



UNIVERSITY of OULU
OULUN YLIOPISTO

FILLER MODELS FOR ACCELEROMETER BASED
CONTINUOUS GESTURE RECOGNITION

Mikko Kauppila

Master's Thesis
May 2008

DEPARTMENT OF MATHEMATICAL SCIENCES
UNIVERSITY OF OULU
FINLAND

Department Department of Mathematical Sciences		Author Mikko Kauppila	
Thesis name Filler Models for Accelerometer Based Continuous Gesture Recognition			
Subject Mathematics	Level of studies Master's thesis	Date May 2008	Number of pages 82
Abstract <p>Interest in the recognition of gestures from accelerometer data has been on the rise due to advances in manufacturing technology. Also, accelerometers are becoming highly ubiquitous due to their inclusion in mobile phones. Such gesture recognition has lots of potential applications in human-computer interaction (HCI). In the recent years, the hidden Markov model (HMM) has been applied successfully to the problem of accelerometer based isolated gesture recognition, where a batch of acceleration data is classified into pre-taught "keyword" gesture classes. The HMM is a popular statistical model for modeling spatiotemporal variability, and has had past success in other fields as well, notably speech recognition.</p> <p>In this thesis work we present a review of existing work on HMMs and gesture recognition. In addition, we study the application of HMMs to the problem of accelerometer based continuous gesture recognition, where a set of keyword gestures is spotted on the fly from accelerometer data. We adopt the so called filler model, which simultaneously models the keyword gestures and the intermittent non-gestural regions. To our knowledge, this is the first time such model is being applied to accelerometer based gesture recognition.</p> <p>We implemented a software based continuous recognizer running on both desktops and mobile phones. Its recognition performance was measured in the single user case, and its efficiency on mobile phones was studied rudimentarily. We obtained error rates of 4.7%, 2.0% and 3.3% for classification, false negatives and false positives, respectively. The biggest disadvantage of the model is the sensitivity of the false positive rate to adjustable recognition parameters. We identify venues for further research, notably the incorporation of more expressive filler models.</p>			
Keywords HMM, filler model, garbage model, accelerometer, pattern recognition, gesture recognition			

Laitos Matemaattisten tieteiden laitos		Tekijä Mikko Kauppila	
Työn nimi Täytemallien käyttö kiihtyvyyssanturipohjaisessa jatkuvassa eleentunnistuksessa			
Oppiaine Matematiikka	Työn laji Pro gradu -tutkielma	Aika Toukokuu 2008	Sivumäärä 82
Tiivistelmä <p>Kiinnostus kiihtyvyyssantureiden käyttöön eleentunnistuksessa on kasvanut kehittyneen valmistusteknologian myötä. Lisäksi kiihtyvyyssanturien sisällyttäminen matkapuhelimiin on kasvattanut niiden yleisyyttä. Tällaisella eleentunnistuksella on paljon potentiaalisia sovelluksia ihmisen ja koneen vuorovaikutuksessa. Viime vuosina Markovin piilomalleja on sovellettu menestyksekkäästi kiihtyvyyssanturipohjaiseen eristetyn eleentunnistukseen, jossa eristetty kiihtyvyysssekvenssi luokitellaan johonkin opetettuun eleluokkaan kuuluvaksi. Markovin piilomalli on suosittu tilastollinen malli spatiotemporaalisen vaihtelun mallintamiseen, ja sitä on sovellettu menestyksekkäästi myös muilla aloilla, eritoten puheentunnistuksessa.</p> <p>Tässä pro gradu -tutkielmassa perehdymme olemassaoleviin Markovin piilomalleja ja eleentunnistusta koskeviin tutkimustuloksiin. Lisäksi tutkimme Markovin piilomallien soveltamista jatkuvaan eleentunnistukseen, jossa opetettuja eleitä tunnistetaan lennossa jatkuvasta kiihtyvyysssekvenssistä. Sovellamme niin kutsuttua täytemallia (filler model), joka mallintaa tunnistettavia eleitä sekä niiden välissä olevia merkityksettömiä liikkeitä samanaikaisesti. Uskoaksemme tämä on ensimmäinen kerta, kun tällaista mallia sovelletaan kiihtyvyyssanturipohjaiseen eleentunnistukseen.</p> <p>Tutkimuksessa toteutettiin ohjelmistopohjainen jatkuva tunnistin, joka toimii sekä pöytäkoneilla että matkapuhelimissa. Järjestelmän tunnistusarkkuus mitattiin yhden käyttäjän tapauksessa, sekä sen tehokkuus matkapuhelinkäyttöä ajatellen selvitettiin alustavasti. Saavuttamamme virhemäärät eleiden luokittelulle, väärille hylkäyksille ja väärille hälytyksille olivat 4,7%, 2,0% ja 3,3%, vastaavasti. Mallin suurin heikkous on väärin hälytysten määrän herkkyys säädettäville tunnistusparametreille. Jatkotutkimusmahdollisuuksista huomioimme muun muassa ilmaisuvoimaisempien täytemallien soveltamisen.</p>			
Avainsanat Markovin piilomalli, täytemalli, kiihtyvyyssanturi, hahmontunnistus, eleentunnistus			

Table of Contents

Table of Contents	4
Preface	8
1 Introduction	9
1.1 Notation	10
2 Background and Related Work	11
3 The Hidden Markov Model	15
3.1 Markov Chains	15
3.1.1 Parameter Estimation	17
3.2 From Markov Chains to HMMs	17
3.3 Evaluation of Observation Sequence Densities	18
3.4 Finding the Most Likely State Sequence	20
3.5 Estimating Model Parameters	21
3.5.1 Additional Notation	21
3.5.2 The Expectation-Maximization Algorithm	23
3.5.3 The Baum–Welch Algorithm	24
3.5.4 Obtaining Initial Estimates	27
3.6 Numerical Issues	27
3.7 Special Topologies	28
3.8 Discussion	29
4 Gesture Recognition	30
4.1 Accelerometer Basics	30

4.2	Sensor Model	31
4.3	Featurization	33
4.3.1	Isolated Versus Continuous Featurization	34
4.3.2	Comparison to Speech Recognition	34
4.3.3	Estimating Gravitational Resistance	35
4.3.4	Tilt Compensation	36
4.3.5	Amplitude Compensation	38
4.3.6	Tempo Compensation	39
4.3.7	Magnitudal Features	39
4.3.8	Primitivization	40
4.4	Isolated Recognition	40
4.4.1	Parameter Estimation	40
4.4.2	Recognition	41
4.4.3	Gesture Spotting	42
4.5	Continuous Recognition	43
4.5.1	Parameter Estimation	44
4.5.2	Recognition	46
4.6	Extensions and Refinements	47
4.6.1	Multiple Sensors	47
4.6.2	Controlling the Subset of Recognized Key Gestures	47
5	Implementation	49
5.1	System Architecture	49
5.2	Accelerometers	50

5.2.1	SHAKE SK6	50
5.2.2	Nokia Mobile Phones	50
5.2.3	Synchronizing Accelerometers	50
5.3	Recorder	51
5.3.1	Detecting Outliers	52
5.4	Estimator	52
5.5	Recognizer	53
5.6	Cross Validator	53
6	Results	54
6.1	Methodology	54
6.1.1	Cross Validation	55
6.1.2	Extension to Continuous Case	55
6.2	Single-User Experiments	57
6.2.1	Classification Performance	57
6.2.2	Spotting Performance	59
6.3	Effect of Parameter Variations	61
6.3.1	Tilt Compensation	61
6.3.2	Magnitudal Features	61
6.3.3	Covariance Constraints	62
6.3.4	HMM State Count	62
6.3.5	Garbage Weight Percentage	62
6.3.6	Garbage Mixture Count	63
6.3.7	Garbage Self-Transition Probability	63
6.3.8	Discussion	63

6.4	Runtime Performance	64
6.4.1	Real-Time Ratio	64
6.4.2	Time Distribution	65
6.4.3	Power Consumption	66
7	Conclusion and Perspectives	68
	Appendix A Common Observation Distribution Families	69
A.1	Univariate Normal Distribution	70
A.2	Multivariate Normal Distribution	70
A.3	Mixture Distribution	72
A.3.1	Parameter Estimation	72
A.4	Categorical Distribution	74
A.4.1	Vector Quantization	74
	Appendix B Logarithm Transformations	76
B.1	The Exponential Function	76
B.2	Products and Divisions	77
B.3	Summation	77
B.3.1	Weighted Summation	78
	References	79

Preface

This thesis work was carried out in the Intelligent Systems Group, Department of Electrical and Information Engineering, University of Oulu. The research started in May 2007, and the thesis was finalized in May 2008.

The work was supported by National Technology Agency of Finland, the latter half of it conducted as a part of the UbiLife project. I would like to express my gratitude to my thesis advisor, Doc. Erkki Laitinen from the Department of Mathematical Sciences. I would also like to thank Prof. Jukka Riekkö and Dr. Susanna Pirttikangas for providing me with the interesting research topic and also the facilities for carrying out the research. My coworker Tuomas Inkeroinen deserves a special mention for the application of the recognizer described in this thesis to the problem of controlling mobile phone functions. The feedback generated by his real-world application was invaluable to the advancement of my own research.

Oulu, May 7, 2008

Mikko Kauppila

1 Introduction

In everyday interaction, people perform physical *gestures*, which can be loosely defined as coordinated movements carrying semantic meanings, for example, waving one’s hand.

With the advent of computing technology, machines can now recognize gestures using a variety of physical sensors and computational algorithms. The most popular sensors for gesture recognition up to date include cameras and accelerometers, both approaches carrying their own advantages and disadvantages. As a computational problem, gesture recognition is highly similar to the much more intensely studied problem of speech recognition, and the grounding algorithms are in fact largely adopted from the speech recognition community.

The intention of this thesis work was to study hidden Markov model (HMM) based gesture recognition algorithms with acceleration sensors, concentrating on gestures performed by hands. In the HMM, each time moment is associated with a visible observation (e.g., the acceleration of user’s hand) and a hidden state. By associating the hidden states with physically meaningful semantics, useful inference about the underlying physical phenomena can be drawn when the hidden states are recovered (in a maximum likelihood sense).

In classical, isolated HMM-based gesture recognition, each keyword gesture type (e.g., a clockwise hand movement) is associated with a single HMM, and isolated acceleration sequences are classified into gesture types via density estimation. Assuming a suitably constrained transition model, each hidden state in an HMM effectively represents a time interval in the corresponding keyword gesture. The HMMs are estimated from training data, which consists of repetitions of exemplar gestures by one or more users.

We adopted the filler model (also known as the garbage model) approach from speech recognition literature that is especially suited to continuous keyword spotting, that is, finding known gestures from a continuous acceleration sequence.¹ In the filler model, the individual gesture HMMs are connected via additional garbage states. These garbage states model intermittent non-gestural regions in the acceleration sequence. Transitions from gesture states back to garbage states are registered as recognized gestures.

¹For the mathematically inclined, it should be noted that by continuous sequences we refer to sequences with an unbounded index set.

The rest of this thesis is organized as follows. In section two, we present the background and related work, its relation to this thesis, and the contribution of the thesis. In section three, we cover the theory and implementation issues of the HMM, our underlying time series model, and section four explains how the HMM is utilized to build algorithms for gesture recognition. Sections five and six present our recognizer implementation and results, respectively. Finally, section seven wraps up the thesis with a conclusion and discussion on future perspectives.

1.1 Notation

Random variables are capitalized (e.g., X) whereas their realizations are non-capitalized, e.g., x . Probabilities of discrete events are denoted as $\Pr[X = x]$. Density functions are generally denoted by f , subscripted with the random variables in question, e.g., $f_X(x)$. If a function is parametrized, the parameters are listed after a semicolon to avoid confusion, e.g., $f_X(x; \theta)$.

Matrices are capitalized (e.g., A) whereas their elements are non-capitalized, e.g., $A = (a_{ij})$. Vectors (tuples) and their elements are generally non-capitalized, e.g., $x = (x_1, \dots, x_T)$.²

In asymptotic analysis, we use the standard symbol Θ to denote asymptotic tight bounds. More precisely, given two functions f and g , we say that f is $\Theta(g)$ if and only if there exist constants c_1 , c_2 and n_0 such that

$$c_1g(n) \leq f(n) \leq c_2g(n), \quad \forall n \geq n_0.$$

²To keep our discussion general, we often can't tell whether elements of tuples are *themselves* tuples. Therefore we decided not to use any special notation for tuples, e.g., boldface letters.

2 Background and Related Work

Pattern recognition is the subset of machine learning that deals with automated spotting and classification of patterns, such as handwriting, fingerprints, retinal scans, speech, and movements (gestures). Exact pattern recognition problems, such as finding occurrences of words from text corpora, are of course solvable with simple matching algorithms. However, the quintessential problem in real-world pattern recognition is how to deal with the underlying *variability* in the patterns. Generally speaking, we expect that recognition becomes harder as the amount of allowed variability grows; for example, it can be much harder to build a speech recognizer that performs well for multiple users as opposed to just a single user.

Recognizing speech is one of the oldest branches in pattern recognition, with many potential applications, for example in person identification and speech-to-text conversion. Speech commands can also be used as an additional modality (i.e., communication pathway) in user interfaces, for example to operate some aspects of aircrafts. A more detailed description of the history of speech recognition research is included in the classic speech recognition book of Rabiner and Juang [28, pp. 6-9].

Speech and gesturing are both examples of *time-related* phenomena. Generally speaking, time-related phenomena possess both spatial and temporal variability, together known as *spatiotemporal* variability. Spatial variability includes aspects such as the pitch of voice or the direction of hand movements, whereas temporal variability includes differences in duration. Notice that spatial and temporal variability may not be independent; for example, performing a hand gesture very fast increases the amplitude of velocity and acceleration, but shortens the duration.

When comparing an input sequence against a known template sequence, time warping is a general way of handling time variability, where a suitably constrained change of variable $t \mapsto \phi(t)$ is constructed for the discretized time $t \in \mathbb{N}$ of the input sequence [28, pp. 200-204]. The mapping ϕ is chosen so that some distortion (distance) between the input and template sequences, possibly penalized by the quality of ϕ , is minimized. The most popular form of time warping is dynamic time warping (DTW), which employs the computationally efficient technique of dynamic programming [28, pp. 221-226]. After this time normalization step, any distortion can serve as a metric of similarity between the sequences.

However, these days DTW combined with distortion metrics is best seen as an *ad hoc* solution — there are no obvious choices as to how ϕ should be constrained and penalized, and how the distortions should be constructed. As a result, researchers have turned to statistical methods, which present attractive and formal solution families. Specifically, the hidden Markov model (HMM) provides an elegant solution to handling spatiotemporal variability in a unified fashion, combining an implicit DTW penalized by transition probabilities with a set of observation distributions for modeling the spatial distortion. The HMM has been employed successfully in both speech recognition [9, 27, 35, 16] and gesture recognition [31, 23, 32, 18, 22, 12, 21, 26, 36, 15].

When recognizing real-life phenomena such as speech or gesturing, an essential consideration is what kind of *sensors* should be employed for capturing the analog phenomena. For speech the obvious answer is a microphone, which captures its surrounding sound waves. For gesturing, a variety of sensors are available.

Starner *et al.* [31, 32] employed a video camera for recognizing American sign language (essentially hand gestures). Video camera based recognition was also done by Lee and Kim [18] for hand gestures and Yang *et al.* [36] for full-body gestures. However, video based systems require *tracking* the moving body parts, which is an active research problem itself³, although tracking can be simplified by having the user wear distinctive clothing [31]. Furthermore, the user must be in the vicinity of the camera and the gesture cannot be occluded; on the other hand, these constraints can be relaxed by having the user wear the camera [32].

In contrast to algorithmically tracking movement from video signals, inertial sensors, such as gyroscopes and accelerometers, can be used to measure the movement of gestures directly. Early work of Nam and Wohn [23] presents a hand gesture recognizer using a position-tracking sensor. Such position-tracking sensors are typically built of a combination of accelerometers and gyroscopes⁴ — however, a cheaper alternative is to use only accelerometers. As explained by Choi *et al.* [4], recent advantages in microelectromechanical systems (MEMS) have reduced the size and cost of accelerometers considerably, while gyroscopes still remain large and relatively expensive. A disadvantage of using only accelerometers is that interaction with the user’s hand

³For a recent survey of hand tracking algorithms, see Mahmoudi and Parviz [19].

⁴Gyroscope-free tracking is also possible, but requires a minimum of six accelerometers and results in an increased error [33].

affects the sensor readings in an ambiguous manner, although this can be somewhat alleviated with compensation techniques [26, 15].

Given the lowered size and cost of accelerometers, mobile phone vendors have recently started including accelerometers to their phone models. To our knowledge, at least 5500 Sport and N95 models from Nokia, in addition to SCH-S310 and E760 models from Samsung, include a triple-axis accelerometer. In fact, the Samsung models can already be controlled with gesture commands [4, 3].

The Samsung recognizer is based on a conditional Gaussian model, which belongs to the same family of acyclic directed graphical models as the HMM, combined with support vector machine (SVM) based classification to distinguish confusing pairs of gestures. Choi *et al.* [4] notes that feedback for gesture interaction on mobile phones should be studied more carefully: trivial options include sound and vibrotactile feedback, but also an elaborate visualization based method that renders gesture trajectories has been proposed by Kallio *et al.* [13]. HMM-based gesture recognition on Nokia phones has been studied by Pylvänäinen [26], who studied the effect of sampling and quantization rate on recognition performance, and also presented a method for compensating against varying hand orientations; our work is probably closest to his, but we extend the algorithms to continuous recognition (described shortly).

Using the inbuilt accelerometers in mobile devices opens up interesting possibilities for human-computer interaction (HCI) since the device is effectively turned into a physical user interface. Mäntyjärvi *et al.* [21] divided such interfaces into three categories based on their operating principle, ordered by increasing level of complexity:

1. Direct measurement of tilting, rotation or amplitude;
2. Discrete (isolated) gesture recognition;
3. Continuous gesture recognition.

