Partial dance test data with 400 time steps

Activity	Duration
2	145
1	245
4	58.3
1	335.3

(d) Predicted activity sequence for 400 steps data



(e) Full dance test data

Activity	Duration
2	145
1	245
4	60
1	409

(f) Recognized activity sequence for full dance data

Figure 7-6: Test results for dance motion with different number of time steps in observed data sequence

able to recognize the durations that correspond to the identified activity sequence.

We can take a look at the resulting Viterbi messages over all the observed time steps in Figure 7-5. The dots in the figure indicate the messages with non-zero values. Of the 2542 HMM states, three states had initially non-zero prior probabilities, which correspond to the beginning of the three trajectories in the TPN. The message evolution over the time steps eventually eliminated the two incorrect trajectories after around 230 time steps. The vertical dot patterns in the figure reflect the non-zero transition probabilities from one activity at different time points to the beginning of the following activity.

7.3 Golf Data

We now run the learning and recognition algorithms through some golf motion data. Of the 30 data sequences available in the database, we used 27 for training data and the remaining 3 for testing. No splines were used because we had at least 4 training data for each motion. The 3 principal components of the training data for the different golf motions are shown in Figure 7-7.

Figure 7-8 shows the results of running unsupervised learning with five activity clusters on the three principal components of the golf training data. Running the EM algorithm on a total of 13,564 data points and recognition resolution of $k = 5$ took about 34 minutes on a standard PC. This execution time included 35 iterations to reach the convergence factor of $\delta = 1$. The complexity of each iteration in the EM algorithm is $O(N \cdot k)$, where N is the number of data points and k is the number of activity clusters.

Sending each training sequence through Algorithm 6.3 to obtain the trajectories produces results shown in Figure 7-9, and are summarized by the temporal plan network shown in Figure 7-10.

We can see that the swing and putt motions are sufficiently different from the other motions that they can be easily distinguished. Specifically, the bottom three trajectories in the TPN in Figure 7-10 summarize slightly different types of swing