Direct measurement of tilting, rotation or amplitude has been used for navigation through selections by tilting the device [29, 1, 8], and for shake detection [4]. Most research on accelerometer-based gesture recognition has focused on the second category, i.e., discrete gesture recognition. In discrete

gesture recognition, the start and end points of the gesture are signaled by the user via a button press [22, 12, 21, 4, 3]. As Bartlett [1] points out, button-based discrete recognition has the obvious problem that users have to synchronize the gesture and the button press, which may be inconvenient. However, the third category, accelerometer-based continuous gesture recognition, where the gestures are spotted automatically by the recognizer, is still very much an open research problem.

The recent work of Cho *et al.* [3] on discrete gesture recognition suggests continuous gesture recognition as possible future work for enabling sequential gestures, e.g., entering a sequence of digits by writing them in the air. In our previous work [15] we proposed a two-pass algorithm where gestures are automatically spotted in the first phase based on the magnitudal features of the acceleration signal, and classified in the second phase — unfortunately, the false positive rate of the resulting system is too high to be used in a mobile device. In the so-called *filler models* the HMM is enhanced with garbage states representing the intermittent non-gestural regions. This allows simultaneous spotting and classification of gestures, and has been applied successfully in both position-based [23] and vision-based [18, 36] gesture recognition; however, the filler model originates from speech recognition literature [35]. In this thesis we adopt such filler models — for the first time, to our knowledge — to the specific problem of accelerometer-based continuous gesture recognition. Other contributions of the thesis include some novel feature extraction algorithms, discussed further in Section 4.3.

The review in this section has been deliberately breadth-wise; many details of the previously proposed algorithms were omitted for conciseness. These details will be discussed depth-wise in the following sections.

3 The Hidden Markov Model

In this section, we shall study families of stochastic processes known as Markov processes. As statistical models, they define convenient probability distributions over fixed-length time series, thus giving simple "recipes" for evaluating densities (or probabilities) of time series, and also for performing more complicated statistical inference, such as recovering latent variables.

We will first study the *Markov chain*, which is a stochastic process generating a sequence of states from a finite state set. The HMM extends the Markov chain by augmenting each state with an *observation*. In addition, the state variables are assumed latent (thus *hidden* Markov model), and in practice they must be either marginalized with the forward algorithm (Section 3.3), recovered with the likelihood-maximizing Viterbi algorithm (Section 3.4), or eliminated using expectation with the Baum–Welch algorithm (Section 3.5.3).

3.1 Markov Chains

Let $t \in \{1, \dots, T\}$ denote the quantized time, where T is the upper bound of time (unbounded if $T = \infty$). Also, let the elements of $X = (X_1, \dots, X_T) : \Omega \rightarrow \mathcal{X}^T$ denote the *state variables* at each time, where Ω is the set of events and \mathcal{X} is the finite state set. Without loss of generality, we can assume that $\mathcal{X} = \{1, \dots, K\}$, where K is the number of possible states — this kind of indexing simplifies later formulae.

The collection of state variables $\{X_t : t \in \mathcal{T}\}$ is an example of a stochastic process with a finite state space \mathcal{X} and a discrete index set $\mathcal{T} = \{1, \dots, T\}$ [34, pp. 381-383, Section 23.1]. In general, the probability of a given state sequence can be written using the chain rule [34, p. 382, equation 23.1]:

$$\Pr[X = x] = \prod_{t=1}^T \Pr[X_t = x_t | X_1 = x_1, \dots, X_{t-1} = x_{t-1}]. \quad (3.1)$$

This means that an arbitrary probability distribution over state sequences can be defined by specifying the conditional probabilities of each state given its preceding states.

Unfortunately, the number of conditional probabilities grows exponentially

with T , and is unbounded if $T = \infty$. Therefore, a common assumption is the conditional independence of X_t from all other states given its p first preceding states X_{t-1}, \dots, X_{t-p} . This assumption is known as the *Markov assumption*, and the corresponding processes are called *p-th order Markov chains*.

For a p -th order Markov chain, equation (3.1) simplifies to⁵

$$\Pr[X = x] = \prod_{t=1}^T \Pr[X_t = x_t | X_{t-1} = x_{t-1}, \dots, X_{t-p} = x_{t-p}]. \quad (3.2)$$

When $p = 1$, we have a first-order Markov chain⁶ (Figure 3.1), where equation (3.1) further simplifies to

$$\Pr[X = x] = \prod_{t=1}^T \Pr[X_t = x_t | X_{t-1} = x_{t-1}].$$

A first-order Markov chain is formally denoted as a pair $\lambda = (\pi, A)$, where $\pi = (\pi_1, \dots, \pi_K)$ is the *prior vector* and $A = (a_{ij}) \in \mathbb{R}^{K \times K}$ is the *transition matrix*, defined as

$$\pi_i = \Pr[X_1 = i]$$

and

$$a_{ij} = \Pr[X_t = j | X_{t-1} = i].$$

With this parametrization, equation (3.1) can be expressed as

$$\Pr[X = x; \lambda] = \pi_{x_1} \prod_{t=1}^{T-1} a_{x_t, x_{t+1}}.$$

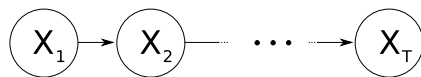


Figure 3.1: A first-order Markov chain and its dependencies depicted as a directed acyclic graph [34, pp. 264-271].

⁵When $t \leq p$ on the right hand side of equation (3.2), not all preceding states of X_t are available since $t - p < 1$. These non-existent state variables are implicitly removed from the condition.

⁶Some authors refer to first-order Markov chains as just Markov chains (e.g., [34, p. 383]).

Notice that the transition probabilities are independent of time — such a chain is said to be homogeneous [34, p. 384]. Unless otherwise mentioned, we restrict ourselves to homogeneous chains.

3.1.1 Parameter Estimation

Given N training sequences x^1, \dots, x^N , each of the form

$$x^k = (x_1^k, \dots, x_{T_k}^k), \quad k = 1, \dots, N,$$

we can estimate the parameters of the corresponding first-order Markov chains by counting the occurrences of the respective events:

$$\hat{\pi}_i = \frac{1}{N} \sum_{k=1}^N \delta(x_1^k, i)$$

and

$$\hat{a}_{ij} = \frac{\sum_{k=1}^N \sum_{t=1}^{T_k-1} \delta(x_t^k, i) \delta(x_{t+1}^k, j)}{\sum_{k=1}^N \sum_{t=1}^{T_k-1} \delta(x_t^k, i)}.$$

Here we have used the delta function

$$\delta(i, j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j \end{cases}$$

for notational simplicity. It can be shown that such $\hat{\pi}_i$ and \hat{a}_{ij} are the maximum likelihood estimates of π_i and a_{ij} [34, p. 394].

3.2 From Markov Chains to HMMs

HMMs extend the first-order Markov chains of Section 3.1 by augmenting each state X_t with an observation Y_t . The observations can be discrete (in which case they are sometimes referred to as *symbols*) or continuous. In the following we assume the observations to be continuous for notational convenience, but the treatment for discrete observations is essentially similar.

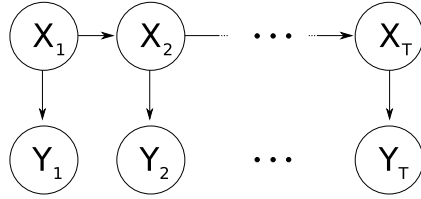


Figure 3.2: A hidden Markov model and its dependencies depicted as a directed acyclic graph [34, pp. 264-271].

In HMMs, we assume that each observation Y_t is conditionally independent of the rest of states and observations given its corresponding state X_t (Figure 3.2). Let $\theta = (\theta_1, \dots, \theta_K)$ denote the parameters of each conditional density $f_{Y_t|X_t=i}(\cdot; \theta_i)$. Then an HMM can be parametrized as a triple $\lambda = (\pi, A, \theta)$, where π and A are defined similarly as with first-order Markov chains.

With these assumptions, the density of a state sequence $X = (X_1, \dots, X_T)$ and its corresponding observation sequence $Y = (Y_1, \dots, Y_T)$ can be expressed as

$$f_{X,Y}(x, y; \lambda) = \pi_{x_1} \prod_{t=1}^{T-1} a_{x_t, x_{t+1}} \prod_{t=1}^T f_{Y_t|X_t=x_t}(y_t; \theta_{x_t}).$$

A discussion of some popular observation distribution families can be found in Appendix A.

3.3 Evaluation of Observation Sequence Densities

When only the observation sequence Y is known, and the state sequence X is latent, we can evaluate the density of the observation sequence by marginalizing over X :

$$f_Y(y; \lambda) = \sum_{x \in \mathcal{X}^T} f_{X,Y}(x, y; \lambda).$$

However, the number of possible state sequences x is K^T , which is prohibitively large. A faster algorithm can be constructed using recursion, as follows.

Let *forward probability* $\alpha_t(j)$ denote the density of ending in state j at time t , and the observations up to that moment, that is

$$\alpha_t(j) = f_{X_t, Y_1, \dots, Y_t}(j, y_1, \dots, y_t; \lambda).$$

A recurrence relation for $\alpha_{t+1}(j)$ can be constructed by marginalizing over X_t :

$$\begin{aligned} \alpha_{t+1}(j) &= f_{X_{t+1}, Y_1, \dots, Y_{t+1}}(j, y_1, \dots, y_{t+1}; \lambda) \\ &= \sum_{i=1}^K f_{X_t, Y_1, \dots, Y_t}(i, y_1, \dots, y_t; \lambda) a_{ij} f_{Y_{t+1}|X_{t+1}=j}(y_{t+1}; \theta_j) \\ &= \sum_{i=1}^K \alpha_t(i) a_{ij} f_{Y_{t+1}|X_{t+1}=j}(y_{t+1}; \theta_j) \end{aligned}$$

with the special case

$$\alpha_1(j) = \pi_j f_{Y_1|X_1=j}(y_1; \theta_j).$$

This means that the "layer" of values $\alpha_{t+1}(\cdot)$ for a fixed t can be computed in $\Theta(K^2)$ time when the values $\alpha_t(\cdot)$ are known. By ordering the computation by increasing t , we can compute all $\alpha_t(\cdot)$ in $\Theta(K^2T)$ time.⁷ Since we only need to store the layer at immediately previous t , the memory requirements

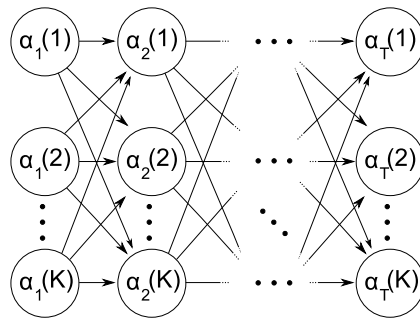


Figure 3.3: Dependencies in the forward algorithm. Time increases to the right. Here we see clearly how each vertical layer is dependent only on its immediately previous vertical layer.

⁷This is the asymptotic *worst-case* tight bound. If the input HMM has a special topology, faster algorithms can be constructed (Section 3.7).

are only $\Theta(K)$. This procedure, known as the forward algorithm [27], is illustrated in Figure 3.3.

Once the last layer of values $\alpha_T(\cdot)$ has been computed, the resulting density $f_Y(y; \lambda)$ can be computed by marginalizing over X_T :

$$\begin{aligned} f_Y(y; \lambda) &= \sum_{j=1}^K f_{X_T, Y}(j, y; \lambda) \\ &= \sum_{j=1}^K \alpha_T(j). \end{aligned}$$

3.4 Finding the Most Likely State Sequence

Given an observed sequence $y = (y_1, \dots, y_T)$, we often wish to solve the maximum likelihood problem

$$\nu = \max_x f_{X, Y}(x, y; \lambda),$$

that is, we wish to find the state sequence $x = (x_1, \dots, x_T)$ that yields the maximum density together with y .

Let $\nu_j(t)$ denote the highest density attainable up to time t so that we end in state j , i.e.,

$$\nu_j(t) = \max_{x_1, \dots, x_{t-1}} f_{X_1, \dots, X_t, Y_1, \dots, Y_t}(x_1, \dots, x_{t-1}, j, y_1, \dots, y_t; \lambda).$$

Clearly we have

$$\nu = \max_j \nu_j(T).$$

We can formulate $\nu_j(t+1)$ recursively by rewriting it as follows (omitting some density function subscripts and parameters for clarity):

$$\begin{aligned} \nu_j(t+1) &= \max_{x_1, \dots, x_t} \{f(x_1, \dots, x_t, y_1, \dots, y_t) a_{x_t, j} f_{Y_{t+1}|X_{t+1}=j}(y_{t+1})\} \\ &= \max_{x_1, \dots, x_{t-1}} \{ \max_{x_t} \{f(x_1, \dots, x_t, y_1, \dots, y_t) a_{x_t, j}\} \} f_{Y_{t+1}|X_{t+1}=j}(y_{t+1}) \\ &= \max_{x_t} \{ \max_{x_1, \dots, x_{t-1}} \{f(x_1, \dots, x_t, y_1, \dots, y_t) a_{x_t, j}\} \} f_{Y_{t+1}|X_{t+1}=j}(y_{t+1}) \\ &= \max_{x_t} \{ \nu_{x_t}(t) a_{x_t, j} \} f_{Y_{t+1}|X_{t+1}=j}(y_{t+1}) \\ &= \max_{1 \leq i \leq K} \{ \nu_i(t) a_{i, j} \} f_{Y_{t+1}|X_{t+1}=j}(y_{t+1}) \end{aligned}$$

with the special case

$$\nu_j(1) = \pi_j f_{Y_1|X_1=j}(y_1).$$

Again, we can order the computation by ascending t , as we did with the forward algorithm, ending up with time complexity $\Theta(K^2T)$. This algorithm is known as the Viterbi algorithm [27].

To recover the maximum likelihood path x_1, \dots, x_T we also need to keep track of which argument values yielded the maximums. This requires $\Theta(KT)$ storage. However, if we are only interested in the window of D last values of the path (i.e., x_{T-D}, \dots, x_T), we only need to store D "layers" at a time, ending up with $\Theta(DK)$ storage. This way the Viterbi algorithm can be run in an online fashion even for unbounded T , assuming that computations are numerically robust (Section 3.6).

3.5 Estimating Model Parameters

Estimating the model parameters $\lambda = (\pi, A, \theta)$ from training data is the hardest subproblem concerning HMMs. Given N observation sequences y^1, \dots, y^N of the form

$$y^k = (y_1^k, \dots, y_{T_k}^k), \quad k = 1, \dots, N,$$

the problem is to find the maximum likelihood parameter estimate

$$\hat{\lambda}^{\text{ML}} = \underset{\lambda}{\operatorname{argmax}} \prod_{k=1}^N f_{Y^k}(y^k; \lambda).$$

As in Section 3.3 and Section 3.4, the problem is that the state variables are missing: the estimation algorithm must be able to build the "semantics" of the states from the observation data alone.

3.5.1 Additional Notation

Before delving into the details of the parameter estimation procedures, we will first introduce some additional notation used later to simplify formulae.

Let $y = (y_1, \dots, y_T)$ be a single observation sequence. In Section 3.3 we defined the *forward probabilities*

$$\alpha_t(j) = f_{X_t, Y_1, \dots, Y_t}(j, y_1, \dots, y_t; \lambda),$$

evaluated with the forward algorithm recurrence equations

$$\begin{cases} \alpha_1(j) = \pi_j f_{Y_1|X_1=j}(y_1; \theta_j), \\ \alpha_{t+1}(j) = \sum_{i=1}^K \alpha_t(i) a_{ij} f_{Y_{t+1}|X_{t+1}=j}(y_{t+1}; \theta_j), \quad t = 1, \dots, T-1. \end{cases}$$

Analogously, we can define the *backward probabilities* as

$$\beta_t(i) = f_{Y_{t+1}, \dots, Y_T|X_t=i}(y_{t+1}, \dots, y_T; \lambda).$$

These can be evaluated with the backward algorithm recurrence equations [27]:

$$\begin{cases} \beta_T(i) = 1, \\ \beta_t(i) = \sum_{j=1}^K a_{ij} f_{Y_{t+1}|X_{t+1}=j}(y_{t+1}; \theta_j) \beta_{t+1}(j), \quad t = T-1, \dots, 1. \end{cases}$$

Next we define two additional variables

$$\begin{aligned} \gamma_t(i) &= f_{X_t|Y=y}(i; \lambda), \\ \xi_t(i, j) &= f_{X_t, X_{t+1}|Y=y}(i, j; \lambda). \end{aligned}$$

The values $\gamma_t(i)$ denote the probabilities of being in state i at time t , and the values $\xi_t(i, j)$ denote the probabilities of transiting from state i to state j at time t , both probabilities being conditioned on the observations. They can be evaluated in terms of forward and backward probabilities as follows [27]:

$$\gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{f_Y(y; \lambda)}, \quad (3.3)$$

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} f_{Y_{t+1}|X_{t+1}=j}(y_{t+1}; \theta_j) \beta_{t+1}(j)}{f_Y(y; \lambda)} \quad (3.4)$$

where $f_Y(y; \lambda)$ is computed from the forward variables with the forward algorithm (Section 3.3).

3.5.2 The Expectation-Maximization Algorithm

Suppose for a moment that the state sequences x^1, \dots, x^N corresponding to the observation sequences y^1, \dots, y^N , each of the form

$$x^k = (x_1^k, \dots, x_{T_k}^k), \quad k = 1, \dots, N,$$

were known. For the sake of clarity, in this section we let $y = (y^1, \dots, y^N)$ and $x = (x^1, \dots, x^N)$ denote all available observed and hidden data, respectively. We define the *complete-data log-likelihood function* [2] $Q(\lambda|X = x, Y = y)$ as

$$Q(\lambda|X = x, Y = y) = \log f_{X,Y}(x, y; \lambda).$$

Finding the value of $\lambda = (\pi, A, \theta)$ that maximizes this function is trivial: the prior and transition parameters π and A are obtained from the formulae presented in Section 3.1.1, and the observation distribution parameters $\theta_1, \dots, \theta_K$ can be estimated straightforwardly since we know in which state x_t^k each observation y_t^k was generated (the maximum likelihood estimators for some common observation distribution families are provided in Appendix A).