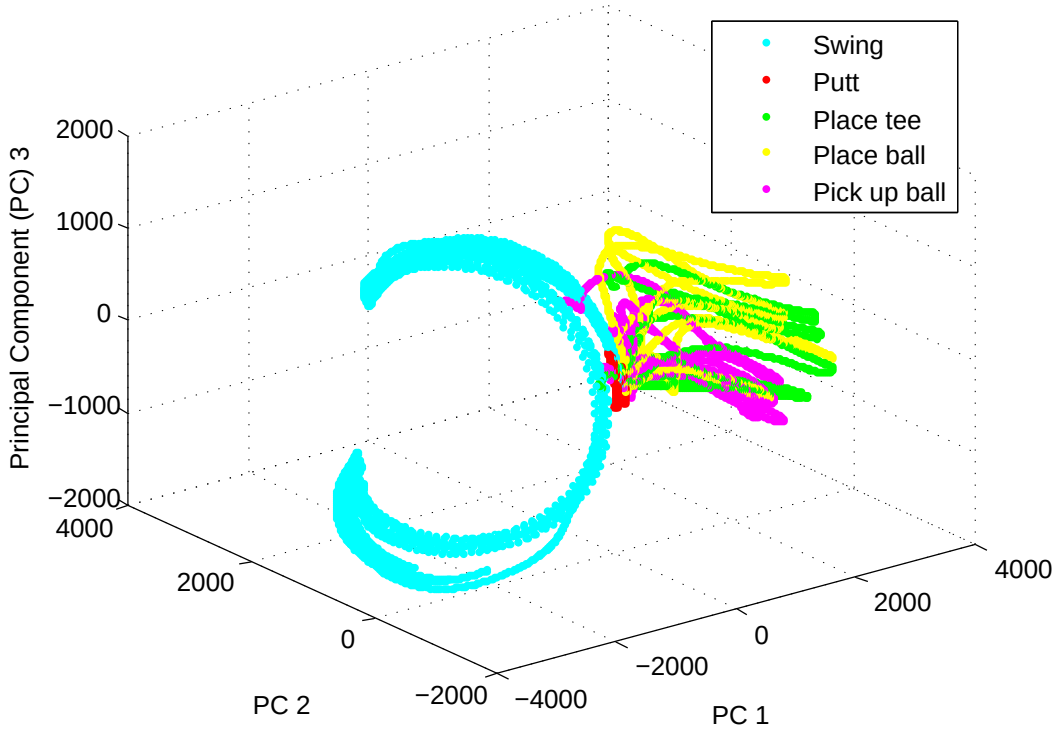


Figure 7-7: Training data for golf motions

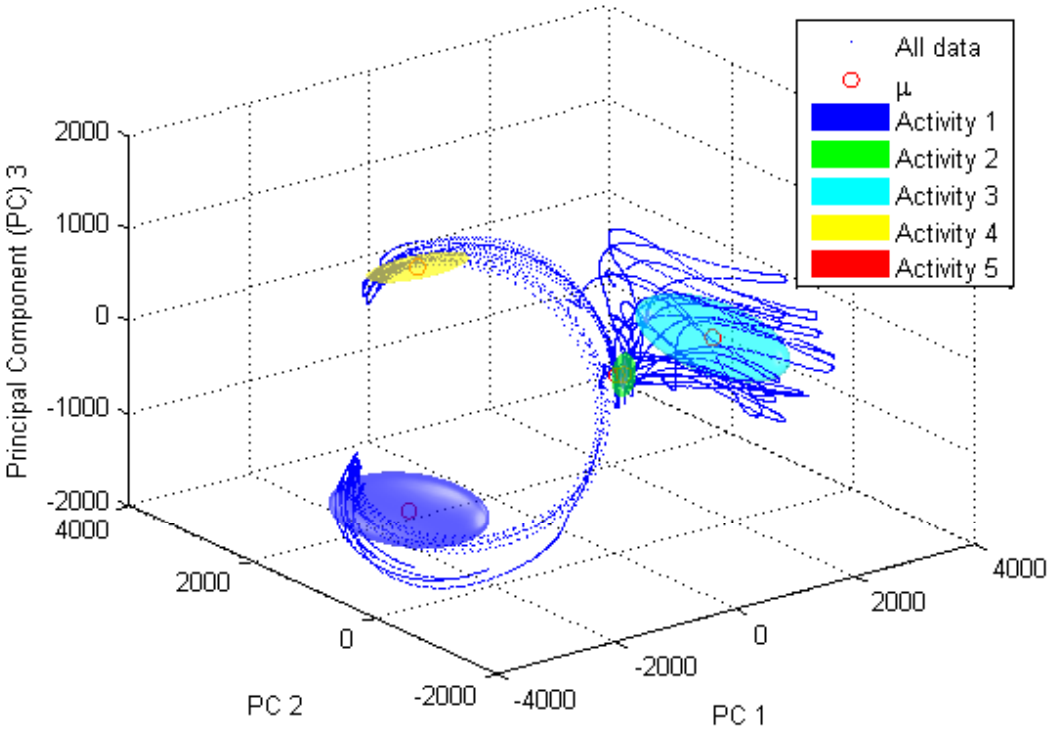


Figure 7-8: Activity clusters for golf data

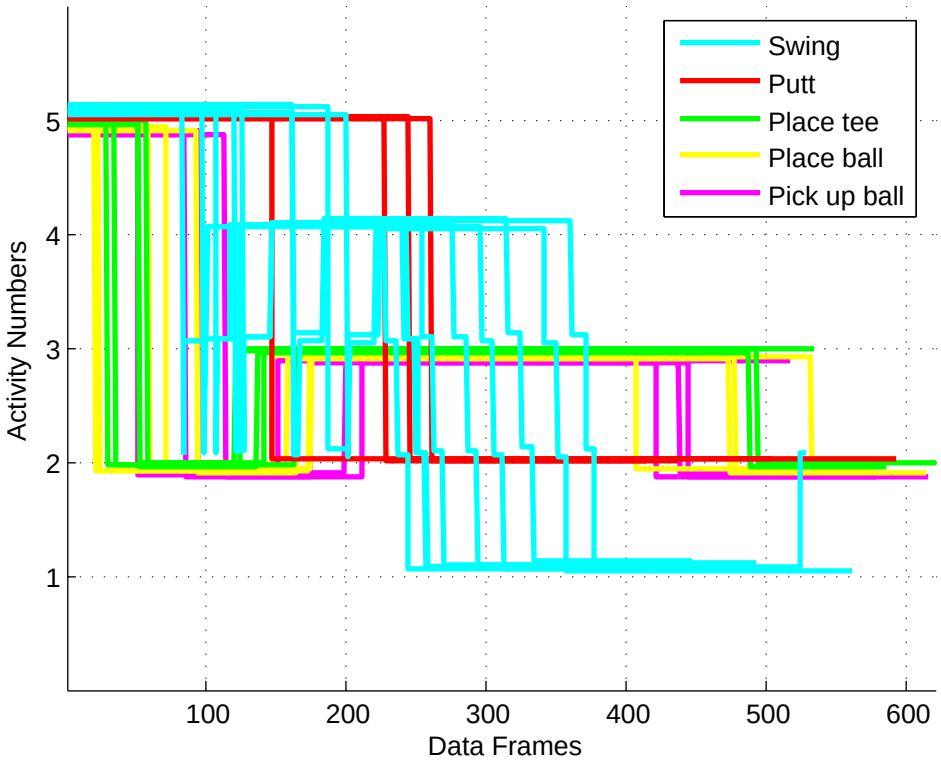


Figure 7-9: Activity trajectories for golf data with 5 activities

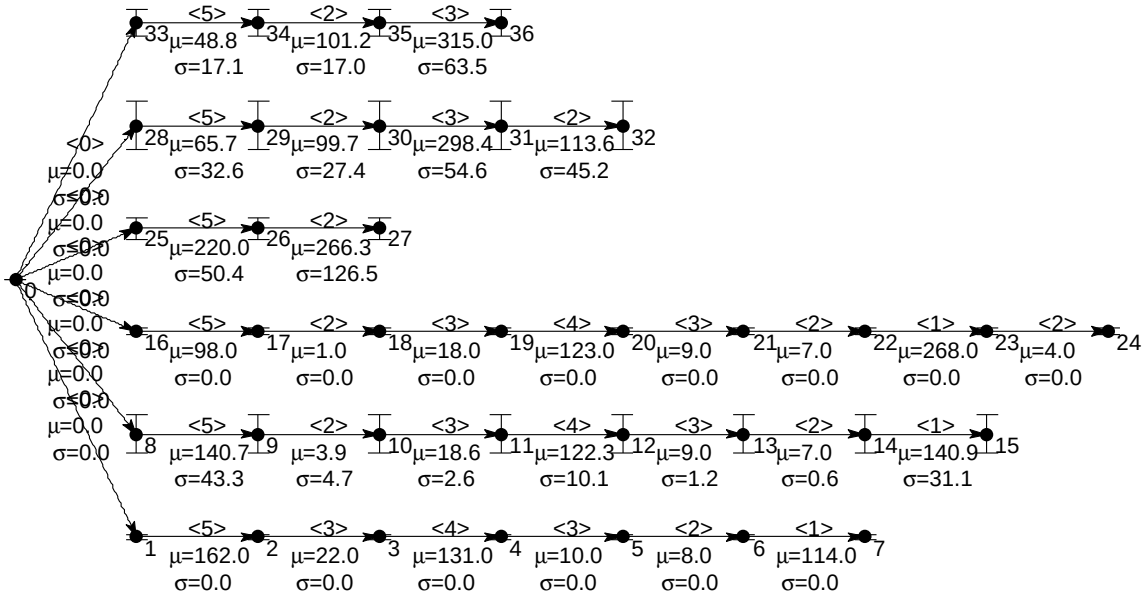
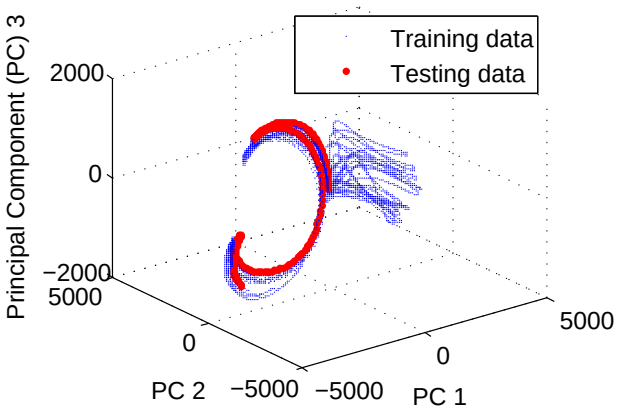


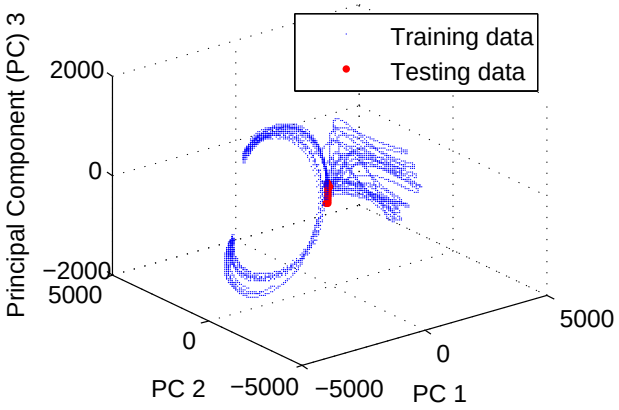
Figure 7-10: TPN learned from golf data assuming 5 activities



(a) Swing

Activity	Duration
5	88
2	7
3	20
4	118
3	9
2	8
1	140.9

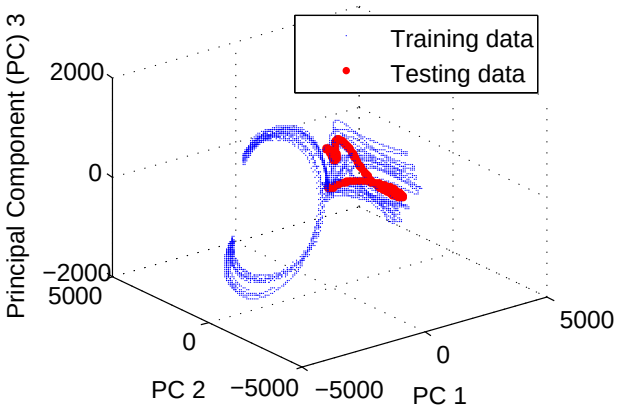
(b) Recognized activity sequence for golf swing



(c) Putt

Activity	Duration
5	169
2	389

(d) Recognized activity sequence for golf putt



(e) Pick up ball

Activity	Duration
5	89
2	80
3	283
2	116

(f) Recognized activity sequence for picking up ball

Figure 7-11: Test results for three different golf motions

motions, thus incurring an error of -2 for splitting twice. The trajectory of $\langle 5 \rangle, \langle 2 \rangle$ summarizes the distinct putt motion. On the other hand, the motions of placing tee, placing ball and picking up ball all consist of the person bending over and reaching. Since the position of objects are not tracked in these motion capture data, it is unsurprising that the recognizer has a difficult time distinguishing between these motions, as it was hard for us, also, to tell apart the different bending motions when viewing the skeletal animations. In this case, the plan learner splits all three bending motions across two plans, incurring errors of -3 for splitting, -2×2 for lumping for each of the 2 plans. Thus in total, the plan learner makes a total error of -13, which fails our success criteria.

Although the plan learner is unable to distinguish the three bending motions, we still run the recognition algorithm on our three test cases to see how it performs on the portion of the plan network that was distinguished, i.e. the swing and putt motions. The results of recognizing the three test motions is shown in Figure 7-11. We see that each of the motions are recognized correctly. The run time for recognizing

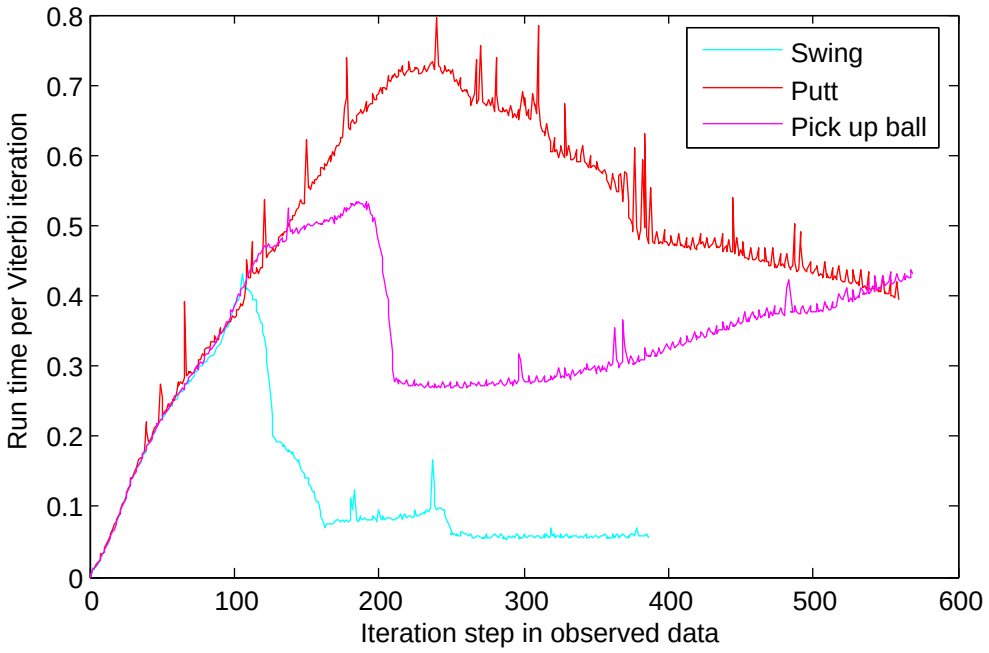


Figure 7-12: Run times for individual iterations during recognition for swing, putt, and pick up ball motions

a whole observation sequence is on the order of 10 to 200 seconds on a standard PC. However, if the recognition algorithm were run incrementally, each recognition iteration occurs within less than 0.8 seconds. Figure 7-12 shows the run times at each iteration for the different motions. We notice that the run times increase when there are more possibilities for recognizing the observed data, and decrease rapidly after some distinguishing action is detected and many possibilities are eliminated.

To seek better performance from the plan learner with the golf data, we re-ran the plan learning algorithm using 7 principal components of the 102 dimensional data with 5 activities, which took roughly 140 minutes of offline computation time. Figure 7-13 shows the activity sequences of the training data, and we can see that there is slightly more differentiation in the bending motions than learning with only the three principal components before. Figure 7-14 is the temporal plan network given by the learner. We see that all the swing motions have been grouped as the

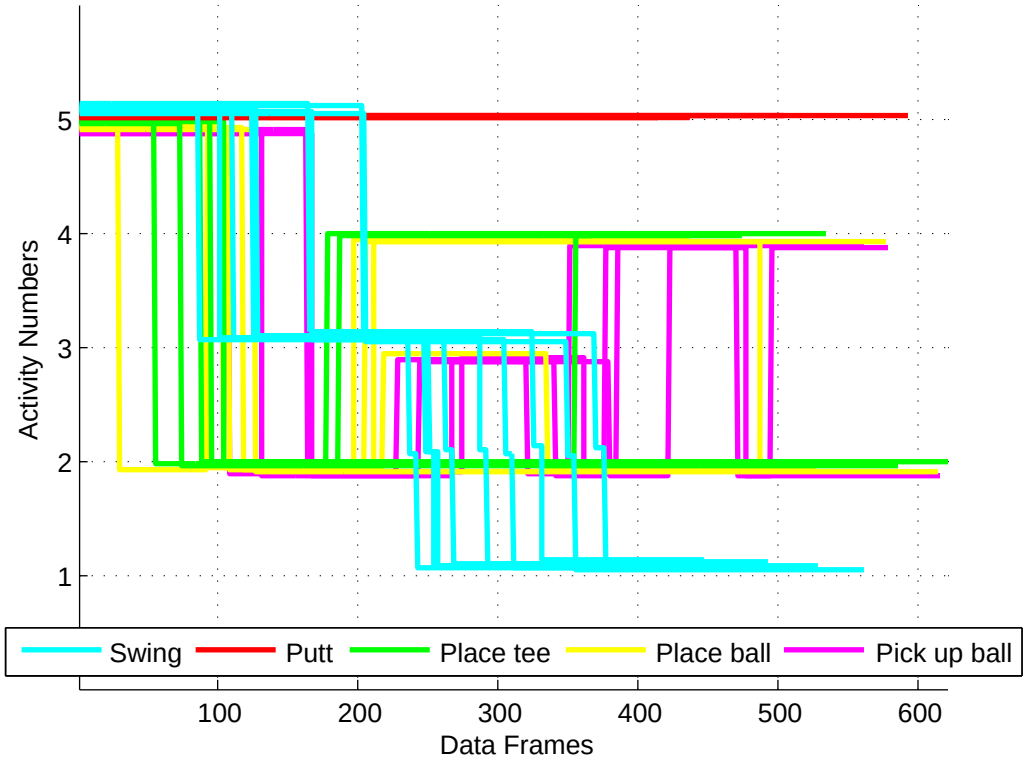


Figure 7-13: Activity trajectories for full golf data using 7 principal components, with 5 activities