Notice that $Q(\lambda|X = x, Y = y)$ can also be written as the conditional expectation

$$Q(\lambda|X = x, Y = y) = E_\lambda[\log f_{X,Y}(X, Y; \lambda)|X = x, Y = y].$$

If the hidden data x are not available, we can remove them from the condition:

$$\begin{aligned} Q(\lambda|Y = y) &= E_\lambda[\log f_{X,Y}(X, Y; \lambda)|Y = y] \\ &= \sum_x f_{X|Y=y}(x; \lambda) \log f_{X,Y}(x, y; \lambda). \end{aligned}$$

To make the maximization of the conditional expectation $Q(\lambda|Y = y)$ analytically tractable, we will perform the expectation with respect to a *fixed* parameter $\lambda' = (\pi', A', \theta')$ as

$$\begin{aligned} Q(\lambda|Y = y; \lambda') &= E_{\lambda'}[\log f_{X,Y}(X, Y; \lambda)] \\ &= \sum_x f_{X|Y=y}(x; \lambda') \log f_{X,Y}(x, y; \lambda). \end{aligned} \tag{3.5}$$

The function $Q(\lambda|Y = y; \lambda')$ is sometimes called the auxiliary function. We can turn this idea into an iterative algorithm starting with an initial estimate $\hat{\lambda}^0$ and constructing a sequence $\hat{\lambda}^0, \hat{\lambda}^1, \dots$ as

$$\hat{\lambda}^{t+1} = \operatorname{argmax}_\lambda Q(\lambda|Y = y; \hat{\lambda}^t), \quad t = 0, 1, \dots$$

This is the famed Expectation-Maximization (EM) algorithm [2]: we first perform expectation with respect to the old estimate and then perform maximization with respect to the new estimate. The iteration can be stopped for example when the relative difference between the log-likelihoods of $\hat{\theta}^t$ and $\hat{\theta}^{t+1}$ is less than a given threshold.

The convergence properties of the EM algorithm are outside the scope of this thesis, but suffice to say that the method generally converges to a local likelihood maximum [2].

3.5.3 The Baum–Welch Algorithm

The application of the EM algorithm to parameter estimation in HMMs yields the Baum–Welch algorithm [27, 2] [28, pp. 342-348]. Rabiner and Juang present an elegant derivation of the algorithm in their classic speech recognition book [28, pp. 344-346]. Here we will follow the outline of their derivation, but directly to the case of multiple input observation sequences ($N > 1$) and arbitrary observation distribution families.

Starting from equation (3.5) we have

$$\begin{aligned} Q(\lambda|Y = y; \lambda') &= \sum_x f_{X|Y=y}(x; \lambda') \log f_{X,Y}(x, y; \lambda) \\ &= \frac{1}{f_Y(y; \lambda')} \sum_x f_{X,Y}(x, y; \lambda') \log f_{X,Y}(x, y; \lambda) \\ &\sim \sum_x f_{X,Y}(x, y; \lambda') \log f_{X,Y}(x, y; \lambda), \end{aligned} \quad (3.6)$$

that is, when maximizing with respect to λ we don't have to worry about the constant denominator $f_Y(y; \lambda')$.

The factors present in equation (3.6) can be written out as

$$\begin{aligned} f_{X,Y}(x, y; \lambda') &= \prod_{k=1}^N f_{X^k, Y^k}(x^k, y^k; \lambda') \\ &= \prod_{k=1}^N \pi_{x_1^k}' \prod_{t=1}^{T_k-1} a_{x_t^k, x_{t+1}^k}' \prod_{t=1}^{T_k} f_{Y_t^k | X_t^k = x_t^k}(y_t^k; \theta_{x_t^k}') \end{aligned}$$

and

$$\begin{aligned}
& \log f_{X,Y}(x, y; \lambda) \\
&= \sum_{k=1}^N \log f_{X^k, Y^k}(x^k, y^k; \lambda) \\
&= \sum_{k=1}^N \left(\log \pi_{x_1^k} + \sum_{t=1}^{T_k-1} \log a_{x_t^k, x_{t+1}^k} + \sum_{t=1}^{T_k} \log f_{Y_t^k | X_t^k = x_t^k}(y_t^k; \theta_{x_t^k}) \right).
\end{aligned}$$

Using these equations, equation (3.5) can be related to a sum of three types of auxiliary functions [28, pp. 344-345]:

$$Q(\lambda | Y = y; \lambda') \sim Q_\pi(\pi; \lambda') + \sum_{i=1}^K Q_{a_i}(a_i; \lambda') + \sum_{i=1}^K Q_{\theta_i}(\theta_i; \lambda'),$$

where

$$\begin{aligned}
Q_\pi(\pi; \lambda') &= \sum_{k=1}^N \sum_{j=1}^K f_{X_1^k, Y^k}(j, y^k; \lambda') \log \pi_j, \\
Q_{a_i}(a_i; \lambda') &= \sum_{k=1}^N \sum_{j=1}^K \sum_{t=1}^{T_k-1} f_{X_t^k, X_{t+1}^k, Y^k}(i, j, y^k; \lambda') \log a_{ij}, \\
Q_{\theta_i}(\theta_i; \lambda') &= \sum_{k=1}^N \sum_{t=1}^{T_k} f_{X_t^k, Y^k}(i, y^k; \lambda') \log f_{Y_t^k | X_t^k = i}(y_t^k; \theta_i)
\end{aligned}$$

and

$$a_i = (a_{i1}, \dots, a_{iK}).$$

Using the notation introduced in Section 3.5.1, these can be rearranged as

$$Q_\pi(\pi; \lambda') = \sum_{j=1}^K \left(\sum_{k=1}^N \gamma_1^k(j) \right) \log \pi_j, \quad (3.7)$$

$$Q_{a_i}(a_i; \lambda') = \sum_{j=1}^K \left(\sum_{k=1}^N \sum_{t=1}^{T_k-1} \xi_t^k(i, j) \right) \log a_{ij}, \quad (3.8)$$

$$Q_{\theta_i}(\theta_i; \lambda') = \sum_{k=1}^N \sum_{t=1}^{T_k} \gamma_t^k(i) \log f_{Y_t^k | X_t^k = i}(y_t^k; \theta_i), \quad (3.9)$$

where the values $\gamma_t^k(i)$ and $\xi_t^k(i, j)$ are computed using the fixed parameter λ' (the superscript k denotes the index of the input observation sequence).

These auxiliary functions depend on different variables, so they can be maximized independently. Notice that equations (3.7) and (3.8) are weighted sums of logarithms

$$\sum_{j=1}^K w_j \log z_j$$

subject to stochastic constraints $z_1, \dots, z_K \geq 0$ and $z_1 + \dots + z_K = 1$. Also notice that $w_1, \dots, w_K \geq 0$. It's easy to see, using the method of Lagrange multipliers, that the global maximum is obtained when [28, p. 345]

$$z_j = \frac{w_j}{\sum_{i=1}^K w_i}.$$

This leads to the following reestimation equations for prior and transition probabilities⁸:

$$\hat{\pi}_j = \frac{\sum_{k=1}^N \gamma_1^k(j)}{\sum_{i=1}^K \sum_{k=1}^N \gamma_1^k(i)},$$

$$\hat{a}_{ij} = \frac{\sum_{k=1}^N \sum_{t=1}^{T^k-1} \xi_t^k(i, j)}{\sum_{i=1}^K \sum_{k=1}^N \sum_{t=1}^{T^k-1} \xi_t^k(i, j)}.$$

Maximizing each $Q_{\theta_i}(\theta_i; \lambda')$ with respect to the corresponding θ_i in equation (3.9) is an example of a maximum pseudo likelihood problem. Essentially, for each θ_i the values $\gamma_t^k(i)$ act as weights for the observations y_t^k . Maximum pseudo likelihood estimators for some common observation distribution families are provided in Appendix A.

⁸The reestimation equations for multiple observation sequences in Rabiner and Juang [28, p. 369] are equivalent, but use the identities seen in equations (3.3) and (3.4).

3.5.4 Obtaining Initial Estimates

As mentioned in Section 3.5.2, parameter estimation is trivial when the state variables are known. A simple way of obtaining an initial estimate $\hat{\lambda}^0$ for the Baum–Welch algorithm is therefore to ”make up” these state variables. This can be done with the K-means clustering [28, pp. 125-129] by associating each state $1, \dots, K$ with a group of similar observations, or with piecewise segmentation [6, 7, 15, 14].

It’s pretty easy to see that if the initial transition probability $\hat{a}_{ij}^0 = 0$, then $\hat{a}_{ij}^t = 0$ for all t [27]. This gives us a simple recipe for enforcing constrained topologies (Section 3.7). On the other hand, this also means that it may not be preferable to use the transition probabilities obtained via clustering, since this may inadvertently result in zero probabilities. Instead, the initial transition probabilities can be set arbitrarily – for example, uniformly distributed. These notes also hold true for the prior probabilities.

3.6 Numerical Issues

In long time sequences, the probabilities $\alpha_j(t)$, $\beta_j(t)$ and $\nu_j(t)$ can become susceptible to numerical underflow. A traditional approach to alleviating this problem involves multiplying the probabilities with *scaling coefficients* [28, pp. 365-368]. An easier alternative is to perform all computations under logarithm transformation (e.g., [28, pp. 340-341]). We use the latter approach, since it can be implemented quite easily ”on top” of the computations. The logarithmization formulae are explained in more detail in Section B.

As noted in Section 3.4, the Viterbi algorithm can be run even for unbounded time sequences if we are only interested in monitoring a fixed-size window of optimal states. In this case, even the logarithmized probabilities $\log \nu_j(t)$ might become susceptible to negative drift, i.e., they might approach $-\infty$ as t grows without bound. In practice, we have never found this to be a problem. If it would ever turn out to be a problem, we could remove the mean component of $\log \nu_j(t)$ for each fixed t ; this would eliminate the drift while preserving the order relations.

3.7 Special Topologies

The state set $\mathcal{X} = \{1, \dots, K\}$ together with transition matrix $A \in \mathbb{R}^{K \times K}$ effectively defines a directed graph $G = (V, E)$, where

$$V = \mathcal{X}$$

and

$$E = \{ (i, j) \in \mathcal{X} \times \mathcal{X} \mid a_{ij} > 0 \}.$$

In this section we discuss some special topologies of these graphs.

If any node of the graph is reachable from any other node, in some number of steps, the model is said to be *ergodic*.⁹ In *left-to-right models* (also known as *Bakis models*) [28, pp. 348-350] the transition matrix A is constrained to be an upper diagonal matrix. This means that it's never possible to go back from state j to a state $i < j$, in any number of steps; the corresponding graph G is acyclic and has a convenient topological ordering.

Another constraint is to assume that A is a band matrix with a bandwidth of k , i.e., there can never be too long state "jumps". It's also common to combine the upper diagonal and band assumptions, i.e., assume $a_{ij} = 0$ for all $j < i$ and $j \geq i + k$ [28, pp. 348-350].

These constrained topologies have three advantages. First, they often present a more intuitive model for time-related phenomena. Second, sparse transition matrices lower the dimension of the parameter space, thus making parameter estimation more robust for small amounts of training data. Third, sparse transition matrices can significantly speed up the execution of HMM-related algorithms: when the number of possible transitions is $\Theta(K)$, evaluation of forward, backward, and Viterbi probabilities can be done in $\Theta(KT)$ time instead of $\Theta(K^2T)$ time.

The enforcement of these constraints upon parameter estimates was discussed in Section 3.5.4.

⁹Technically, the number of steps must not have a period greater than one with probability one [34, p. 390]. While such periodic chains are easy to construct artificially, they occur rarely in practice. Furthermore, the chain states must be positive, but this is implied by the finite number of states [34, p. 390, Lemma 23.20].

3.8 Discussion

We end this chapter with a discussion of pros and cons of Markov chains and the HMM, and possible variants.

A well-known restriction of the Markov chain is that the duration of staying in a state is geometrically distributed [27]. This may not be a realistic assumption in many problem domains, and extensive work has been done to incorporate explicit state duration densities via the semi-Markov model [28, pp. 358-362]. Unfortunately, the algorithms for semi-Markov models can be an order of magnitude slower than the corresponding algorithms for Markov chains.

On the other hand, the duration of staying in a *subset* of states is generally not geometrically distributed. Therefore, given enough of states, Markov chains should be able to encapture more expressive duration distributions. In a similar notion, it can be shown that HMMs with mixture densities (Appendix A.3) are equivalent to HMMs with simple densities but enough states [11]. Figuratively speaking, we could interpret the HMM as a "connectionist" model in which complex statistical behavior emerges from the interaction of a large number of simple states.

There are many more extensions and variants available of the HMM. For example, in segmental models [6, 7, 14] the observation densities depend not only on the state, but also on the duration that has been stayed in that state. Using heterogeneous HMMs, where the transition probabilities depend on the time, has been proposed to increase both robustness and speed [16].

4 Gesture Recognition

In this section, we shall put the theory of hidden Markov models into use by studying its application to both isolated and continuous recognition of gestures.

In addition to the general notation outlined in Section 1.1, in this section we use boldface letters to denote vectors in Euclidean spaces, e.g., $\mathbf{v} = (x, y, z) \in \mathbb{R}^3$.

4.1 Accelerometer Basics

A *single-axis accelerometer* measures physical acceleration along a specific direction. Let $\theta^{\text{LCS}} \in \mathbb{R}^3$ denote this direction in the sensor's local coordinate space (LCS). Also, let $\mathbf{a}^{\text{WCS}} \in \mathbb{R}^3$ denote the acceleration, excluding gravity, acting directly on the accelerometer in the world coordinate space (WCS).¹⁰

Let $F \in \mathbb{R}^{3 \times 3}$ be an orthonormal matrix that maps between directions in the local and world coordinate spaces. For example, for the acceleration we have¹¹

$$\mathbf{a}^{\text{LCS}} = F\mathbf{a}^{\text{WCS}}.$$

The accelerometer's one-dimensional reading a is given by the output equation [33, equation 2.1]

$$\begin{aligned} a(\theta^{\text{LCS}}) &= \langle \theta^{\text{WCS}}, \mathbf{a}^{\text{WCS}} \rangle \\ &= \langle F^T \theta^{\text{LCS}}, \mathbf{a}^{\text{WCS}} \rangle, \end{aligned}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product of two column vectors.

In the special case where the specific direction is along a coordinate axis, i.e., $\theta^{\text{LCS}} = \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$, where

$$\begin{aligned} \mathbf{e}_1 &= (1, 0, 0)^T, \\ \mathbf{e}_2 &= (0, 1, 0)^T, \\ \mathbf{e}_3 &= (0, 0, 1)^T, \end{aligned}$$

¹⁰A more general treatment where the force is not necessarily acting directly on the accelerometer is given by Tan and Park [33].

¹¹Tan and Park [33] use the opposite convention $\mathbf{a}^{\text{WCS}} = F\mathbf{a}^{\text{LCS}}$, but this would be inconvenient for our later discussion.

we have

$$\begin{aligned} a(\mathbf{e}_k) &= \langle F^T \mathbf{e}_k, \mathbf{a}^{\text{WCS}} \rangle \\ &= \langle F_k^T, \mathbf{a}^{\text{WCS}} \rangle, \end{aligned}$$

where F_1, F_2, F_3 are the rows of F .

A *triple-axis accelerometer* combines three single-axis accelerometers. For simplicity, we assume that the specific directions of the component accelerometers are $\mathbf{e}_1, \mathbf{e}_2$ and \mathbf{e}_3 , respectively. Then the output equation is given as

$$\mathbf{a} = \begin{pmatrix} a(\mathbf{e}_1) \\ a(\mathbf{e}_2) \\ a(\mathbf{e}_3) \end{pmatrix} = F \mathbf{a}^{\text{WCS}} = \mathbf{a}^{\text{LCS}}. \quad (4.1)$$

In other words, the readings for such a triple-axis accelerometer are effectively reported in the sensor's local coordinate space. Keeping this in mind, in the following we shall use \mathbf{a}^{LCS} and \mathbf{a} interchangeably.

4.2 Sensor Model

Over time, a triple-axis accelerometer produces a discretization of an ideal, continuous input signal

$$\mathbf{a}(t) = (a_x(t), a_y(t), a_z(t))^T \in \mathbb{R}^3, \quad t \geq 0.$$

For simplicity we assume that sampling is done at a constant frequency f , i.e., at a regular time interval $\Delta t = 1/f$. We denote the corresponding discretized signal as

$$\mathbf{a}[t] \approx \mathbf{a}(t\Delta t), \quad t \in \mathbb{N}.$$

An example of such a discretized acceleration signal is provided in Figure 4.1.