bottom trajectory in the TPN, and all the putt motions as the next two. The bending

motions, however, have been differentiated a bit more. For example, the movement to pick up a ball are described by the top three trajectories in the TPN beginning with $\langle 5\rangle, \langle 2\rangle, \langle 3\rangle, \langle 2\rangle, \langle 4\rangle$. The total error in this TPN is -9, which is significantly better than the performance using only three principal components.

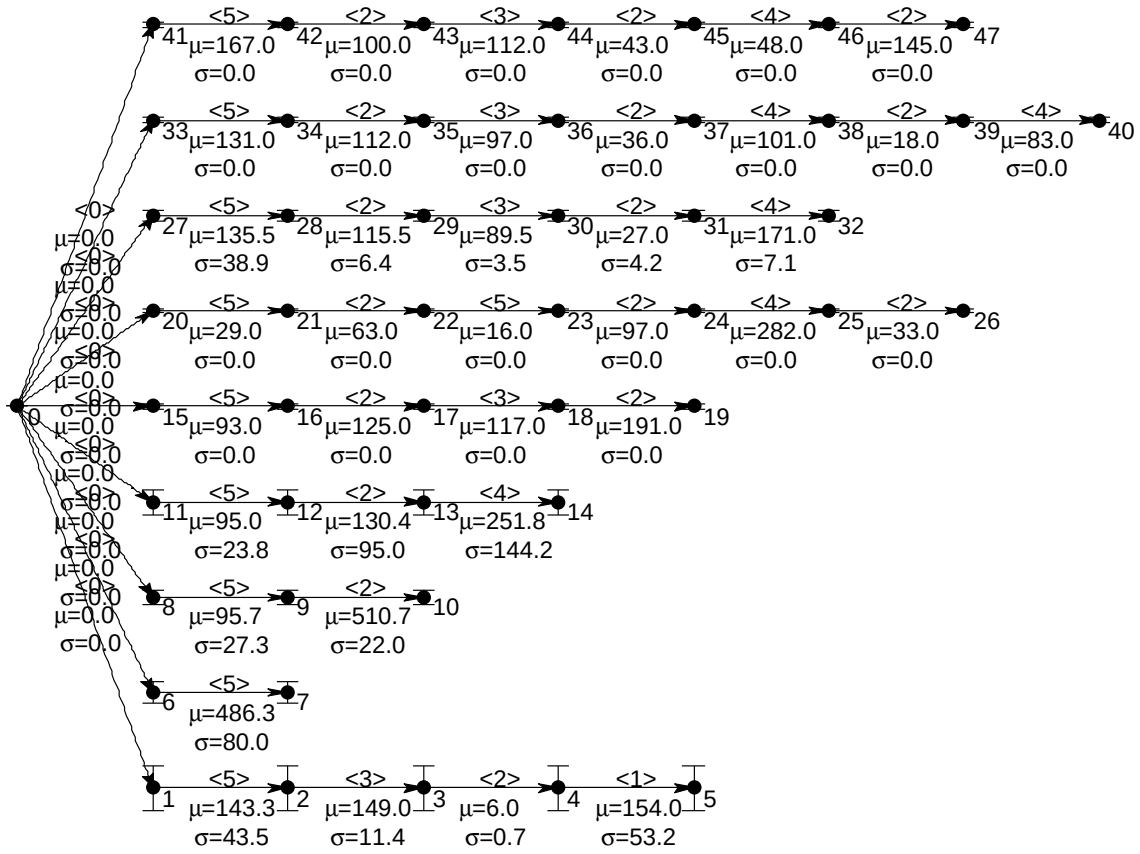


Figure 7-14: TPN learned from full golf data using 7 principal components, assuming 5 activities

We now perform recognition for the picking up ball test motion in Figure 7-11(e) using 7 principal components. The recognizer was able to correctly identify that the motion was picking up a ball instead of generically bending over. The recognized activity sequence with corresponding durations is given in Table 7.1. The recognition process takes on the order of 30 to 60 seconds for the entire observed data, and the incremental computation times are shown in Figure 7-15. It seems that the run times of the recognizer are not much affected by the number of principal components used in describing the data, but they are reduced from those in Figure 7-12 because of the

added differentiability with more principal components.

Activity	Duration
5	126
2	123
3	77
2	28
4	214

Table 7.1: Pick up ball motion recognized

We note that the plan learning and recognition algorithms do not require very many training data to work. In this case, we only used 27 training sequences, with 5 or fewer training sequences for some types of motions. Certainly, for more accurate results, more training sets are necessary, but even with very few training sequences, the plan learner and recognizer is guaranteed to produce a result.

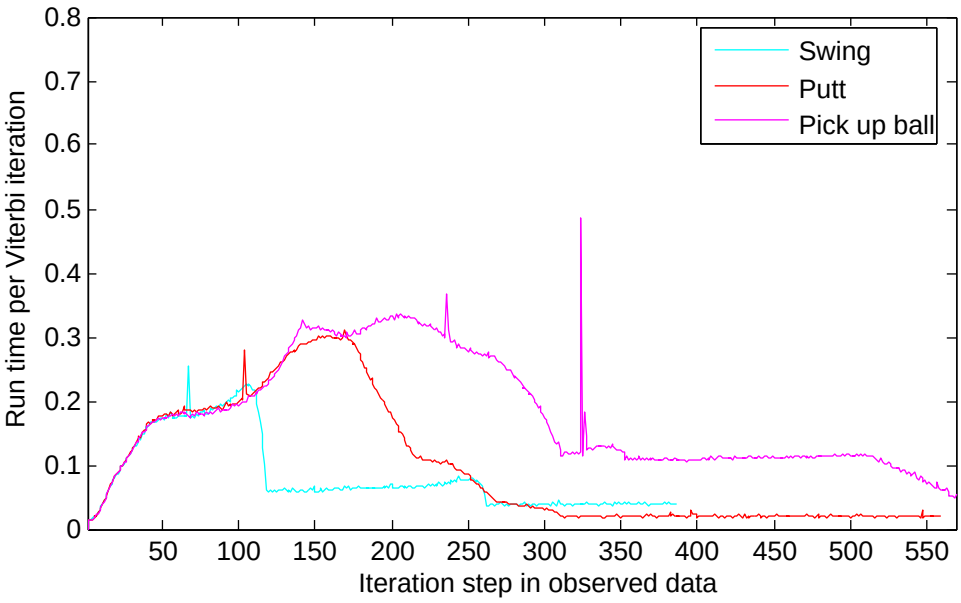


Figure 7-15: Run times for individual iterations during recognition using 7 principal components

To better see how the performance of plan learning varies with the number of activities k, we ran the learning algorithm on the golf data with different numbers of activities, ranging from 1 to 10, using 7 principal components. The results are summarized in Table 7.2.

k	# Lumping	# Splitting	Total Error
1	4	0	-8
2	6	5	-17
3	4	3	-11
4	3	11	-17
5	2	6	-10
6	3	9	-15
7	1	7	-9
8	3	8	-14
9	1	7	-9
10	1	8	-10

Table 7.2: Performance of learner as k changes

There are several things we notice from the results: First, there tends to be more error for small numbers of activities, because the learner has a harder time distinguishing different activity sequences. Secondly, the error levels off and even worsens around 7 activities because more activities do not encode much more new information, and tends to split the same motions into multiple plans. Finally, we observe an interesting pattern that performance with even number of activities tends to be worse than with odd number of activities. This could be due to the geometric asymmetries in the state space of the training data. Following our success criteria, the smallest number of activities that enables a successful plan learning is 5 for the set of golf training data.

We also ran our plan recognizer on the TPNs generated for each k using the pick up ball test data. In each case, the recognizer was able to identify a recognized activity sequence that corresponded to at least one of the pick up ball training data, thus satisfying our success criteria for recognition.

Chapter 8

Conclusions

This chapter presents several ideas for future expansion of this research. Section 8.1.1 discusses the possibility of representing parallel activities in a temporal plan network, while Section 8.1.2 presents an idea for automatically combining similar partial trajectories in a plan. Furthermore, we consider the prospect of learning the recognition resolution rather than provide it a priori in Section 8.1.3, and outline a possible incremental algorithm for recognition in Section 8.1.4. Finally, we conclude our discussions and summarize our contributions in Section 8.2.

8.1 Future Advancements

We provide some ideas for expanding the capabilities of the research in this thesis. Most of these ideas have not been proven to work, but are conceptually compelling.

8.1.1 Representing parallel activities in plan

Instead of learning and recognizing the activities of only one human subject, we might want to recognize the activities of two or more persons working together. In this case, the actions of each person are independent, and we would need a way to represent parallel activities in the learned temporal plan network. Figure 8-1 shows a generic TPN in the current representation used in this thesis. One possible way

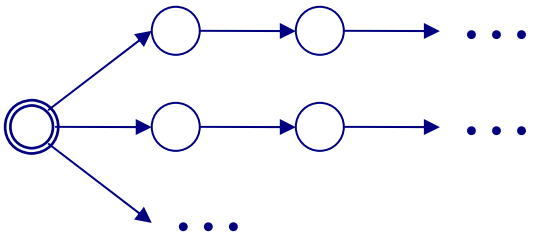


Figure 8-1: Current representation of a TPN without parallel activities

to represent two independent persons acting in parallel is to represent each possible combination of activities separately, as shown in Figure 8-2. This is a valid temporal

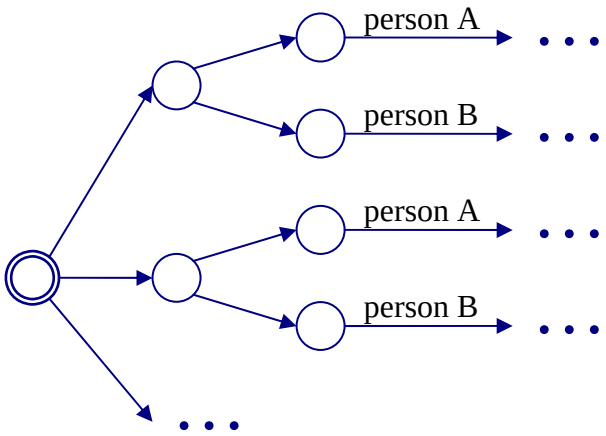


Figure 8-2: One way to represent parallel activities

plan network, but the representation can be inefficient in some cases. For example, if when person A performs one activity sequence, person B can perform a variety of possible sequences, then each of these combinations will be represented separately by the method in Figure 8-2.

A slightly better representation in this case would be to first branch on the possible activity sequences of person A, and then for each, branch on possible activity sequences of person B, as shown in Figure 8-3.

These methods can be extended to represent parallel activities for multiple persons. If there are n persons, and each person has s different activity sequences to choose from, then the worst case representation of the parallel TPN grows as $O(s^n)$. Thankfully, however, worst case branching does not usually occur in practice, as one

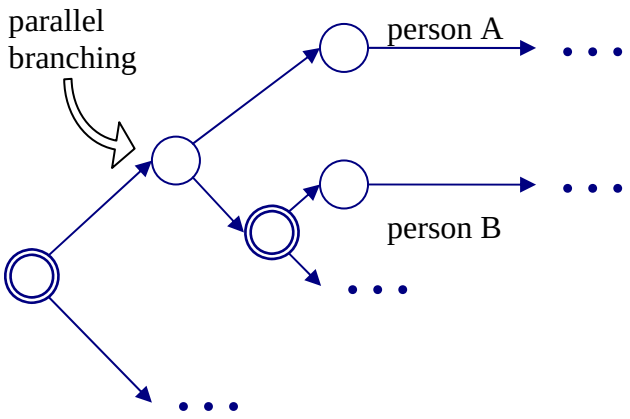


Figure 8-3: A more compact representation of parallel activities

person's activity choices will be limited by other people's choices.

8.1.2 Enabling intelligent combination of trajectories

Combining portions of trajectories that are similar is a tricky problem. Suppose we have two simlar activity sequences: one that is $\langle 1 \rangle, \langle 2 \rangle, \langle 3 \rangle$ and another that is $\langle 1 \rangle, \langle 2 \rangle, \langle 4 \rangle$. We can easily combine the first two activities and have the last activity branch from a choice event. However, what if one activity sequence is $\langle 1 \rangle, \langle 2 \rangle, \langle 3 \rangle$ and the other is $\langle 1 \rangle, \langle 2 \rangle, \langle 4 \rangle, \langle 2 \rangle, \langle 3 \rangle$? There is not one unique solution, as can be seen in Figure 8-4.

Figure 8-4: Two minimal representations of activity sequences $\langle 1 \rangle, \langle 2 \rangle, \langle 3 \rangle$ and $\langle 1 \rangle, \langle 2 \rangle, \langle 4 \rangle, \langle 2 \rangle, \langle 3 \rangle$

We propose that the issue of minimally representing a temporal plan network can be considered an application of learning a minimal deterministic finite automata (DFA). This kind of problem has been proven to be NP-hard [43] in the worst case, but tractable in the average case. Several researchers have been able to create algorithms that successfully learned DFAs from data of binary sequences [30]. We suspect that a similar method may be derived to minimally represent a temporal plan network.