Detecting dynamic hand gestures from such a signal is complicated by two matters. First, as we saw in the previous section, sensor readings are reported in the sensor's local coordinate space — consequently, the orientation of the sensor affects the readings. This fact is already modeled by the matrix F in equation (4.1). Second, accelerometers not only measure the dynamic acceleration caused by user's hand movements, but also the static acceleration

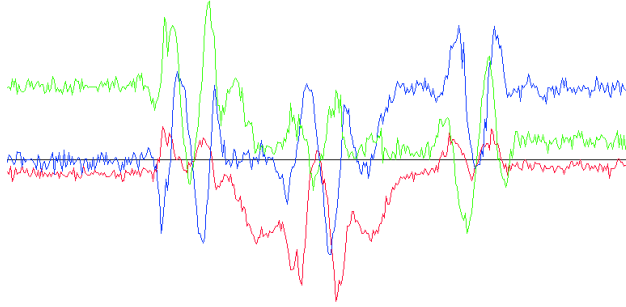


Figure 4.1: An example acceleration sequence recorded with a Nokia 5500 Sport mobile phone. Each color corresponds to a single signal channel: blue to a_x , green to a_y and red to a_z .

caused by gravitational resistance.¹²

We model the second fact by separating the world space acceleration into a dynamic (time-dependent) and a static (time-independent) component, i.e.,

$$\mathbf{a}^{\text{WCS}}(t) = \mathbf{a}_d^{\text{WCS}}(t) + \mathbf{a}_s^{\text{WCS}}.$$

The length of $\mathbf{a}_s^{\text{WCS}}$ is g , the gravitational constant ($g \approx 9.81 \frac{\text{m}}{\text{s}^2}$), and its direction is away from the center of the earth.¹³

This leads to a time-dependent output equation

$$\mathbf{a}(t) = F(t) (\mathbf{a}_d^{\text{WCS}}(t) + \mathbf{a}_s^{\text{WCS}}), \quad (4.2)$$

where $F(t) \in \mathbb{R}^{3 \times 3}$ is the orthonormal mapping between the local and world coordinate spaces. The sensor geometry is illustrated in Figure 4.2.

This sensor model is similar in nature, albeit much more simple, as vocal tract models used in speech recognition [28, pp. 132-139]. Essentially, both serve as a model of the underlying phenomena to offer a mathematical "starting point" for performing signal processing.

¹²Here resistance refers to the opposing force, not electrical resistance. For example, if the sensor is resting on a surface, the normal force resists gravity. However, if the sensor is, say, pinched between fingers to keep it from falling, there is no normal force *per se* — therefore, we decided to use the more encompassing term resistance.

¹³In practice, $\mathbf{a}_s^{\text{WCS}}$ is set to be in the direction of the sensor reading in a "typical" orientation, e.g., mobile phone screen facing up (when a phone's internal accelerometer is used). For all of our sensors, this corresponds to $\mathbf{a}_s^{\text{WCS}} = (0, 0, g)^T$.

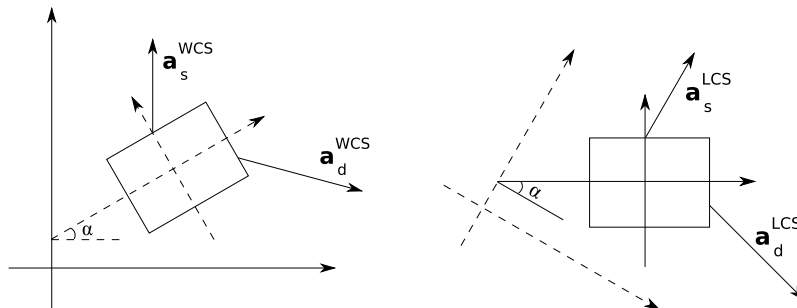


Figure 4.2: Two-dimensional illustration of the sensor geometry. The rectangle depicts the sensor. The left image is in the world coordinate space: $\mathbf{a}_s^{\text{WCS}}$ points upwards, and $\mathbf{a}_d^{\text{WCS}}$ is chosen arbitrarily. The right image shows the same setting in the sensor’s local coordinate space.

4.3 Featurization

The most straightforward way to build a gesture recognizer around the HMM framework is to use the input acceleration vectors $\mathbf{a}[t]$ as observations. A more general idea is to first transform the input acceleration sequence $\mathbf{a}[1], \dots, \mathbf{a}[T_1]$ into a sequence of *feature vectors* $\mathbf{f}[1], \dots, \mathbf{f}[T_2]$. The transformation of acceleration sequences to feature sequences is called *feature extraction* or *featurization*.¹⁴ In statistical nomenclature, features are also known as *covariates* [34, p. 209].

Featurization seeks to eliminate unnecessary variability while preserving essential differences relevant to recognition. Unnecessary variability might include the orientation of user’s hand [26, 15], amplitude of the acceleration [22, 12, 21, 4, 26, 3, 15], and temporal variability [12, 21, 3, 15].

A key advantage of eliminating unnecessary variability is that we may be able to estimate models with a lot less training data: we don’t need to teach the system with gestures of all combinations of hand orientation, amplitudes and temporal durations. Also, the distribution of the features might be simpler than the distribution of the corresponding accelerations, which further enhances estimability.

A complementary approach to featurization is to generate more training sam-

¹⁴Some authors, e.g., Cho *et al.* [3], distinguish between preprocessing and subsequent feature extraction. We make no such distinction; any mapping from acceleration sequences can be called a featurization.

ples via *simulation*, e.g., by adding white noise to copies of the original samples. The distribution of the noise can be based for example on signal-to-noise ratio [21] or principal component analysis [36].

4.3.1 Isolated Versus Continuous Featurization

In most previous work [22, 12, 4, 26, 3, 15], featurization is applied to an isolated batch of acceleration vectors $\mathbf{a}[1], \dots, \mathbf{a}[T]$ that (potentially) represents a single gesture. We refer to this as *isolated featurization*.

In continuous recognition of gestures, we have a potentially unbounded sequence of acceleration vectors $\mathbf{a}[1], \mathbf{a}[2], \dots$ from which the corresponding feature vectors $\mathbf{f}[1], \mathbf{f}[2], \dots$ must be computed on the fly. We refer to this as *continuous featurization*.

Clearly, continuous featurization is more complicated than isolated featurization since the start and end points of gestures are not known *a priori*.

4.3.2 Comparison to Speech Recognition

As Pylvänäinen [26] notes, featurization for accelerative gestures is inherently simpler than for speech recognition, since the input is already rather well suited for HMMs.

In speech recognition, the sampling frequency can be as high as 8 kHz to 16 kHz [30, p. 570]. At any given time, such a signal may contain multiple frequencies, and it is the *spectrum* of these frequencies that defines the "state" of speech (see [28, p. 21, Figure 2.9] for an example of a speech spectrogram). As a result, featurization for speech recognition usually involves some kind of spectral analysis.

In contrast, the acceleration signal produced by human hand can be sampled at a relatively low frequency: around 30 Hz should be enough [26]. Also, gestures aren't periodic in nature so they really aren't amenable to spectral analysis.

4.3.3 Estimating Gravitational Resistance

To begin the description of our featurization process, we will first cover methods for estimating the local space gravitational resistance

$$\mathbf{a}_s^{\text{LCS}}(t) = F(t)\mathbf{a}_s^{\text{WCS}}.$$

In isolated featurization, this has traditionally been estimated as the mean of the input accelerations (e.g., [26]) assuming constant $F(t)$ over the batch:

$$\hat{\mathbf{a}}_s^{\text{LCS}} = \frac{1}{T} \sum_{t=1}^T \mathbf{a}[t].$$

This simple averaging process is problematic since the acceleration is highly variable during gesturing. We propose the use of a weighted average

$$\hat{\mathbf{a}}_s^{\text{LCS}} = \frac{\sum_{t=1}^T h(\|\mathbf{a}[t]\| - g) \mathbf{a}[t]}{\sum_{t=1}^T h(\|\mathbf{a}[t]\| - g)},$$

where

$$h(x) = \frac{1}{\varepsilon + |x|}$$

and ε is a small positive constant to avoid division by zero. The motivation is that when the norm of the acceleration is close to g , the dynamic acceleration is small, and the static acceleration dominates in equation (4.2) — consequently, such inputs serve as more reliable estimates of the static acceleration, so we assign them higher weights.

For continuous featurization, we propose to solve $\mathbf{a}_s^{\text{LCS}}(t)$ via the following differential equation¹⁵:

$$\frac{d\hat{\mathbf{a}}_s^{\text{LCS}}}{dt}(t) = h(\|\mathbf{a}(t)\| - g) (\mathbf{a}(t) - \hat{\mathbf{a}}_s^{\text{LCS}}(t)).$$

¹⁵We already proposed a similar differential equation in [15]; here we have just added the weight function h . Notice that if the weight is constant, then the solution of the differential equation is simply an exponentially weighted moving average.

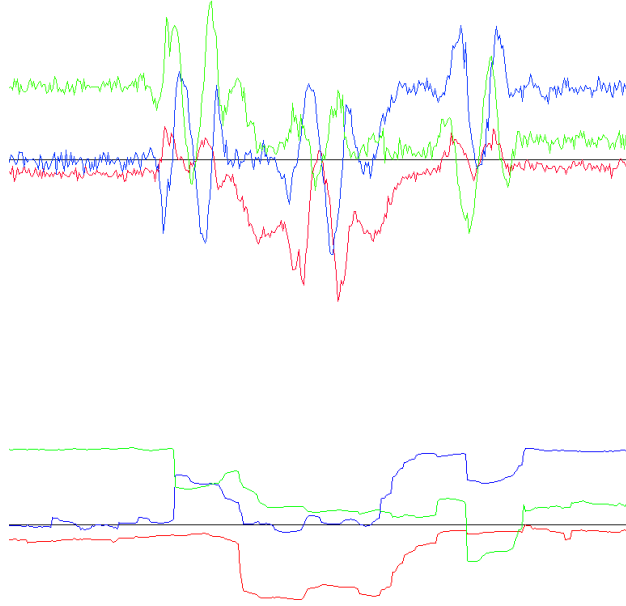


Figure 4.3: On the top is the acceleration sequence of Figure 4.1. At the bottom is the corresponding gravitational resistance estimate sequence $\hat{\mathbf{a}}_s^{\text{LCS}}$. Notice that the two signals are very similar when the acceleration is at rest, but that the gravitational resistance is much smoother when the acceleration varies wildly.

This differential equation can be solved analytically by assuming constant $\mathbf{a}(t)$ over time interval Δt :

$$\hat{\mathbf{a}}_s^{\text{LCS}}[t + 1] = \hat{\mathbf{a}}_s^{\text{LCS}}[t] + (1 - e^{-h(\|\mathbf{a}[t]\| - g)\Delta t}) (\mathbf{a}[t] - \hat{\mathbf{a}}_s^{\text{LCS}}[t]).$$

An example sequence of $\hat{\mathbf{a}}_s^{\text{LCS}}[t]$ is provided in Figure 4.3.

4.3.4 Tilt Compensation

When recognizing hand gestures, we are often not interested in the orientation of the sensor, but simply the translational direction of the user's hand in the world coordinate space – for example, whether the user moved his or her hand to the left. Ultimately, we are then interested in the quantity $\mathbf{a}_d^{\text{WCS}}$

in equation (4.2), for which we can derive an estimation formula

$$\hat{\mathbf{a}}_d^{\text{WCS}}(t) = \hat{F}^{-1}(t) (\mathbf{a}(t) - \hat{\mathbf{a}}_s^{\text{LCS}}(t)).$$

Constructing the estimate $\hat{\mathbf{a}}_s^{\text{LCS}}(t)$ was covered in the previous section. We must still construct the estimate $\hat{F}^{-1}(t)$. We do this by solving the equation

$$\hat{\mathbf{a}}_s^{\text{LCS}}(t) = c\hat{F}(t)\mathbf{a}_s^{\text{WCS}}, \quad c > 0. \quad (4.3)$$

In other words, we seek an $\hat{F}(t)$ that "rotates" the world space gravitational resistance so that it points to the direction of the estimated local space gravitational resistance.

Pylvänäinen [26] constructs the rotation via a Gram-Schmidt process by using the unit vector in the direction of $\hat{\mathbf{a}}_s^{\text{LCS}}(t)$ as one basis vector and choosing two other basis vectors arbitrarily, e.g., $(1, 0, 0)^T$ and $(0, 1, 0)^T$. However, it's not obvious how this behaves geometrically, especially in the presence of very large changes in orientation (over 90 degrees). In [15] we proposed a simple geometrical method for solving equation (4.3), described below.

We construct the matrix $\hat{F}^{-1}(t)$ by fixing the axis of rotation to

$$\mathbf{v}(t) = \mathbf{a}_s^{\text{WCS}} \times \hat{\mathbf{a}}_s^{\text{LCS}}(t).$$

Notice that $\mathbf{v}(t)$ is always perpendicular to $\mathbf{a}_s^{\text{WCS}}$. This is desirable since $\hat{\mathbf{a}}_s^{\text{LCS}}$ does not carry information about the sensor's rotation around $\mathbf{a}_s^{\text{WCS}}$ — in fact, rotating the sensor around $\mathbf{a}_s^{\text{WCS}}$ has no effect on its reading. The rotation matrix is given as [24, Appendix G]

$$\hat{F}^{-1} = \mathbf{u}\mathbf{u}^T + (\cos \alpha) (I - \mathbf{u}\mathbf{u}^T) + (\sin \alpha)S,$$

where

$$\begin{aligned} \alpha &= \arccos \left(\frac{\mathbf{a}_s^{\text{WCS}} \cdot \hat{\mathbf{a}}_s^{\text{LCS}}}{\|\mathbf{a}_s^{\text{WCS}}\| \|\hat{\mathbf{a}}_s^{\text{LCS}}\|} \right), \\ \mathbf{u} &= \frac{\mathbf{v}}{\|\mathbf{v}\|} \equiv (x, y, z)^T, \\ S &= \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}. \end{aligned}$$

Notice that α is simply the angle between $\mathbf{a}_s^{\text{WCS}}$ and $\hat{\mathbf{a}}_s^{\text{LCS}}$.

An example sequence of $\mathbf{a}_d^{\text{WCS}}[t]$ is provided in Figure 4.4.

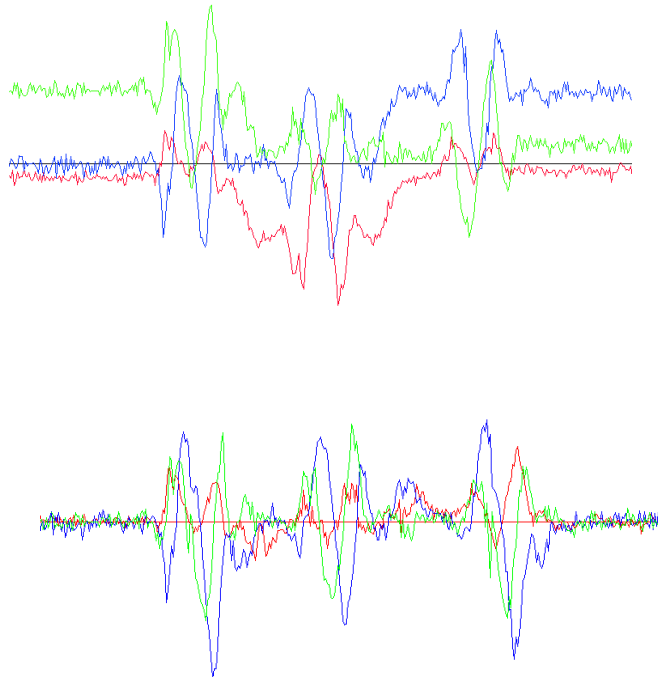


Figure 4.4: On the top is the acceleration sequence of Figure 4.1. At the bottom is the corresponding world-space dynamic acceleration estimate sequence $\hat{\mathbf{a}}_d^{\text{WCS}}$. The raw acceleration sequence contains three "clockwise" hand movements performed with highly varying orientations of the device, and it's hard to tell they are of the same type. In the world-space dynamic acceleration sequence the three movements are much more similar.

4.3.5 Amplitude Compensation

In the case of isolated featurization, compensating for the varying amplitudes of acceleration has been very popular. Many approaches have been proposed: normalizing component-wise variance [22, 12], min-max scaling [21], normalizing the data frame formed by the acceleration sequence with respect to some matrix norm [15], or simply fixing each acceleration to have unit length [3].

Unfortunately, in the case of continuous featurization the lack of endpoint information limits the possibility of amplitude compensation, so we ignore it in the context of this thesis.

4.3.6 Tempo Compensation

In the case of isolated featurization, the acceleration sequence with T_1 samples can be rescaled to have a fixed number T_2 of samples [12, 21, 15], e.g., $T_2 = 40$ in [21]. Cho *et al.* [3] employ a more elaborate tempo compensation procedure where time is warped so that the Euclidean distance between two successive accelerations remains constant.

In the case of continuous featurization, compensation for tempo changes is not straightforward and so it is ignored in this thesis. We note, however, that the HMM implicitly incorporates a form of dynamic time warping, which alleviates the problem.

4.3.7 Magnitudal Features

In our previous work [15] we used a set of features based on the *magnitude* of the acceleration in order to extract ballistic subsequences from a continuous acceleration sequence (Section 4.4.3). In the same paper we hypothesized that the inclusion of such magnitudal features to our feature vectors $\mathbf{f}[t]$ might lead to better spotting performance in the context of filler models (Section 4.5). The validity of this hypothesis will be studied later in Section 6.3.2.

We propose the use of the following magnitudal feature vector:

$$\mathbf{f}_{\text{mag}}(t) = \begin{pmatrix} \|\mathbf{a}(t)\| \\ \|\mathbf{a}'(t)\| \end{pmatrix} \in \mathbb{R}^2,$$

where $\mathbf{a}'(t)$ is discretized as

$$\mathbf{a}'[t] = \frac{1}{\Delta t} (\mathbf{a}[t] - \mathbf{a}[t-1]).$$

Our final feature vectors are then of the form

$$\mathbf{f}[t] = \begin{pmatrix} \hat{\mathbf{a}}_d^{\text{WCS}}[t] \\ \mathbf{f}_{\text{mag}}[t] \end{pmatrix} \in \mathbb{R}^5.$$

4.3.8 Primitivization

As their final featurization step, Cho *et al.* [3] extract *primitives* from the input acceleration sequence. These primitives correspond to short local extrema in the acceleration signal.

In this thesis we ignore such primitivization techniques since extending them to continuous recognition or multiple sensors is not straightforward.