8.1.3 Determining the recognition resolution

The plan learning algorithm used in this thesis requires two inputs: the set of all training data, and the recognition resolution. The recognition resolution is the number of activity clusters that will be identified during EM learning. Throughout this thesis, we assumed that we know something about the training data a priori that enables us to pick the correct recognition resolution. However, in many applications, we may not intuitively know how many activity clusters to choose. One possible way to determine the recognition resolution is by obtaining it from the training data using a v-fold cross-validation technique [52].

We can first divide the all-encompassing training data vector $\mathbf{X}$ into v random and roughly even sections. Next, we can perform unsupervised EM learning on $v-1$ sections, and classification testing on the remaining section of data. Cross-validation occurs by switching the section of data on which we perform learning versus testing. The classification errors are reflected by the average negative log-likelihood computed for the data points in the test section. The classification errors are aggregated, or averaged, over all v cross-validation cases. For some given recognition resolution k, we perform v-fold cross-validation to determine the corresponding classification error. Thus we can perform v-fold cross-validation over a variety of recognition resolutions to find the one that produces the lowest classification error.

We know that there is generally an optimal recognition resolution because small numbers of clusters may not explain the data well enough, producing large classification errors, and large numbers of clusters may overfit the data, also producing large classification errors. Thus v-fold cross-validation can identify the recognition resolution that gives best results.

8.1.4 Anytime algorithm for real-time incremental recognition

In practical recognition applications, new data arrives at a high frequency, and recognition must occur fast enough to be useful. First, we propose performing incremental

recognition by storing previous Viterbi messages in memory and updating only the most recent message. We also keep track of the previous most likely path, so that if the new back pointer points to the same previous back pointer, then there is no need to retrace all other back pointers–we just use the previously stored most likely path. This is a simpler version of the incremental Viterbi method used by Minka et al. in the image decoding application [37].

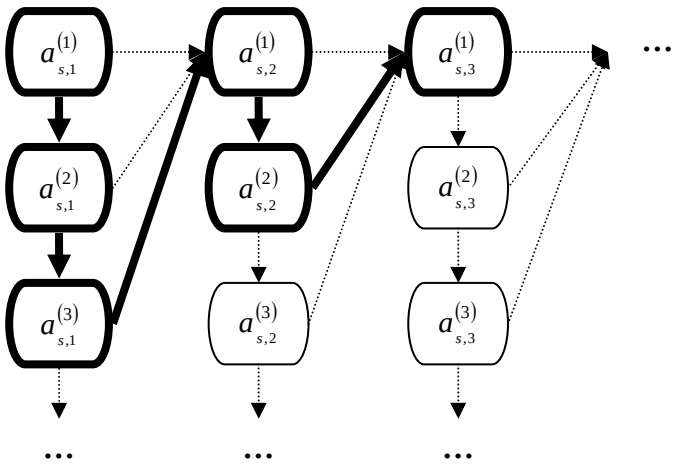


Figure 8-5: A most likely path in the HMM that we have cached

For example, in Figure 8-5, the highlighted path is the most likely path at the previous time step. Suppose at the next time step, the Viterbi message concludes that the most likely HMM state is $a^{(1)}_{s,4}$, which has a back pointer to $a^{(1)}_{s,3}$. Then we immediately know that the most likely path is now the highlighted path, plus the new state $a^{(1)}_{s,4}$. If the current HMM state has a different back pointer, such as $a^{(2)}_{s,3}$, then we will have to follow the back pointers to find the new most likely path. As iteration proceeds, however, the best path often becomes invariant as the recognizer becomes more certain that it is identifying the correct path, so this method of caching the previous most likely path at each iteration can greatly improve the recognizer's efficiency.

Even with the added efficiency, however, the recognizer still may not execute faster than the rate at which new data arrives. This calls for the need of an anytime algorithm. One method for an anytime recognizer is to perform iterative recognition

on the segment of data obtained during the last recognition session. While the current recognition session is executing, a new segment of data has arrived and is cached for use during the following recognition session.

8.2 Conclusions

In this thesis, we presented a plan learning and recognition capability to enable a computer or robotic agent to learn the temporal plan network of a human executing some set of tasks, and then perform recognition on newly observed motions. Our plan learner employed unsupervised learning on training data to determine the activity clusters describing the motions, and then created a temporal plan network based on the activity trajectories of the training data, such that all activities and durations were represented probabilistically. Following this process, our plan recognizer encodes the temporal plan network as a hidden Markov model to determine the most likely activity sequence based on the observed data. It then refers back to the temporal plan network to obtain the predicted activity sequence containing most likely future activities.

Past research in recognition have often focused on identifying gestures, where data were pre-segmented, so different forms of supervised learning techniques could be performed [26]. In contrast, this thesis employed unsupervised learning to automatically detect the activity segmentation in continuous state data. Futhermore, previous research in plan recognition have often been limited to discrete activities [14, 46, 9], and thus were unsuitable for recognizing physical motions as in our applications. Additionally, some past plan recognition algorithms assumed the existance of plan recipes [5, 32], forcing the user to manually create a plan recipe before using the algorithms. Our research was designed for recognition of continuous physical motions, and instead of manually creating a plan recipe, our algorithm automatically learned a plan network from training data.

This thesis presented several innovations: First, we introduced a modified representation of temporal plan networks that incorporates probabilistic information into

the state space and temporal representations. Second, we learned plans from actual data, such that the notion of an activity is not arbitrarily or manually defined, but is determined by the characteristics of the data. Third, we developed a recognition algorithm that can perform recognition continuously by making probabilistic updates. Finally, our recognizer could not only identify previously executed activities, but could also predict future activities based on the plan network.

We demonstrated the capabilities of our algorithms on motion capture data using a simple dancing example, followed by a more complex golfing scenario. Our results showed that the plan learning algorithm was able to generate reasonable temporal plan networks, depending on the dimensions of the training data and the recognition resolution used. The plan recognition algorithm was also successful in recognizing the correct activity sequences in the temporal plan network corresponding to the observed test data.

We have discussed some possible future areas of expansion, and we feel that plan learning and recognition is a valuable field of research, as it provides a necessary step toward making more intelligent, interactive, and useful robots.

Appendix A

Splines

As discussed in Section 4.2.2, when we have a scarcity of data, we employ a method of essentially duplicating the existing data, and adding some amount of noise to the newly created data. In order to ensure that the resulting data is smooth, we first evently sample some relatively small number of data points from the original data, add a certain amount of Gaussian noise to the data points, and interpolate a new data sequence by applying a cubic spline function. We label the number of data points sampled as q, and the sampled data as *control points*. This section will discuss the details of the cubic spline function, closely following the explanation provided by Weisstein [55].

Given q control points, a cubic spline is constructed from $q-2$ piece-wise cubic polynomials passing though all control points. The second derivative of the polynomials at the end points are set to zero to complete the necessary system of equations. We now present the process of creating one-dimensional splines, which can be extrapolated to multiple dimensions.

Suppose our set of control points are $(y_1, y_2, \ldots, y_q)$, and that the i^{th} piece of the spline is represented by

$$Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3, \tag{A.1}$$

where $i = 1, 2, \ldots, q-1$ and $0 \leq t \leq 1$ so that each piece of spline's end points occur

at $t = 0$ and $t = 1$, or in other words,

$$Y_i(0) = y_i = a_i \tag{A.2}$$

$$Y_i(1) = y_{i+1} = a_i + b_i + c_i + d_i. \tag{A.3}$$

Taking the first derivative at each spline's end points produces

$$Y_i'(0) = \frac{dy_i}{dt} = b_i \tag{A.4}$$

$$Y_i'(1) = \frac{dy_{i+1}}{dt} = b_i + 2c_i + 3d_i. \tag{A.5}$$

We can now solve Equations A.3 – A.5 for the coefficients, which gives

$$a_i = y_i \tag{A.6}$$

$$b_i = \frac{dy_i}{dt} \tag{A.7}$$

$$c_i = 3(y_{i+1} - y_i) - 2\frac{dy_i}{dt} - \frac{dy_{i+1}}{dt} \tag{A.8}$$

$$d_i = 2(y_i - y_{i+1}) + \frac{dy_i}{dt} + \frac{dy_{i+1}}{dt}. \tag{A.9}$$

So now we need to solve for the derivatives.

The internal boundary conditions require that the boundary points, first derivative, and second derivative are all compatible between two pieces of splines Y_i and Y_{i+1}. Specifically, the internal boundary conditions are

$$Y_i(0) = y_i \tag{A.10}$$

$$Y_i(1) = Y_{i+1}(0) = y_{i+1} \tag{A.11}$$

$$Y_i'(1) = Y_{i+1}'(0) \tag{A.12}$$

$$Y_i''(1) = Y_{i+1}''(0). \tag{A.13}$$

The external boundary conditions ensure that the end points of the spline satisfy

$$Y_1(0) = y_1 \tag{A.14}$$

$$Y_{q-1}(1) = y_q. \tag{A.15}$$

Equations A.11 – A.15 provide $4(q-1)+2 = 4q-2$ relations for the $4q$ unknowns. Thus we include the two extra boundary conditions requiring the second derivative at the end points to be zero, or

$$Y_1''(0) = 0 \tag{A.16}$$

$$Y_{q-1}''(1) = 0. \tag{A.17}$$

We can now express all these equations in a tridiagonal system

$$\begin{bmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 2 \end{bmatrix} \cdot \frac{d}{dt} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{q-1} \\ y_q \end{bmatrix} = \begin{bmatrix} 3(y_2 - y_1) \\ 3(y_3 - y_1) \\ 3(y_4 - y_2) \\ 3(y_5 - y_3) \\ \vdots \\ 3(y_q - y_{q-2}) \\ 3(y_q - y_{q-1}) \end{bmatrix}. \tag{A.18}$$

Solving the tridiagonal system gives the derivatives at each piece of spline's end points, which can be used to find the spline coefficients using Equations A.7 – A.9. This same method can be used to generate splines in multiple dimensions.

Appendix B

Encoding a TPN into XML

Our plan learning algorithm encodes temporal plan networks in XML format. The following is an example excerpt of the XML code that discribes Figure 4-6 in Section 4.5.

```
<root>
   <choices>
      <choice>
         <start>
            0
         </start>
      </choice>
   </choices>
   <tpn>
      <arcs>
       ...
      </arcs>
      <arcs>
         <start>
            1
         </start>
         <end>
            2
         </end>
         <times_k>
            19.25
         </times_k>
         <times_theta>
```

```
            0.82
         </times_theta>
         <activity>
            <name>
               3
            </name>
            <mu>
               [1.0039,0.9070]
            </mu>
            <sigma>
               [0.0820,0.0562;0.0562,0.0720]
            </sigma>
         </activity>
         <prob>
            0.2
         </prob>
      </arcs>
      ...
   </tpn>
</root>
```

Bibliography

[1] J. K. Aggarwal and Q. Cai. Human motion analysis: A review. *Computer Vision and Image Understanding: CVIU*, 73(3):428–440, 1999.

[2] James F. Allen. Planning as temporal reasoning. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *KR'91: Principles of Knowledge Representation and Reasoning*, pages 3–14. Morgan Kaufmann, San Mateo, CA, 1991.

[3] James F. Allen. Time and time again: the many ways to represent time. *International Journal of Intelligent Systems*, 6:341–355, 1991.

[4] R.O. Ambrose, H. Aldridge, R.S. Askew, R.R. Burridge, W. Bluethmann, M. Diftler, C. Lovchik, D. Magruder, and F. Rehnmark. Robonaut: NASA's space humanoid. *IEEE Intelligent Systems and Their Applications*, 15(4):57–63, August 2000.

[5] Dorit Avrahami-Zilberbrand and Gal A. Kaminka. Fast and complete symbolic plan recognition. In *International Joint Conference on Artificial Intelligence*, Scotland, Edinburgh, 2005.

[6] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1215–1222, Portland, Oregon, USA, 1996. AAAI Press / The MIT Press.

[7] Jernej Barbič, Alla Safonova, Jia-Yu Pan, Christos Faloutos, Jessica Hodgins, and Nancy Pollard. Segmenting motion capture data into distinct behaviors. *ACM International Conference Proceeding Series*, 62:185–194, 2004.

[8] Mathias Bauer. Integrating probabilistic reasoning into plan recognition. In *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI '94)*, pages 620–624, 1994.

[9] Mathias Bauer. A Dempster-Shafer approach to modeling agent preferences for plan recognition. *User Modeling and User-Adapted Interaction*, 5(3-4):317–348, 1996.

[10] Jeff Bilmes. A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and Hidden Markov Models. Technical Report ICSI-TR-97-021, University of Berkeley, 1997.

[11] Aaron F. Bobick and Andrew D. Wilson. A state-based technique for the summarization and recognition of gesture. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 382–388, Cambridge, MA, June 1995.

[12] Roger Boyle. Viterbi algorithm. From http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/viterbi_algorithm/s1_pg1.html, 2007.

[13] Richard Cangelosi and Alain Goriely. Component retention in principal component analysis with application to cDNA microarray data. *Biology Direct*, 2(1):2, January 2007.

[14] Eugene Charniak and Robert P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, 1993.

[15] Carroll Croarkin and Paul Tobias, editors. *NIST/SEMATECH e-Handbook of Statistical Methods*, chapter 1.3.6.6.11. Gamma Distribution. National Institute of Standards and Technology, July 2006.

[16] Philip J. Davis. Gamma function and related functions. In Milton Abramowitz and Irene A. Stegun, editors, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, chapter 6. Superintendent of Documents, U.S. Government Printing Office, Washington, DC, 1972.