4.4 Isolated Recognition

In *isolated recognition*, we are given an isolated, finite acceleration sequence (batch) $\mathbf{a}[1], \dots, \mathbf{a}[T]$ that (potentially) represents a single gesture instance, and we must decide which of M gesture types (e.g., "clockwise" and "counterclockwise") the batch represents.

In HMM-based isolated recognition, this is traditionally done by associating each of the M gesture types with a single HMM. Let $\lambda^1, \dots, \lambda^M$ denote the parameters of these HMMs. As explained in Section 3.2, each parameter is a triple

$$\lambda^k = (\pi^k, A^k, \theta^k), \quad k = 1, \dots, M,$$

where π^k is the prior vector, A^k is the transition matrix and θ^k are the observation density parameters.

4.4.1 Parameter Estimation

Most previous HMM-based gesture recognition research has used a constant amount of states in each of the HMMs $\lambda^1, \dots, \lambda^M$ [22, 21, 26, 36, 15]. However, variable state counts have been tried as well, where the amount of states is proportional to the complexity of the gesture [23, 18]. On the other hand, it has been found that the exact number of states is not too important [22]. With a constant number N of states the parameter components take the form

$$\pi^k \in \mathbb{R}^N, \quad A^k \in \mathbb{R}^{N \times N}, \quad \theta^k = (\theta_1^k, \dots, \theta_N^k), \quad k = 1, \dots, M.$$

Both ergodic (e.g., [21]) and left-to-right HMM topologies (e.g., [23, 18, 36, 15]) have been tried in the past. We use left-to-right HMMs with a bandwidth of one, as depicted in Figure 4.5 — as we shall see, this topology generalizes straightforwardly to the case of continuous recognition.

The observations in the HMMs are simply the features $\mathbf{f}[t]$ of Section 4.3. Both discrete (e.g., [22, 21]) and continuous (e.g., [26, 15]) observation densities have been proposed. However, Mäntylä *et al.* [22] found that in the case of discrete densities, choosing a good size for the vector quantization codebook (Appendix A.4.1) is essential. For simplicity, we use continuous densities, more precisely multivariate Gaussians (Appendix A.2).

Each HMM is trained separately. First, we record a set of acceleration sequences, each representing the gesture type to be trained. (The recording procedure is explained in detail in Section 5.3.) Next, the model parameters are estimated using the Baum–Welch algorithm (Section 3.5.3), where the initial estimates are obtained with piecewise linear regression [6, 7, 15, 14].

4.4.2 Recognition

In the recognition phase, an input acceleration sequence $\mathbf{a}[1], \dots, \mathbf{a}[T_1]$ is first featurized into a feature sequence $\mathbf{f}[1], \dots, \mathbf{f}[T_2]$. Assuming that all gesture types are equally likely, the optimal gesture type (classification) is given by [34, p. 352, Theorem 22.6]

$$\operatorname{argmax}_k f_{Y_1, \dots, Y_{T_2}}(\mathbf{f}[1], \dots, \mathbf{f}[T_2]; \lambda^k), \quad (4.4)$$

evaluated with the forward algorithm (Section 3.3).

If the acceleration sequence doesn't represent any of the known M gesture types, its density against each of the M gesture HMMs will be very small. A density threshold for detecting these unknown gestures can be based on the training data [4].

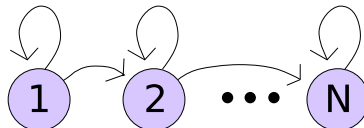


Figure 4.5: HMM topology for a single gesture with N states.

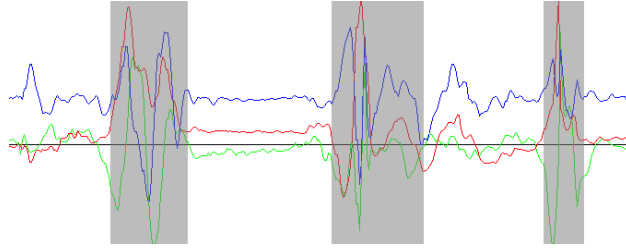


Figure 4.6: Three gestures spotted based on their magnitudal features. The regions of the gestures are darkened. Reprinted from [15].

This basic procedure for isolated gesture recognition is an example of density estimation. For each gesture type, a density function from the family of HMMs is estimated using the respective training gestures. These density functions are then used to rank unseen gestures. One problem with this approach is that since each density function is estimated separately, information about mutual differences between gesture types isn't utilized. Methods for taking such mutual information into account exist [27], but are outside the scope of this thesis.

4.4.3 Gesture Spotting

So far, we have not explained where the input acceleration batches come from, and how we know that they represent gestures in the first place. How can we "extract" these subsequences from the sequence produced by the sensor?

In most previous work on accelerometer-based gesturing, e.g., [21, 3], this extraction is done by requiring the user to hold down a button while performing a gesture. Another idea is to somehow automate this extraction – also known as *endpoint detection* (e.g., [7] [28, pp. 143-148]) or *spotting* (e.g., [36]).

In machine vision, a spotting algorithm based on the idea that some gestures have distinctively ballistic velocity profiles has been proposed [25]. In our previous work [15] we proposed a similar algorithm for accelerometer-based gesture recognition (Figure 4.6). Unfortunately, our algorithm suffered from excess false positives since the users often perform ballistic movements not corresponding to any known gesture type. In addition, we don't really want

to require the users to perform the gestures with *too* much strength, since this is physiologically demanding.

4.5 Continuous Recognition

In *continuous recognition* we are given a potentially unbounded input acceleration sequence $\mathbf{a}[1], \mathbf{a}[2], \dots$ from which we must recognize M known gesture types on the fly.

In the HMM framework, this can be accomplished by adding garbage states that represent the intermittent regions not corresponding to sought keywords [35], creating a so-called filler model. This idea has been applied in position based [23] and vision based [18, 36] gesture recognition. Nam and Wohn [23] called their garbage states *juncture HMMs*. Lee and Kim [18] call their model *threshold model*, but it is similar in nature to the filler model.

Unfortunately, Nam and Wohn [23] omit the details of how their juncture models are constructed. Lee and Kim [18] and Yang *et al.* [36] create their garbage states by duplicating all states of the gesture HMMs. Their garbage states are fully connected, i.e., there is a transition from each garbage state to each garbage state. Since the number of connections is quadratic, for efficiency they reduce the number of garbage states by merging states with with small Kullback-Leibler divergences [34, p. 126, equation 9.6].

In these previously proposed gestural filler models, we end up with *several* garbage states. However, there is a well-known connection between multi-state HMMs and mixture densities [11]. Therefore, we propose the use of a filler model that has only a *single* garbage state with a mixture observation density. This is conceptually simple and lowers the dimension of the parameter space considerably.

To elucidate this point, consider a single-state HMM with a mixture observation distribution. The observation density is of the form

$$f_Y(y; \theta) = \sum_{i=1}^M \alpha_i f_{Z_i}(y; \theta_i),$$

where M is the mixture component count, $\alpha_1, \dots, \alpha_M$ are the mixture weights, $\theta_1, \dots, \theta_M$ are the component density parameters and Z_1, \dots, Z_M are the

component random variables. (See Appendix A.3 for a detailed explanation of the mixture distribution.) This single-state HMM is clearly equivalent to an M -state HMM with prior vector

$$\pi = (\alpha_1, \dots, \alpha_M),$$

transition matrix

$$A = \begin{pmatrix} \alpha_1 & \dots & \alpha_M \\ \vdots & \ddots & \vdots \\ \alpha_1 & \dots & \alpha_M \end{pmatrix}$$

and observation density functions

$$f_{Y_t|X_t=k}(y; \theta_k) = f_{Z_k}(y; \theta_k), \quad k = 1, \dots, M.$$

Notice that all rows of the transition matrix are equal. This is the main limitation of the single-state mixture HMM that we use in our filler model.

4.5.1 Parameter Estimation

Our filler model aims to model all M gesture types plus the intermittent "garbage" acceleration simultaneously. This is done by combining the N -state HMMs $\lambda^1, \dots, \lambda^M$ of Section 4.4.1 into a large, sparse HMM with $K = 1 + NM$ states, where the first state models the garbage and the rest of the states model the gesture types. The filler HMM can therefore be parametrized by the triple

$$\lambda = (\pi, A, \theta),$$

where $\pi = (1, 0, \dots, 0) \in \mathbb{R}^K$ is the prior vector and

$$\theta = (\theta_G, \theta_1^1, \dots, \theta_N^1, \dots, \theta_1^M, \dots, \theta_N^M)$$

is the tuple of observation density parameters.

The garbage density parameter θ_G is estimated using all available gesture data plus special garbage data representing actions such as walking, running and generally moving around with the sensor. The samples are weighted so that

$$\frac{W_{\text{garbage}}}{W_{\text{garbage}} + W_{\text{gesture}}} = \omega_G.$$

Here W_{garbage} and W_{gesture} are the sums of garbage and gesture sample weights, respectively, and the garbage weight percentage $\omega_G \in [0, 1]$ is a configurable

parameter that describes how much weight is given to garbage samples as a whole. This makes the estimate independent of the relative amounts of gesture and garbage training data. We use a mixture multivariate normal distribution with M_G components to model the garbage density. The effect of the garbage weight percentage ω_G and garbage mixture component count M_G is studied later in Section 6.3.

The transition matrix $A \in \mathbb{R}^{K \times K}$ is defined in block form as follows:

$$A = \begin{pmatrix} \gamma & \bar{\pi}^1 & \bar{\pi}^2 & \dots & \bar{\pi}^M \\ \omega^1 & \bar{A}^1 & 0 & \dots & 0 \\ \omega^2 & 0 & \bar{A}^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^M & 0 & 0 & \dots & \bar{A}^M \end{pmatrix}. \quad (4.5)$$

Here the scalar γ represents the probability of remaining in the garbage state, and is chosen in an *ad hoc* fashion. The effect of the parameter γ will be studied in Section 6.3.7.

The *modified prior vectors* $\bar{\pi}^1, \dots, \bar{\pi}^M$ are defined as

$$\bar{\pi}^k = \frac{1 - \gamma}{M} \pi^k, \quad k = 1, \dots, M \quad (4.6)$$

to ensure that the first row of A sums up to one.

Each *modified transition matrix* \bar{A}^k for $k = 1, \dots, M$ is defined as

$$\bar{a}_{ij}^k = \begin{cases} a_{ij}^k, & \text{if } i \neq N \text{ or } j \neq N, \\ 1 - \left(\frac{1}{N-1} \sum_{l=1}^{N-1} (1 - a_{il}^k)^{-1} \right)^{-1}, & \text{if } i = N \text{ and } j = N, \end{cases} \quad (4.7)$$

where we have used the convention

$$\begin{aligned} A^k &= (a_{ij}^k), \\ \bar{A}^k &= (\bar{a}_{ij}^k) \end{aligned}$$

to denote the elements of the transition matrices. This modification is necessary since $a_{NN}^k = 1$ for all $k = 1, \dots, M$ by the left-to-right constraint. That is, there are no transitions out of the end states of individual gesture types. We know that the duration of staying in a state i in a Markov chain is geometrically distributed with an expected duration $E[d_i] = \frac{1}{1 - a_{ii}}$ [27]. Equation (4.7) is designed to satisfy $E[d_N] = \frac{1}{N-1} \sum_{i=1}^{N-1} E[d_i]$.

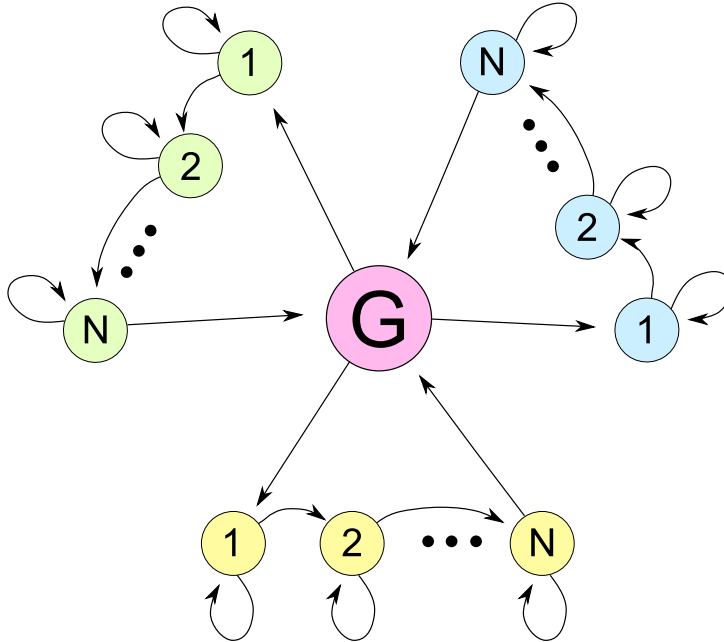


Figure 4.7: HMM topology in our filler model. In this example we have three N -state gesture types, depicted in different colors. Notice how the gesture states are connected to the single garbage state via their first and last states.

Finally, the *exit vectors* $\omega^1, \dots, \omega^M$ are defined as

$$\omega^k = \text{diag}(I - \bar{A}^k), \quad k = 1, \dots, M.$$

This ensures that all rows of A sum up to one. The topology of the filler HMM is depicted in Figure 4.7.

4.5.2 Recognition

In the recognition stage we are presented with a potentially unbounded acceleration sequence $\mathbf{a}[0], \mathbf{a}[1], \dots$. The corresponding feature vectors $\mathbf{f}[0], \mathbf{f}[1], \dots$ are used as input observations to the Viterbi algorithm.

For the sake of efficiency, we only store the $D = 64$ last observations in the Viterbi procedure, corresponding to a window of about one or two seconds, depending on the output frequency of the sensors.

At each time $t \in \mathbb{N}$, the Viterbi algorithm then decodes the maximum likeli-

hood estimates of the (at most D previous) hidden states in the filler HMM given the observations. If this state sequence contains a transition from a gesture type’s end state to the garbage state at time $t' \leq t$, we register that as a recognized gesture. The Viterbi algorithm is then restarted from time t' .

4.6 Extensions and Refinements

In order to make our recognizer more general and usable in real-life contexts, we propose a few simple extensions and refinements to it.

4.6.1 Multiple Sensors

We can use multiple sensors to recognize two-handed gestures, for example. Let S be the total number of sensors. Similar to the sensor model in Section 4.2, we assume the data arrives at a constant frequency (implementation details are provided in Section 5.2.3).

The acceleration data from each sensor is featurized independently and appended together to produce $5S$ -dimensional feature vectors. In the HMMs the observations are still modeled with (mixtures of) Gaussians, but with the constraint that covariances between data from separate sensors must be zero, i.e., the sensors are independent.¹⁶

Each gesture type generally depends only on a subset of sensors, e.g., we may wish to recognize both one-handed and two-handed gestures. In this case, the observation densities are modeled in such a way that they are they don’t depend on the sensors not in the respective subset.

4.6.2 Controlling the Subset of Recognized Key Gestures

We expect that as the number of recognized gesture types M grows, the error rates grow as well. Therefore, in real applications, it is often desirable to

¹⁶The independence assumption also makes it easy to marginalize the observation densities if some sensor(s) become unavailable e.g., due to lack of power.

recognize only a *subset* of gesture types at any given moment, effectively partitioning the M gesture types into *active* gesture types and *inactive* gesture types. At least Choi *et al.* [4] have used such partitioning in their real-world mobile phone implementation.

For isolated recognition, this refinement is trivial: we simply don't take inactive gesture types into account in equation (4.4).

For continuous recognition, we modify the transition matrix so that being in states corresponding to inactive gesture types is impossible.¹⁷ More precisely, let $M' > 0$ be the number of active gesture types.¹⁸ Then the modified prior vector (equation (4.6)) for gesture type $k = 1, \dots, M$ is

$$\bar{\pi}^k = \begin{cases} \frac{1-\gamma}{M'} \pi^k, & \text{if } k \text{ is an active gesture type,} \\ 0, & \text{if } k \text{ is an inactive gesture type.} \end{cases}$$

The modified transition matrix (equation (4.7)) remains unchanged for active gestures types, and becomes zero for inactive gesture types.

Notice that since the transition probabilities to inactive states are all zero, we can avoid performing the potentially costly observation density evaluations for these states in the Viterbi algorithm.

¹⁷Notice that this kind of heterogeneous (time-dependent) transition probabilities don't affect the correctness of the Viterbi algorithm.

¹⁸In the special case when $M' = 0$, i.e., when all gesture types are inactive, the transition matrix of the filler model (equation (4.5)) becomes such that the first column is all ones, and all other columns are all zeros.

5 Implementation

The algorithms described in Sections 3 and 4 were implemented in the Java programming language. We used a relatively portable subset of the language and a highly portable open source Javolution real-time standard library [5], so the recognizer works both on J2SE (e.g., desktop applications) and J2ME (e.g., mobile phones). However, for performance reasons, parameter estimation is always done on a desktop machine.

5.1 System Architecture

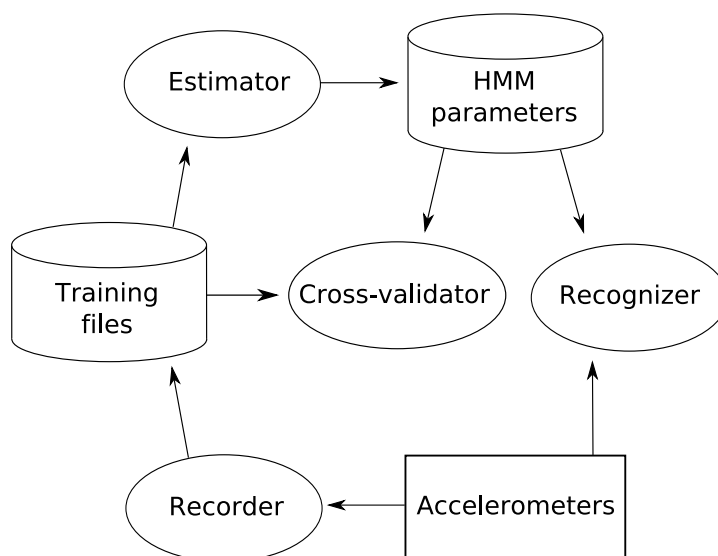


Figure 5.1: The architecture of the system. Rectangles denote hardware components, ellipses denote software components and cylinders denote file storage. Arrows indicate the direction of data flow.

The overall system architecture is depicted in Figure 5.1. In the following sections we will describe the individual hardware and software components in more detail.

5.2 Accelerometers

The system can use one or more accelerometers as input hardware. The accelerometers are responsible for sending their three-dimensional sensor readings at a constant frequency.

We implemented support for two sensor models: a commercial SHAKE SK6 sensor, and the inbuilt sensors in selected Nokia mobile phones.

5.2.1 SHAKE SK6

The SHAKE SK6 is a high-quality sensor produced by SAMH Engineering. It includes a triple-axis accelerometer, an optional triple-axis gyroscope, a triple-axis magnetometer, and capacitive sensors. The sensor readings are communicated wirelessly over Bluetooth. The sampling frequency of SHAKE SK6 is programmable: we chose a frequency of 50 Hz.

5.2.2 Nokia Mobile Phones

Some Nokia mobile phones, e.g., Nokia 5500 Sport and Nokia N95, include a triple-axis accelerometer with a sampling frequency of about 35 Hz. Unfortunately, current mobile phones don't yet implement a Java API for accessing the accelerometer data, although a specification for such Mobile Sensor API has already been published [10]. For the time being, we use a native Symbian application to access the accelerometer, and send the sensor readings to the Java-based recognizer over a localhost socket connection.

5.2.3 Synchronizing Accelerometers

If we use more than accelerometer, e.g., one for each hand, then we need to synchronize the asynchronous sensor readings. That is, we wish to produce a discretization of the compound acceleration signal

$$\mathbf{a}(t) = \begin{pmatrix} \mathbf{a}^1(t) \\ \vdots \\ \mathbf{a}^S(t) \end{pmatrix} \in \mathbb{R}^{3S},$$

where $\mathbf{a}^1(t), \dots, \mathbf{a}^S(t)$ are the input accelerations of each sensor and S is the number of sensors.

Let f^1, \dots, f^S be the input frequencies of each sensor. The compound frequency is then $f = \min_k f^k$ — that is, the slowest sensor sets the pace. The synchronization algorithm receives acceleration vectors from each sensor until each sensor has sent at least one vector; each component of the discretized compound acceleration is set to be the average of the acceleration vectors sent by the corresponding sensor.

5.3 Recorder

In order to train the system, the user(s) must first record exemplar gestures with a recorder application. In a single run of the application, a user will record repetitions of a single type of gesture, e.g., ten repetitions of gesture "clockwise". The recorder application receives readings from one or more accelerometers and writes them continuously into a file.

While performing an exemplar gesture, the user must hold down a button during the whole gesture.¹⁹ The input accelerations are "tagged" with a flag that describes whether the button was held down or not. This simplifies parameter estimation greatly since we know wherein the input acceleration sequence the gestures lie.

On the other hand, this approach has the disadvantage that pressing a button on the sensor constrains the orientation of the user's hand [1], and in the subsequent button-free recognition phase the hand orientation is less constrained. On the other hand, our tilt compensation algorithm (Section 4.3.4) is specifically designed to remove the effect of hand orientation, which alleviates the problem.

¹⁹In SHAKE SK6 sensors we use its navigation switch as the button, whereas in Nokia mobile phones we use the navigation key. It is also possible to use the spacebar key of the desktop keyboard.

5.3.1 Detecting Outliers

Anomalous training data can greatly decrease the quality of parameter estimates [12]. For example, if the training gestures are performed very slowly, it may not be possible to distinguish the gestures from non-gestural data during recognition. We use the following heuristic to detect such outlier data.

Given an acceleration sequence $\mathbf{a}[1], \dots, \mathbf{a}[T]$ corresponding to a single gesture, we first estimate the static acceleration component $\hat{\mathbf{a}}_s^{\text{LCS}}$ with weighted averaging (Section 4.3.3) and set the dynamic acceleration component as

$$\hat{\mathbf{a}}_d^{\text{LCS}}[t] = \mathbf{a}[t] - \hat{\mathbf{a}}_s^{\text{LCS}}, \quad t = 1, \dots, T.$$

The sequence is deemed anomalous if and only if

$$\frac{1}{T} \sum_{t=1}^T \|\hat{\mathbf{a}}_d^{\text{LCS}}[t]\| < \frac{g}{4}$$

or

$$\sqrt{\det \hat{\Sigma}_{\hat{\mathbf{a}}_d^{\text{LCS}}} < \frac{g}{4}},$$

where g is the gravitational constant and $\hat{\Sigma}_{\hat{\mathbf{a}}_d^{\text{LCS}}}$ is the unbiased estimate of the covariance matrix of the dynamic acceleration component. If there are multiple sensors, the sequence is deemed anomalous if and only if the above criterion holds true for all sensors related to the gesture.

We note that these formulae are completely *ad hoc*, and relate to the intuitive notions that gestures should be relatively fast and varied in order to be separable from non-gestures. *Ad hoc* removal of outliers was also used by Starner and Pentland [31], and also by Choi *et al.* [4].

5.4 Estimator

The estimator uses the training files produced by the recorder as its input data. It implements the filler model parameter estimation algorithm (Section 4.5.1), extended with support for multiple sensors (Section 4.6.1). Anomalous training data (Section 5.3.1) are not used in estimation.

5.5 Recognizer

The recognizer implements the filler model recognition algorithm (Section 4.5.2) extended with support for multiple sensors (Section 4.6.1) and the possibility to control the subset of recognized key gestures (Section 4.6.2).

5.6 Cross Validator

The cross validator uses the same algorithms as the recognizer, but against the training data files; indeed, the training data can be thought of as a "playback" accelerometer. This allows us to numerically measure how well the recognizer performs. The theory of cross validation is explained in more detail in Section 6.1.

6 Results

In this section, we evaluate the recognition performance of our proposed recognizer. Since we wish to run the recognizer on computationally restricted mobile devices, we will also measure the runtime performance of the system.

6.1 Methodology

We will briefly review the basics of classification [34, pp. 349-352], since these results will be used in our experiments.

Let $X : \Omega \rightarrow \mathcal{X}$ and $Y : \Omega \rightarrow \mathcal{Y}$ be random variables, where Ω is the set of events. The random variable X can be thought of as the predictor, and Y as the outcome. We assume that the outcome set \mathcal{Y} is finite, i.e., its elements can be thought of as classes. A *classifier* is any mapping $h : \mathcal{X} \rightarrow \mathcal{Y}$. For example, X could be the accelerometer data of a single gesture, Y could be the gesture class, and h could be an isolated HMM-based recognizer (Section 4.4).

Validation of a classifier h can be done by measuring its *true error rate* [34, p. 351]

$$L(h) = \Pr[h(X) \neq Y].$$

Given an independent and identically distributed random sample

$$Z_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$$

of (X, Y) , we can estimate L via the *empirical error rate* [34, p. 351]

$$\hat{L}_n = \frac{1}{n} \sum_{i=1}^n I\{h(X_i) \neq Y_i\},$$

where $I\{\cdot\}$ is the indicator function. However, if h is trained (estimated) from Z_n , then \hat{L}_n will clearly have a downward bias, since the same samples were used for both training and validation.

6.1.1 Cross Validation

Cross validation [34, pp. 363-364] is a general method for eliminating the downward bias on the estimated error rate. The idea is to introduce a *training set* \mathcal{T} and a *validation set* \mathcal{V} such that $\mathcal{T} \cup \mathcal{V} = Z_n$. The cross validated error rate estimate is

$$\hat{L}(\mathcal{T}, \mathcal{V}) = \frac{1}{|\mathcal{V}|} \sum_{(X,Y) \in \mathcal{V}} I \left\{ \hat{h}(X; \mathcal{T}) = Y \right\},$$

where $\hat{h}(\cdot; \mathcal{T})$ denotes the classifier trained using the set \mathcal{T} .

As long as $\mathcal{T} \cap \mathcal{V} = \emptyset$ ²⁰, the downward bias will be eliminated. However, cross validation with a training set of size $m < n$ introduces an *upward* bias in the estimated error rate since the validating classifier is trained with fewer samples (m) than the target classifier (n).²¹

This upward bias, in turn, can be alleviated by choosing m suitably close to n , e.g., $m = n - 1$. However, this will increase the variance of \hat{L} . The variance can be lowered by repeating the computation with different choices of \mathcal{T} and \mathcal{V} and averaging the results, at the cost of computation time.

To reduce variance systematically, K-fold cross validation [34, p. 364] can be applied. The set Z_n is partitioned into (approximately) equal-sized sets S_1, \dots, S_K , and the error rate estimate is given as

$$\hat{L} = \frac{1}{K} \sum_{k=1}^K \hat{L}(Z_n \setminus S_k, S_k).$$

K-fold cross validation is a form of stratified sampling that ensures that each sample is used for both training and validation, exactly the same number of times. The special case where $K = n$ is known as one-out cross validation.

6.1.2 Extension to Continuous Case

One problem with continuous recognition is that it doesn't straightforwardly lend itself to the framework of classification. A continuous gesture recognizer

²⁰It is possible to let \mathcal{T} and \mathcal{V} overlap, of course; in fact, the non cross validated estimate can be interpreted as a special case of cross validation with $\mathcal{T} = \mathcal{V} = Z_n$.

²¹Based on the intuitive assumption that using more training data lowers the error rate.

is best understood as a *detection system* that detects (spots) M types of keyword gestures on the fly, and implicitly solves the gesture classification problem by making the detection mutually exclusive. Some formalism is needed to bridge between the detection and classification views.

Our recorded training data form a long acceleration sequence of length T that intermittently contains gestures. Each gesture in the recorded data can be described as a *batch*, formally denoted as a triple

$$b = (t, d, c)$$

where $t \in \{1, \dots, T\}$ is the starting time of the batch, $d \in \{1, \dots, T\}$ is the duration of the batch, and $c \in \{1, \dots, M\}$ is the type of the gesture that the batch represents. It is these batches that are used to estimate the individual gesture HMMs (Section 4.4).

Validation is done by running the recorded data through the recognizer. Each recognized gesture can also be described as a batch triplet $b' = (t', d', c')$.²² We say that a recognized batch $b' = (t', d', c')$ *matches* a recorded batch $b = (t, d, c)$ if and only if

$$t \leq t' + \frac{d'}{2} < t + d,$$

that is, the midpoint of the recognized batch is inside the recorded batch.

Suppose b' and b match. The match is a *correct classification* if and only if $c' = c$. Otherwise it's a *misclassification*. A recorded batch with no matching recognized batch is said to be a *false negative*. A recognized batch with no matching recorded batch is said to be a *false positive*.²³

It is important to notice that this formalism results in a combined way to measure both classification and spotting performance from real recognition results. Cross validation can be done normally by introducing a validation set and a training set whose elements are the recorded batches. Anomalous batches (Section 5.3.1) are excluded from both training and validation sets.

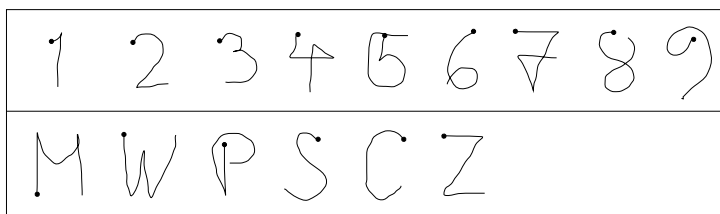


Figure 6.1: The set of gestures used in the experiments. The start points of the gestures are denoted with large dots. The gestures are drawn in the horizontal-vertical plane in front of the user.

6.2 Single-User Experiments

To measure the recognition performance, we used a vocabulary of 15 gestures seen in Figure 6.1. This vocabulary consists of nine digit gestures (1-9) and six letter gestures (M,W,P,S,C and Z). We used similar digit gestures as Cho *et al.* [3]. Ten repetitions of each gesture were recorded from a single user (author of the thesis). The parameter values used were:

- HMM states per gesture $N = 10$ (Section 4.4.1)
- garbage weight ratio $\omega_G = 0.25$ (Section 4.5.1)
- garbage mixture count $M_G = 4$ (Section 4.5.1)
- garbage self transition probability $\gamma = 0.95$ (Section 4.5.1)

6.2.1 Classification Performance

In the first experiment, we wanted to measure the classification rate, i.e., how large a percentage of recognized gestures were correct classifications. It turns out that this experiment can also be used to measure the false negative rate. One-out cross validation with the formalism introduced in Section 6.1.2 was used.

²²The starting time and the duration of the recognized gesture are found via the Viterbi algorithm using an unbounded window size.

²³In the terminology of Lee and Kim [18] misclassifications, false negatives and false positives are called substitution errors, deletion errors and insertion errors, respectively.

	1	2	3	4	5	6	7	8	9	M	W	P	S	C	Z	\emptyset
1	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10
2	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	90	0	0	0	0	0	0	0	0	0	0	10	0
5	0	0	0	0	90	0	0	0	10	0	0	0	0	0	0	0
6	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	80	0	0	0	0	20	0	0	0
9	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	90	0	0	0	10
S	0	0	0	0	0	0	0	20	0	0	0	0	70	10	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	90	10

Table 6.1: Confusion matrix. Values are reported in percentages.

The results are reported in Table 6.1 as a confusion matrix, formally denoted as

$$C = (c_{i,j}) \in \mathbb{R}^{M \times (M+1)}.$$

Rows denote the correct gesture classes, and columns denote the recognized gesture classes. Put precisely, each element $c_{i,j}$ is an estimate of the probability of classifying a gesture of class i into the gesture class j . Diagonal values denote correct classifications, and should ideally be close to 100%; off-diagonal values denote misclassifications, and should ideally be close to 0%.

We have augmented the confusion matrix with an extra column (symbol \emptyset) that tells how large a percentage of the gestures were false negatives, i.e., went unrecognized. This can be likened to a "null" gesture class. Notice that all rows of the matrix sum up to one.

We can extract three interesting statistics from the confusion matrix that we call the *recognition rate*

$$\frac{1}{M} \sum_{i=1}^M c_{i,i} \approx 93.3\%,$$

the *classification rate*

$$\frac{1}{M} \sum_{i=1}^M \frac{c_{i,i}}{1 - c_{i,M+1}} \approx 95.3\%,$$

and the *false negative rate*

$$\frac{1}{M} \sum_{i=1}^M c_{i,M+1} \approx 2.00\%.$$

Notice that the classification rate is essentially the recognition rate with the false negative cases removed: both can be interpreted as the trace of a right stochastic matrix.

Such good rates are not at all uncommon in the literature: practically all research groups have reported classification rates well over 90%. On the other hand, results are difficult to compare since different groups have used different vocabularies. Our fifteen-gesture vocabulary is among the largest, although Kallio *et al.* [12] experimented with a sixteen-gesture vocabulary, obtaining a classification rate of over 95% in the user-dependent case.

We can compare the false negative rate against previous research using sensors other than accelerometers. Nam and Wohn [23] reported rates of 0% to 3.6% depending on gesture type. Lee and Kim [18] and Yang *et al.* [36] obtained overall rates of $\frac{17}{420} \approx 4\%$ and $\frac{6}{602} \approx 1\%$, respectively. We find our false negative rate ($\approx 2.00\%$) comparable to these results.

6.2.2 Spotting Performance

A false positive occurs when the recognizer spots a gesture where there is none, for example, during a walk or during a meaningless hand movement. Unfortunately, measuring false positive rate isn't straightforward since there are potentially an infinite number of meaningless input data.

We propose a simple method of measuring the false positive rate using only the gestural training data, i.e., no extra garbage data is required. In this method, the recognizer is trained M times, each time for a single gesture type only. The recognizer is then validated against all *other* gesture types — ideally, the recognizer shouldn't then spot a single gesture. We use the formalism introduced in Section 6.1.2 to count false positives.

The results are reported in Table 6.2 as what we refer to as a *spotting matrix*, whose leftmost M columns are formally denoted as the matrix

$$S = (s_{i,j}) \in \mathbb{R}^{M \times M}.$$

	1	2	3	4	5	6	7	8	9	M	W	P	S	C	Z	Σ
1	-	0	0	0	0	20	10	0	0	30	0	0	0	0	0	4
2	0	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	-	0	60	0	0	0	10	0	0	0	0	0	0	5
4	0	0	0	-	0	0	0	0	0	0	20	0	0	0	0	1
5	0	90	10	30	-	0	0	0	0	0	0	20	0	0	10	11
6	0	0	0	0	0	-	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	-	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	-	0	0	0	0	100	0	0	7
9	0	10	0	0	60	0	0	0	-	0	0	0	0	0	0	5
M	0	0	0	0	0	0	0	0	0	-	10	0	0	10	0	1
W	0	0	0	0	0	0	0	0	0	0	-	0	0	0	0	0
P	0	0	0	0	10	0	0	10	0	0	20	-	0	0	0	3
S	0	0	0	0	0	0	0	80	0	0	0	0	-	0	0	6
C	0	0	0	0	0	40	0	10	0	0	0	0	30	-	0	6
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	0

Table 6.2: Spotting matrix. Values are reported in percentages.

The rows of S denote the gesture class used for training, and columns denote the recognized gesture classes. Put precisely, each element $s_{i,j}$ is an estimate of the probability of spotting a gesture of class j when the spotter was trained to recognize only gestures of class i . Notice that the diagonal values are meaningless.

The rightmost column of Table 6.2 (symbol Σ) consists of the averages of the corresponding rows of S .