[17] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[18] Arthur Dempster, Nan Laird, and Donald Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[19] Robert Effinger. Optimal temporal planning at reactive time scales via dynamic backtracking branch and bound. S.M. Thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, September 2006.

[20] Maria Fox, Malik Ghallab, Guillaume Infantes, and Derek Long. Robot introspection through learned Hidden Markov Models. *Artificial Intelligence*, 70(2):59–113, February 2006.

[21] D. M. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding: CVIU*, 73(1):82–98, 1999.

[22] Andreas Hofmann. *Robust Execution of Bipedal Walking Tasks from Biomechanical Principles*. Ph.D. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, January 2006.

[23] Robert Hogg, Joseph McKean, and Allen Craig. *Introduction to Mathematical Statistics*, pages 359–364. Pearson Prentice Hall, Upper Saddle River, New Jersey, 2005.

[24] J. Edward Jackson. *A User's Guide to Principal Components*. John Wiley & Sons, New York, 1991.

[25] Ian T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, second edition, 2002.

[26] Mohammed Waleed Kadous. A general architecture for supervised classification of multivariate time series. Technical Report UNSW-CSE-TR-9806, University of New South Wales, Department of Artificial Intelligence, School of Computer Science & Engineering, September 1997.

[27] Henry Kautz. A formal theory of plan recognition and its implementation. In J. Allen, H. Kautz, R. Pelavin, and J. Tenenberg, editors, *Reasoning about Plans*, pages 69–125. Morgan Kaufman, San Mateo, CA, 1991.

[28] Henry A. Kautz and James F. Allen. Generalized plan recognition. In *Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 32–37, Menlo Park, CA, August 1986. AIAA Press.

[29] P. Kim, B. C. Williams, , and M. Abramson. Executing reactive, model-based programs through graph based temporal planning. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 487–493. Lawrence Erlbaum Associates LTD, 2001.

[30] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. *Lecture Notes in Computer Science*, 1433:1–12, 1998.

[31] Thomas Léauté. Coordinating agile systems through the model-based execution of temporal plans. S.M. Thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, August 2005.

[32] Neal Lesh, Charles Rich, and Candace Sidner. Using plan recognition in human-computer collaboration. In *Seventh International Conference on User Modeling*, pages 23–32, 1999.

[33] Lin Liao, Donald J. Patterson, Dieter Fox, and Henry Kautz. Learning and inferring transportation routines. *Artificial Intelligence*, 171(5-6):311–331, April 2007.

[34] Jeroen Lichtenauer, E. A. Hendriks, and M. J. T. Reinders. 3D versus 2D pose information for recognition of NGT signs. In *27th Symposium on Information Theory*, Benelux, 2006.

[35] Hugo Liu and Push Singh. Commonsense reasoning in and over natural language. *Lecture Notes in Computer Science*, 3215:293–306, October 2004.

[36] R. Bowen Loftin and Patrick J. Kenney. Training the Hubble Space Telescope flight team. *Computer Graphics and Applications, IEEE*, 15(5):31–37, September 1995.

[37] Thomas P. Minka, Dan S. Bloomberg, and Kris Popat. Document image decoding using iterated complete path search. *Document Recognition VIII*, January 2001.

[38] T. Moeslund, A. Hilton, and V. Krueger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3):90–127, 2006.

[39] Paul Morris and Nicola Muscettola. Execution of temporal plans with uncertainty. In *Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, pages 491–496. AAAI Press/The MIT Press, 2000.

[40] Sarah Osentoski, Victoria Manfredi, and Sridhar Mahadevan. Learning hierarchical models of activity. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004.

[41] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572, 1901.

[42] Joelle Pineau, Michael Montemerlo, Martha Pollack, Nicholas Roy, and Sebastian Thrun. Towards robotic assistants in nursing homes: challenges and results. *Robotics and Autonomous Systems*, 42(3-4):271–281, 2003.

[43] Leonard Pitt and Manfred K. Warmuth. The minimum DFA consistency problem cannot be approximated within any polynomial. In *Twenty-First Annual ACM Symposium on Theory of Computing*, pages 421–432, Seattle, WA, May 1989.

[44] Nancy Pollard, Jessica Hodgins, Marcia Riley, and Christopher Atkeson. Adapting human motion for the control of a humanoid robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1390–1397, Washington, D.C., May 2002.

[45] Rudolph W. Preisendorfer. *Principal Component Analysis in Meteorology and Oceanography*. Elsevier Science Publishing Company, Amsterdam, December 1988.

[46] David V. Pynadath and Michael P. Wellman. Accounting for context in plan recognition, with application to traffic monitoring. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 472–481, San Francisco, 1995. Morgan Kaufmann.

[47] Lawrence R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *IEEE*, 77(2), February 1989.

[48] Stuart J. Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*, section 20.3, pages 724–727. Pearson Education, Inc, Upper Saddle River, New Jersey, second edition, 2003.

[49] Stuart J. Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*, section 15, pages 537–551. Pearson Education, Inc, Upper Saddle River, New Jersey, second edition, 2003.

[50] Leonid Sigal and Michael J. Black. Predicting 3D people from 2D pictures. In *International Conference on Articulated Motion and Deformable Objects*, Andratx, Mallorca, Spain, July 2006. Springer LNCS 4069.

[51] Bongkee Sin and Jin H. Kim. Nonstationary Hidden Markov Model. *Signal Processing*, 46(1):31–46, September 1995.

[52] Inc StatSoft. *Electronic Statistics Textbook*, chapter Cluster Analysis. StatSoft, Tulsa, OK, 2007.

[53] Michael E. Tipping and Christopher M. Bishop. Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443–482, 1999.

[54] Seema Vyas and Lilani Kumaranayake. Constructing socio-economic status indices: how to use principal components analysis. *Health Policy and Planning*, 21(6):459–468, October 2006.

[55] Eric W. Weisstein. Cubic spline. From *MathWorld*—A Wolfram Web Resource: http://mathworld.wolfram.com/CubicSpline.html.

[56] Eric W. Weisstein. Run-length encoding. From *MathWorld*—A Wolfram Web Resource: http://mathworld.wolfram.com/Run-LengthEncoding.html.

[57] Brian Williams, Phil Kim, Michael Hofbaur, Jon How, Jon Kennell, Jason Loy, Robert Ragno, John Stedl, and Aisha Walcott. Model-based reactive programming of cooperative vehicles for Mars exploration. *Int. Symp. on Artificial Intelligence, Robotics and Automation in Space (ISAIRAS-01)*, 2001.

[58] Andrew D. Wilson and Aaron F. Bobick. Using Hidden Markov Models to model and recognize gesture under variation. *International Journal on Pattern Recognition and Artificial Intelligence*, Special Issue on Hidden Markov Models in Computer Vision, 2000.

[59] C. F. Jeff Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

[60] Wenyi Zhao, Arvindh Krishnaswamy, Rama Chellappa, Daniel Swets, and John Weng. Discriminant analysis of principal components for face recognition. *Face Recognition: From Theory to Applications*, pages 73–85, 1998.