Now we can extract the *false positive rate*

$$\frac{1}{M(M-1)} \sum_{i=1}^M \sum_{\substack{j=1 \\ j \neq i}}^M s_{i,j} \approx 3.33\%.$$

Since this validation method is new, meaningful comparisons to previous research are hard to make. Lee and Kim [18] report a false positive rate of $\frac{3}{420} \approx 0.71\%$, or $\frac{3+9}{420} \approx 2.86\%$ if misclassifications are counted as false positives. Yang *et al.* [36] obtained a rate of $\frac{14}{602} \approx 2.3\%$. We note that our result ($\approx 3.33\%$) is slightly higher.

Notice that the spotting matrix is very "spiky", mostly consisting of zeroes and a few very high percentages. Some of these spikes are symmetrical (e.g.,

between "8" and "S"), whereas others are non-symmetrical (e.g., between "2" and "5"). The confusion between pairs such as "8" and "S" is clearly caused by their related trajectories. Robust treatment of such related trajectories is a difficult problem left for future research.

6.3 Effect of Parameter Variations

Having measured the "baseline" single-user recognition performance, we are now interested in how the performance varies with the recognition parameters (including binary parameters, e.g., whether tilt compensation is used). For each analyzed parameter, we will perform the same experiments as in Section 6.2, and briefly report and discuss the results. Finally, a broader discussion is given.

Results are reported as tables of parameter values against classification rate (CR), false negative rate (FNR) and false positive rate (FPR). Referential parameter values used in the baseline single-user experiments are italicized for convenience; the best performances in each experiment are boldfaced.

6.3.1 Tilt Compensation

	<i>CR</i>	<i>FNR</i>	<i>FPR</i>
<i>with tilt compensation</i>	95.3%	2.00%	3.33%
without tilt compensation	96.3%	2.67%	2.90%

Table 6.3: Effect of tilt compensation on performance.

Table 6.3 indicates that tilt compensation (Section 4.3.4) has no huge effect on performance. This could be because training data were acquired from a single user only, without major variation in hand orientation.

6.3.2 Magnitudal Features

Table 6.4 indicates that using additional magnitudal features clearly improves spotting performance (FPR), as we hypothesized in Section 4.3.7. However, adding these features has a detrimental effect on CR and FNR.

	<i>CR</i>	<i>FNR</i>	<i>FPR</i>
<i>with magnitudal features</i>	95.3%	2.00%	3.33%
without magnitudal features	98.7%	0.667%	10.7%

Table 6.4: Effect of magnitudal features on performance.

6.3.3 Covariance Constraints

	<i>CR</i>	<i>FNR</i>	<i>FPR</i>
<i>no constraints</i>	95.3%	2.00%	3.33%
diagonal constraint	95.1%	2.67%	14.3%

Table 6.5: Effect of covariance constraints on performance.

Pylvänäinen [26] used multivariate Gaussians (Appendix A.2) to model HMM observations, but with the additional constraint that the covariance matrices must be diagonal. Table 6.5 indicates that while such a diagonal constraint doesn't affect CR and FNR much, it has a major detrimental effect on FPR.

6.3.4 HMM State Count

	<i>CR</i>	<i>FNR</i>	<i>FPR</i>
$N = 5$	93.0%	2.00%	24.0%
$N = 10$	95.3%	2.00%	3.33%
$N = 15$	98.0%	6.00%	1.33%

Table 6.6: Effect of HMM state count per gesture (N) on performance.

Table 6.6 indicates that increasing the number of HMM states used per gesture (Section 4.4.1) generally has a positive effect on FPR, but too high values increase FNR.

6.3.5 Garbage Weight Percentage

Table 6.7 indicates that a good choice of garbage weight ratio (Section 4.5.1) helps improve FPR. It is interesting to see that the optimal FPR is obtained at a non-zero ω_G — this suggests that using extra garbage data in the estimation of filler models is beneficial.

	<i>CR</i>	<i>FNR</i>	<i>FPR</i>
$\omega_G = 0.00$	92.5%	3.33%	7.24%
$\omega_G = 0.25$	95.3%	2.00%	3.33%
$\omega_G = 0.50$	94.8%	1.33%	4.67%

Table 6.7: Effect of garbage weight percentage (ω_G) on performance.

6.3.6 Garbage Mixture Count

	<i>CR</i>	<i>FNR</i>	<i>FPR</i>
$M_G = 1$	93.24%	2.66%	9.43%
$M_G = 4$	95.3%	2.00%	3.33%
$M_G = 16$	95.3%	6.67%	2.81%

Table 6.8: Effect of garbage mixture count (M_G) on performance.

Table 6.8 indicates that using a higher number of mixture components to model the garbage state (Section 4.5.1) helps improve FPR. However, too high a count increases FNR.

6.3.7 Garbage Self-Transition Probability

	<i>CR</i>	<i>FNR</i>	<i>FPR</i>
$\gamma = 0.05$	89.7%	3.33%	24.0%
$\gamma = 0.50$	95.6%	1.33%	6.29%
$\gamma = 0.95$	95.3%	2.00%	3.33%

Table 6.9: Effect of garbage self-transition probability (γ) on performance.

Table 6.9 indicates that the FPR is highly sensitive to the *ad hoc* garbage self-transition probability parameter (Section 4.5.1). Values closer to unity seem to give vastly better results.

6.3.8 Discussion

The false positive rate was found quite sensitive to parameter variations. However, we identified many parameter choices that helped lower FPR significantly.

Generally speaking, using more HMM states and mixture components improved FPR. Future work includes seeing whether using more than one garbage HMM state, alike Kim and Lee [18] and Yang *et al.* [36], helps improve FPR with less parameter tuning.

6.4 Runtime Performance

A usable gesture recognizer has to process its input at a suitable speed. In isolated recognizers an input acceleration sequence has to be classified into a gesture type very fast. For example, the user may get confused by a multi-second lag. In continuous recognition, efficiency constraints are even more concrete. If the recognizer cannot "keep up" with the continuously arriving sensor readings, then it simply *cannot* operate correctly. In other words, continuous gesture recognition is a form of *real-time computation problem*.

Since our gesture vocabularies are rather small — less than twenty gestures — runtime performance is suspected to be quite good. However, since we plan to run the recognizer on mobile phones, at least a rudimentary measurement of performance is warranted. Also, we present an early experiment on power consumption.

6.4.1 Real-Time Ratio

The simplest indicator of runtime performance is the real-time ratio, or the xRT [17]. It is defined as the ratio of time taken to process the signal (by the recognizer) to the time taken to produce the signal (by the sensor). The system satisfies the real-time constraint if and only if $xRT \leq 1$.

Lai *et al.* [17] measured the xRT of a large vocabulary continuous speech recognizer (LVCSR) under varying conditions using the commercial VTune profiler by Intel. We measured the xRT of our system on three platforms — namely, a desktop PC (with an Intel Core 2 Duo 2.40 GHz processor), Nokia 5500 Sport, and Nokia N95 — using the same 15-gesture vocabulary and parameters as in the single-user experiments. We used a nanosecond resolution timer on PC, and a millisecond resolution timer on mobile phones.

Our results are summarized in Table 6.10. We see that the xRT is clearly less

<i>Platform</i>	<i>xRT</i>
PC	0.0034
Nokia 5500 Sport	0.34
Nokia N95	0.36

Table 6.10: Real-time ratio (xRT) on three different platforms.

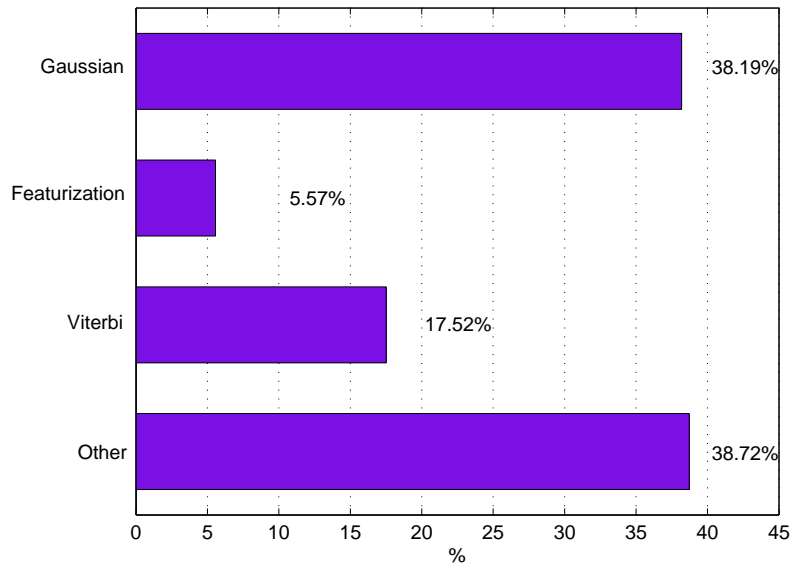


Figure 6.2: Time distribution of the recognizer. Values are reported in percentages.

than one on all three platforms — therefore, continuous gesture recognition using filler models is viable on mobile devices, at least in the absence of other processor-intensive tasks.

6.4.2 Time Distribution

To study how time is spent in various components of the recognizer, we used the profiler in the Wireless Toolkit by Sun Microsystems. The test was performed on the Wireless Toolkit phone emulator, using the same 15-gesture vocabulary and parameters as in the single-user experiments.

Results are shown in Figure 6.2. Evaluating Gaussian densities for the 5-

dimensional observation vectors took 38.19% of the time, even though these computations were heavily optimized (Appendix A.2). Gaussian computations were also a major bottleneck in the performance evaluation of Lai *et al.* [17]. Computations related to the Viterbi algorithm, which included keeping track of the maximum likelihood state sequence and searching for gestures from the said sequence, took 17.52%.

There is still a considerable 38.72% overhead caused by miscellaneous computations, most notably the logic for handling multiple sensors. This suggests that xRT can be decreased by simple optimizations. Featurization took only 5.57%, but on the other hand, its absolute time is independent of the vocabulary size: for smaller vocabularies, it becomes relatively more expensive.

6.4.3 Power Consumption

Performing computations consumes power. This is an important consideration when the system is run on a battery-powered mobile phone. In this context, power consumption shouldn't be ignored as a mere engineering issue: an overly high power consumption indicates the need for further research on more efficient algorithms, just like a high xRT does.

Here we study the effect of continuous gesture recognition on power consumption. The experiments were performed on a Nokia 5500 Sport using a vocabulary of 16 gestures.²⁴ We are especially interested in how the number of *active* gestures (Section 4.6.2) affects the power consumption.

<i>Active gesture count</i>	<i>Battery time (hours)</i>
0	28
1	27
16	15

Table 6.11: Battery times with varying active gesture counts.

All experiments were performed by fully charging the battery and then running the recognizer until battery exhaustion. The phone was set in offline mode to eliminate all network communication. Battery times are reported in Table 6.11. The times ranged from 15 to 28 hours. For comparison, simply

²⁴This 16-gesture vocabulary was different from the 15-gesture one used in the single-user experiments. This experiment was conducted by my coworker Tuomas Inkeroinen.

reading the accelerometer data resulted in a battery life of 92 hours. In full standby mode the battery life was 456 hours.

Clearly, continuous recognition introduces a notable drop in battery time even for a small number of active gestures. This highlights a need for optimizations to ensure acceptable battery times. Also, the difference between power consumption of Java and Symbian C++ based recognizers should be studied for completeness. These are left as future work.

7 Conclusion and Perspectives

In this thesis we have studied the application of filler models — a type of hidden Markov models — to the problem of recognizing keyword gestures from accelerometer data. A Java-based implementation running portably on both desktops and mobile phones was presented, and evaluated with respect to recognition performance and runtime performance. While the speed was found viable for mobile phones, power consumption issues require more work. Recognition performance was generally comparable to previous results, but false positive rates are still quite sensitive to parameter choices. However, interesting ideas for lowering false positive rates were identified in the experiments, notably using additional magnitudal features and garbage training data. Future work includes trying out more expressive filler models having multiple garbage states. Also, we feel that the speech recognition literature hasn't been "exploited" in gesture recognition too well yet; for example, gain-independent autoregressive models [27] could be used to compensate for amplitude variations, especially in continuous recognition.

So far, the vocabularies used in gesture recognition have been rather small: less than twenty gestures in all accelerometer-based studies that we're aware of. In speech recognition, much larger vocabularies are used — for example, in their evaluation, Lai *et al.* [17] used a continuous speech recognizer with 51000 words! Pylvänäinen [26] is more skeptical about the inclusion of large vocabularies to gesture recognition, since people have a hard time remembering even a small amount of gestures. There just don't seem to be as many combinations of hand movements as there are spoken sounds. On the other hand, using widely understood gesture trajectories such as digits and letters should alleviate the memorization problem. For example, Cho *et al.* [3] performed a handwriting study to discover the most common trajectories for digits, and used the results to build a working user-independent gesture recognizer.

One problem with current research in accelerometer based recognition systems is that results from different research groups are difficult to compare. Traditionally, each team has implemented their own algorithms and recorded their own data. Reimplementing algorithms for comparison's sake can be a truly gargantuan task; however, expecting all groups to release their algorithms as open source may not be realistic. On the other hand, simply publishing the recorded data should be enough to facilitate sound evaluation between different algorithms.

Appendix A Common Observation Distribution Families

In this appendix, we will review some common probability distribution families used for observation variables in the HMM, and the corresponding parameter estimation formulae for these distributions.

Let Y denote the observation random variable parametrized by θ . Let Y_1, \dots, Y_N be a random sample of Y (i.e., the Y_1, \dots, Y_N are independent and distributed identically to Y). Let *training samples* y_1, \dots, y_N be a realization of this random sample.

The *maximum likelihood estimate* [34, pp. 122-124, Section 9.3] of the parameter θ is

$$\hat{\theta}^{\text{ML}}(y_1, \dots, y_N) = \operatorname{argmax}_{\theta} \prod_{i=1}^N f_Y(y_i; \theta)$$

or equivalently

$$\hat{\theta}^{\text{ML}}(y_1, \dots, y_N) = \operatorname{argmax}_{\theta} \sum_{i=1}^N \log f_Y(y_i; \theta).$$

Maximum likelihood estimates have a simple statistical interpretation: they maximize the likelihood of the distribution having generated the training samples in the given distribution family.

Often, the training samples are augmented with corresponding non-negative weights w_1, \dots, w_N , of which at least one must be strictly positive. We define the *maximum pseudo likelihood estimate* as

$$\hat{\theta}^{\text{MPL}}(y_1, \dots, y_N, w_1, \dots, w_N) = \operatorname{argmax}_{\theta} \sum_{i=1}^N w_i \log f_Y(y_i; \theta).$$

Maximum pseudo likelihood estimates have no straightforward statistical interpretation, but are nevertheless useful and, as we shall see, often appear in applications of the Expectation-Maximization algorithm.

A.1 Univariate Normal Distribution

Among the simplest and most common distribution families for univariate continuous random variables is the family of normal (or Gaussian) distributions. A normal distribution is parametrized by its mean $\mu \in \mathbb{R}$ and its covariance $\sigma \in \mathbb{R}, \sigma > 0$. We use $Y \sim N(\mu, \sigma^2)$ to denote that a random variable Y is normally distributed with mean μ and covariance σ .

The density function of $Y \sim N(\mu, \sigma^2)$ is [34, pp. 23-29]

$$f_Y(y; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}.$$

Given N training samples y_1, \dots, y_N with corresponding weights w_1, \dots, w_N , the maximum pseudo likelihood estimates for μ, σ are

$$\hat{\mu}^{\text{MPL}}(y_1, \dots, y_N, w_1, \dots, w_N) = \frac{\sum_{i=1}^N w_i y_i}{\sum_{i=1}^N w_i}$$

and

$$\hat{\sigma}^{\text{MPL}}(y_1, \dots, y_N, w_1, \dots, w_N) = \sqrt{\frac{\sum_{i=1}^N w_i (y_i - \hat{\mu}^{\text{MPL}})^2}{\sum_{i=1}^N w_i}}.$$

A.2 Multivariate Normal Distribution

The normal distribution generalizes well into the multivariate case [34, pp. 234-235, Section 14.3]. We use $Y \sim N_d(\mu, \Sigma)$ to denote that Y is a d -dimensional normally distributed multivariate random variable²⁵ with a mean vector $\mu \in \mathbb{R}^d$ and a positive definite covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$.

The density function of $Y \sim N_d(\mu, \Sigma)$ is

$$f_Y(y; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} \sqrt{\det \Sigma}} e^{-\frac{1}{2} r_M^2},$$

²⁵In what follows, we always assume that multivariate random variables are random *column* vectors. With this convention, the inner product $Y^T Y$ is a univariate random variable whereas the outer product $Y Y^T$ is a random square matrix.

where

$$r_M^2 = (y - \mu)^\top \Sigma^{-1} (y - \mu)$$

is the (squared) Mahalanobis distance [34, p. 353, equation 22.11]. In applications where we need to evaluate density functions for a potentially high number of distributions the Mahalanobis distance may become a computational bottleneck. Notice that since Σ is symmetric and positive definite, its inverse Σ^{-1} has the Cholesky decomposition

$$\Sigma^{-1} = U^\top U,$$

where $U = (u_{ij}) \in \mathbb{R}^{d \times d}$ is an upper triangular matrix. This leads to a highly efficient formula for computing the Mahalanobis distance, namely

$$\begin{aligned} r_M^2 &= (y - \mu)^\top U^\top U (y - \mu) \\ &= (U(y - \mu))^\top (U(y - \mu)) \\ &= \sum_{i=1}^d \left(\sum_{j=i}^d u_{ij} (y_j - \mu_j) \right)^2 \\ &= \sum_{i=1}^d \left(c_i + \sum_{j=i}^d u_{ij} y_j \right)^2, \end{aligned}$$

where

$$c_i = - \sum_{j=i}^n u_{ij} \mu_j,$$

which amounts to $\frac{d(d+3)}{2}$ additions and multiplications when the values u_{ij} and c_i are precomputed.

Given N training samples y_1, \dots, y_N , the maximum pseudo likelihood estimates for μ, Σ are

$$\hat{\mu}^{\text{MPL}}(y_1, \dots, y_N, w_1, \dots, w_N) = \frac{\sum_{i=1}^N w_i y_i}{\sum_{i=1}^N w_i}$$

and

$$\hat{\Sigma}^{\text{MPL}}(y_1, \dots, y_N, w_1, \dots, w_N) = \frac{\sum_{i=1}^N w_i (y_i - \hat{\mu}^{\text{MPL}}) (y_i - \hat{\mu}^{\text{MPL}})^\top}{\sum_{i=1}^N w_i}.$$

A.3 Mixture Distribution

The normal distribution families of the preceding sections have some deficiencies for practical modeling.

First, normal distributions are *unimodal*, that is, their density functions have only a single local maximum (at the mean). In many applications, this may not be a realistic assumption. For example, even if the heights of individuals in two populations (say, male Filipinos and male Dinaric Alpines) were approximately unimodally distributed, the heights of individuals in the *combined* population may not be.

Second, the normal distribution doesn't have control over its *higher-order moments* such as skewness or kurtosis — these are entirely determined by the mean and the (co)variance. There are other continuous probability distribution families, such as exponential and gamma distributions, that have different skewness and kurtosis profiles from the normal distribution, but they are hard to generalize to the multivariate case.

The mixture distribution offers a general solution to these problems by modeling the density as a convex combination of component densities. The density function of a mixture distributed random variable Y is

$$f_Y(y; \theta) = \sum_{i=1}^M \alpha_i f_{Z_i}(y; \theta_i),$$

where the constant M is the number of mixture components and Z_1, \dots, Z_M are the random variables distributed according to the component densities. The $\alpha_1, \dots, \alpha_M$ are non-negative mixture weights such that $\sum_{i=1}^M \alpha_i = 1$, and $\theta_1, \dots, \theta_M$ are the parameters of the component densities. The mixture density is then parametrized by the $2M$ -tuple $\theta = (\alpha_1, \dots, \alpha_M, \theta_1, \dots, \theta_M)$.

A.3.1 Parameter Estimation

Unfortunately, there exists no general closed-form solution for maximum (pseudo) likelihood estimation for mixture distributions, even if the component distribution families were easily estimable. Rather, numerical methods must be employed. We use the Expectation-Maximization algorithm [2], which begins with an initial parameter estimate $\hat{\theta}^0$ and uses a reestimation

procedure to obtain improved estimates $\hat{\theta}^1, \hat{\theta}^2, \dots$, converging to a stationary point. The full derivation is tedious [2], so we present only the final formulae here.²⁶

On each iteration $t = 0, 1, \dots$ we first compute the *partial membership values*

$$\gamma_{i,j}^t = \frac{\alpha_i^t f_{Z_i}(y_j; \hat{\theta}_i^t)}{f_Y(y_j; \hat{\theta}^t)}.$$

Each partial membership value $\gamma_{i,j}^t$ represents the "degree of belief" that the j th training sample y_j was "generated" by the i th component distribution.

The updated mixture weights in the non-weighted case (maximum likelihood estimation) are then given by

$$\alpha_i^{t+1} = \frac{1}{N} \sum_{j=1}^N \gamma_{i,j}^t$$

and the updated component density parameters by

$$\hat{\theta}_i^{t+1} = \hat{\theta}_i^{\text{MPL}}(y_1, \dots, y_N, \gamma_{i,1}^t, \dots, \gamma_{i,N}^t),$$

that is, by employing maximum pseudo likelihood estimation using the partial membership values as weights.

In the weighted case (maximum pseudo likelihood estimation) we insert the weights w_1, \dots, w_N into the update equations straightforwardly:

$$\alpha_i^{t+1} = \frac{\sum_{j=1}^N w_j \gamma_{i,j}^t}{\sum_{j=1}^N w_j}$$

and

$$\hat{\theta}_i^{t+1} = \hat{\theta}_i^{\text{MPL}}(y_1, \dots, y_N, w_1 \gamma_{i,1}^t, \dots, w_N \gamma_{i,N}^t).$$

We stop the iteration once the relative difference between the pseudo or log-likelihoods of $\hat{\theta}^t$ and $\hat{\theta}^{t+1}$ is less than a given threshold.

²⁶The underpinnings of the Expectation-Maximization algorithm will be explored further in the main text (Section 3.5.2).

A.4 Categorical Distribution

The categorical distribution is the most general discrete distribution family when the number of outcomes is finite. Without loss of generality, we assume that the observation Y can have outcomes $1, \dots, K$, where $K \in \mathbb{N}$.

A categorical distribution is then parametrized by the K -dimensional vector $\theta = (p_1, \dots, p_K)$ whose components p_1, \dots, p_K are non-negative and sum up to one. The probability measure of the categorical distribution is defined as

$$\Pr[Y = y; \theta] = p_y.$$

The maximum pseudo likelihood estimates for the parameter components p_1, \dots, p_K are

$$\hat{p}_i^{\text{MPL}}(y_1, \dots, y_N, w_1, \dots, w_N) = \frac{\sum_{j=1}^N w_j \delta(y_j, i)}{\sum_{j=1}^N w_j}.$$

A.4.1 Vector Quantization

In physical recognition systems, the input is usually inherently continuous, e.g., physical acceleration, sound wave pressure, or spectral vectors. For the categorical distribution to be useful in these applications, we need some mechanism for transforming vectors from a continuous vector space V to a finite, K -symbol alphabet.

Vector quantization [28, pp. 122-131] is a general name for such transformations. The transforming function $g : V \rightarrow \{1, \dots, K\}$ is also called the *codebook*, since vector quantization is an example of source coding [28, pp. 244-264] where continuous data are digitized for transmissibility.

The canonical way to construct the function g is to select K *prototype vectors* $v_1, \dots, v_K \in V$ and set

$$g(v) = \underset{i \in \{1, \dots, K\}}{\operatorname{argmin}} d(v, v_i), \quad (\text{A.1})$$

where d is some distance metric (e.g., Euclidean distance). Estimating the prototype vectors from training data is an example of *clustering*. A popular clustering method is the K-means algorithm [28, pp. 125-129].

Evaluating the minimization in equation (A.1) can be costly for large values of K , but vector quantization can still incur significant computational savings if the codebook is *shared* with the observation distributions. In this case the cost of the single minimization is amortized by the fact that evaluating the underlying categorical densities requires just table lookups.

On the other hand, categorical distributions with vector quantization can be viewed as a special case of mixture distribution (Appendix A.3) where each component distribution X_1, \dots, X_K is uniformly distributed in a subset of V :

$$X_i \sim U(V_i), \quad i = 1, \dots, K,$$

where

$$V_i = \{ v \in V \mid g(v) = i \}.$$

Therefore, this "sharing trick" should generalize to any mixture distribution. In fact, the semi-continuous HMM of Huang and Jack [9] uses this kind of tied observation densities, and they explicitly note its equivalence to the mixture distribution.

Appendix B Logarithm Transformations

To ensure numerical robustness, it's often a good idea to carry computations under logarithm transformation. Let f_1, \dots, f_N be a set of non-negative values (for example, densities). Let l_1, \dots, l_N denote the logarithm transformation of each corresponding f_1, \dots, f_N , i.e.,

$$l_i = \log f_i, \quad i = 1, \dots, N,$$

where \log refers to the natural logarithm. We extend the logarithm and exponential functions to handle the case of zero and negative infinity (respectively) as

$$\log 0 = -\infty$$

and

$$\exp -\infty = 0.$$

We note that this extension doesn't affect the logarithmic identities employed in the following subsections.

B.1 The Exponential Function

Suppose that $f_i = e^x$ for some i and x . Then the corresponding logarithm transformation is simply $l_i = x$.

The exponential function e^x becomes susceptible to numerical underflow as $x < 0$ grows in magnitude.²⁷ In our experiments we found that e^x is numerically equivalent to zero starting from $x \approx -745$ for double-precision and $x \approx -104$ for single-precision IEEE floating point numbers. The logarithm transformation l_i is essentially free of such problems.

²⁷Analogously, e^x becomes susceptible to numerical overflow if $x > 0$ grows in magnitude. In statistical applications, the underflow problem is usually more prevalent, but of course logarithmization helps with the overflow problem as well.

B.2 Products and Divisions

From the logarithmic identities we have

$$\log \prod_{i=1}^N f_i = \sum_{i=1}^N \log f_i = \sum_{i=1}^N l_i$$

and

$$\log \frac{f_i}{f_j} = \log f_i - \log f_j = l_i - l_j.$$

For example, suppose $f_i \equiv f$ for all i . Then we have

$$\prod_{i=1}^N f_i = f^N.$$

Again, as N grows, the exponential f^N becomes susceptible to underflow and overflow. The logarithm transformation $N \log f$ changes linearly with N and is therefore much more robust.

B.3 Summation

Suppose we need to evaluate a sum of densities

$$f \equiv \sum_{i=1}^N f_i.$$

The corresponding logarithm transformation is

$$l \equiv \log f = \log \sum_{i=1}^N f_i = \log \sum_{i=1}^N e^{l_i}.$$

The problem is that given only the values l_1, \dots, l_N , computing the terms e^{l_i} might result in underflow as we saw earlier in Section B.1. However, for every $c \in \mathbb{R}$ we have

$$l = c + \log \sum_{i=1}^N e^{l_i - c}. \tag{B.1}$$

By choosing the value c judiciously, we can ensure that the terms e^{l_i-c} are less susceptible to underflow. We have found that the value $c = \max_i l_i$ gives robust results.

For example, Mann [20] uses the same trick to define a binary addition operator

$$\log(e^{l_i} + e^{l_j}) = \begin{cases} l_i + \log(1 + e^{l_j-l_i}), & \text{if } l_i \geq l_j, \\ l_j + \log(1 + e^{l_i-l_j}), & \text{if } l_j \geq l_i. \end{cases}$$

A sum of N terms can be written as a series of $N - 1$ binary additions, so we have to perform $\Theta(N)$ logarithm and $\Theta(N)$ exponential operations. By using equation (B.1) directly we only need to perform a single logarithm operation.

B.3.1 Weighted Summation

Consider a weighted sum

$$f \equiv \sum_{i=1}^N w_i f_i,$$

where w_1, \dots, w_N are non-negative weights. The logarithm transformation is

$$\begin{aligned} l \equiv \log f &= \log \sum_{i=1}^N w_i f_i \\ &= \log \sum_{i=1}^N e^{\log w_i + \log f_i} \\ &= \log \sum_{i=1}^N e^{\log w_i + l_i} \\ &= \log \sum_{i=1}^N e^{l'_i}, \end{aligned}$$

where $l'_i = \log w_i + l_i$. This reduces the problem of weighted summation to the problem of non-weighted summation with modified logarithmized input values l'_1, \dots, l'_N .

References

- [1] J. F. BARTLETT, *Rock 'n' Scroll is here to stay*, IEEE Computer Graphics Applications, 20 (2000), pp. 40–45.
- [2] J. A. BILMES, *A gentle tutorial of the EM algorithm and its applications to parameter estimation for Gaussian mixture and hidden Markov models*, Tech. Report ICSI-TR-97-021, University of Berkeley, 1997.
- [3] S.-J. CHO, E. CHOI, W.-C. BANG, J. YANG, J. SOHN, D. Y. KIM, Y.-B. LEE, AND S. KIM, *Two-stage recognition of raw acceleration signals for 3D gesture-understanding cell phones*, in Tenth International Workshop on Frontiers in Handwriting Recognition, 2006.
- [4] E.-S. CHOI, W.-C. BANG, J. YANG, D.-Y. KIM, AND S.-R. KIM, *Beatbox music phone: Gesture-based interactive mobile phone using a tri-axis accelerometer*, in Proceedings of the IEEE International Conference on Industrial Technology (ICIT), 2005, pp. 97–102.
- [5] J.-M. DAUTELLE, *Javolution*, 2007.
- [6] X. GE AND P. SMYTH, *Deformable Markov model templates for time-series pattern matching*, in Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD), New York, NY, USA, 2000, ACM, pp. 81–90.
- [7] ———, *Segmental semi-Markov models for endpoint detection in plasma etching*, IEEE Transactions on Semiconductor Engineering, (2001).
- [8] Y. HORRY, I. NAKAJIMA, T. HOSHINO, AND Y. MARUYAMA, *A passive-style buttonless mobile terminal*, IEEE Transactions on Consumer Electronics, 49 (2003), pp. 530–535.
- [9] X. D. HUANG AND M. A. JACK, *Hidden Markov modelling of speech based on a semicontinuous model*, Electronics Letters, 24 (1988), pp. 6–7.
- [10] JAVA COMMUNITY PROCESS, *JSR-256: Mobile Sensor API*, 2007.
- [11] B.-H. JUANG, S. E. LEVINSON, AND M. M. SONDHI, *Maximum likelihood estimation for multivariate mixture observations of Markov chains*, IEEE Transactions on Information Theory, 32 (1986), pp. 307 – 309.

- [12] S. KALLIO, J. KELA, AND J. MÄNTYJÄRVI, *Online gesture recognition system for mobile interaction*, in Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, vol. 3, October 2003, pp. 2070–2076.
- [13] S. KALLIO, J. KELA, J. MÄNTYJÄRVI, AND J. PLOMP, *Visualization of hand gestures for pervasive computing environments*, in Proceedings of the Working Conference on Advanced Visual Interfaces (AVI), New York, NY, USA, 2006, ACM, pp. 480–483.
- [14] M. KAUPPILA, *A tutorial on segmental semi-Markov models*. Available online, 2007.
- [15] M. KAUPPILA, S. PIIRTIKANGAS, X. SU, AND J. RIEKKI, *Accelerometer based gestural control of browser applications*, in International Workshop on Real Field Identification (RFId), Tokyo Denki University, Tokyo, Japan, 2007, pp. 2–17.
- [16] W.-G. KIM, J.-Y. CHOI, AND D.-H. YOUN, *HMM with global path constraint in Viterbi decoding for isolated word recognition*, in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 1994, pp. 605–608.
- [17] C. LAI, S.-L. LU, AND Q. ZHAO, *Performance analysis of speech recognition software*, in Proceedings of the Fifth Workshop on Computer Architecture Evaluation using Commercial Workloads, 2002.
- [18] H.-K. LEE AND J. H. KIM, *An HMM-based threshold model approach for gesture recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 21 (1999), pp. 961–973.
- [19] F. MAHMOUDI AND M. PARVIZ, *Visual hand tracking algorithms*, in Proceedings of the Geometric Modeling and Imaging (GMAI), 2006, pp. 228–232.
- [20] T. P. MANN, *Numerically stable hidden Markov model implementation*. Available online, 2006.
- [21] J. MÄNTYJÄRVI, P. KORPIPÄÄ, AND S. KALLIO, *Enabling fast and effortless customisation in accelerometer based gesture interaction*, in Proceedings of the 3rd International Conference on Mobile and Ubiquitous Media (MUM), New York, NY, USA, 2004, ACM, pp. 25–31.

- [22] V.-M. MÄNTYLÄ, J. MÄNTYJÄRVI, T. SEPPÄNEN, AND E. TUULARI, *Hand gesture recognition of a mobile device user*, in Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), 2000, pp. 281–284.
- [23] Y. NAM AND K. WOHN, *Recognition of space-time hand-gestures using hidden Markov model*, in ACM Symposium on Virtual Reality Software and Technology, 1996, pp. 51–58.
- [24] J. NEIDER AND T. DAVIS, *OpenGL programming guide: The official guide to learning OpenGL, Release 1*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.
- [25] V. S. N. PRASAD, V. KELLOKUMPU, AND L. S. DAVIS, *Ballistic hand movements*, in Articulated Motion and Deformable Objects (AMDO), Proceedings, Lecture Notes in Computer Science 4096, 2006, pp. 153–164.
- [26] T. PYLVÄNÄINEN, *Accelerometer based gesture recognition using continuous HMMs*, in Proceedings of the Second Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA), 2005, pp. 639–646.
- [27] L. R. RABINER, *A tutorial on hidden Markov models and selected applications in speech recognition*, vol. 77, 1989, pp. 257–286.
- [28] L. R. RABINER AND B.-H. JUANG, *Fundamentals of speech recognition*, Signal Processing, Prentice Hall, Inc., Upper Saddle River, NJ, USA, 1993.
- [29] J. REKIMOTO, *Tilting operations for small screen interfaces*, in Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology (UIST), ACM Press, 1996, pp. 167–168.
- [30] S. J. RUSSELL AND P. NORVIG, *Artificial Intelligence: A Modern Approach (International Edition)*, Pearson Education, 2003.
- [31] T. STARNER AND A. PENTLAND, *Visual recognition of American sign language using hidden Markov models*, in Proceedings of the International Workshop on Automatic Face And Gesture Recognition (IWAfGR), Zürich, Switzerland, 1995, pp. 189–194.
- [32] T. STARNER, J. WEAVER, AND A. PENTLAND, *Real-time American sign language recognition using desk and wearable computer based video*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 20 (1998), pp. 1371–1375.

- [33] C.-W. TAN AND S. PARK, *Design of accelerometer-based inertial navigation systems*, 54 (2005), pp. 2520–2530.
- [34] L. WASSERMAN, *All of statistics: A concise course in statistical inference*, Springer, second ed., 2004.
- [35] L. D. WILCOX AND M. A. BUSH, *Training and search algorithms for an interactive wordspotting system*, in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), vol. 2, 1992, pp. 97 – 100.
- [36] H.-D. YANG, A.-Y. PARK, AND S.-W. LEE, *Robust spotting of key gestures from whole body motion sequence*, in Proceedings of the 7th International Conference on Automatic Face and Gesture Recognition (FGR), Washington, DC, USA, 2006, IEEE Computer Society, pp. 231–236.