Babeş-Bolyai University of Cluj-Napoca, Romania
Faculty of Mathematics and Computer Science

Bachelor's Thesis in Computer Science

# Motion-based Gesture Recognition with an Accelerometer

Advisor
Assoc. Prof. PhD. Simona Motogna
motogna@cs.ubbcluj.ro

Author
Paul-Valentin Borza
paulvalentin@borza.ro

July 3, 2008

Universitatea Babeş-Bolyai din Cluj-Napoca, România
Facultatea de Matematică și Informatică

Lucrare de Diplomă

# Recunoaștere de Gesturi pe bază de Mișcare cu un Accelerometru

Coordonator
Conf. Dr. Simona Motogna
motogna@cs.ubbcluj.ro

Absolvent
Paul-Valentin Borza
paulvalentin@borza.ro

iulie 3, 2008

# Table of Contents

# Preface

I have conceived and demoed an initial version of the solution at the Imagine Cup 2007 Worldwide Finals in Seoul – South Korea, where I have qualified and represented Romania together with the 921UBB Team (Assoc. Prof. PhD. Simona Motogna, Daniel Ghiță, Mihai Dan Nadăș, Ovidiu Sabou, and me).

I have also participated with this project in the Google Summer of Code 2008 and won the Accelerometer-based Gestures proposal for OpenMoko Inc. Therefore, I'm currently involved in GSoC 2008 with OpenMoko Inc.

If you would like to read more on how the project evolves, please visit http://gestures.borza.ro. You should also check http://gestures.projects.openmoko.org to see how you can check-out the latest development version via Subversion. To read my proposal, please continue to http://code.google.com/soc/2008/openmoko/appinfo.html?csaid=55BC4E4D615BA291.

I hope that this paper will help interested people understand how acceleration should be modeled with hidden Markov models, and understand how to enable isolated and continuous recognition with such a system. I'm currently working on accelerometer-based gestures for the Neo FreeRunner (i.e. the second generation of mobile phones on which OpenMoko runs) that embeds two accelerometers. Most articles that are available on the Web describe their solution only in general terms and don't provide source code; this isn't the case, as the code I've written in C99 is available as open source under LGPL.

I wanted to incorporate the work I have done in a commercial product, and not just write the thesis and eventually forget about it. I believe I have succeeded and not only that it will be deployed on the Neo FreeRunner, it will also change the way we are used to interact with mobile devices.

Paul-Valentin Borza
Cluj-Napoca, Romania

# Chapter 1. Introduction

This paper assumes that you have some theoretical knowledge with probability, and statistics, but you're unfamiliar with the hidden Markov models subject. When I started studying continuous density hidden Markov models, I had no clue how such statistical models work. I was later amazed how well many problems can be modeled and solved with hidden Markov models. Some use them with speech (1); state-of-the-art speech recognizers use hidden Markov models.

More and more devices incorporate sensors like accelerometers and gyroscopes that enable us to measure acceleration and respectively orientation. We should be excited as we could add cool new features for mobile phones like auto screen orientation, or more… we could understand how people work and use their phone: we could detect and react to certain motion gestures and ease their lives. As great as it sounds, recognizing gestures is not a trivial task and requires theoretical and applied knowledge with probability and statistics. For the purposes of this demonstration, we will only work with the data provided by the 3-Axis ±3G accelerometer that resides inside the Nintendo Wii Remote.

This paper explains how a unique experience can easily be enabled on our mobile devices. The entire mechanism is presented. Implementation was entirely done in C99.

Here is an overall look of this thesis:

Chapter 1. Introduction: Basic assumptions used throughout this thesis and expressive photos that prove the applicability of such an innovative solution. Chapter 2. Hardware: We show how the axes are positioned inside a Nintendo Wii Remote and how to achieve communication over Bluetooth. Chapter 3. Architecture: We detail the basic components of such a recognition system. Chapter 4. Signal Processing: Applicable salient features for the end-point detector implemented as a two-class Gaussian classifier. Chapter 5. Use of Hidden Markov Models in Speech: Where were hidden Markov models first used and a case study of Microsoft Whisper. Chapter 6. Alternatives: We present the dynamic time warping alternative method versus the hidden Markov models. Chapter 7. Hidden Markov Models: We enumerate the mathematical characteristics of continuous density hidden Markov models. We model the theoretical left-to-right continuous density hidden Markov models on our practical problem. We use the trivariate normal distribution as the Gaussian distribution. Chapter 8. Implementation Issues and Limitations: We overcome some common problems like underflow with scaling, and logarithms; we also use deleted interpolation and impose parameter constraints. Chapter 9. Usage and Scenarios: We use the Nintendo Wii Remote to control Amarok with a small-vocabulary of predefined gestures; we also show how to create and use new gestures. Chapter 10. Future Work and Conclusions: Future work will be achieved within Google Summer of Code 2008 and OpenMoko Inc. We discuss more techniques to achieve a better recognition rate.

Before we jump into details, let's take a look at how we can make a change in future mobile phones, and remote controls.

# Vision: Mobile Phones

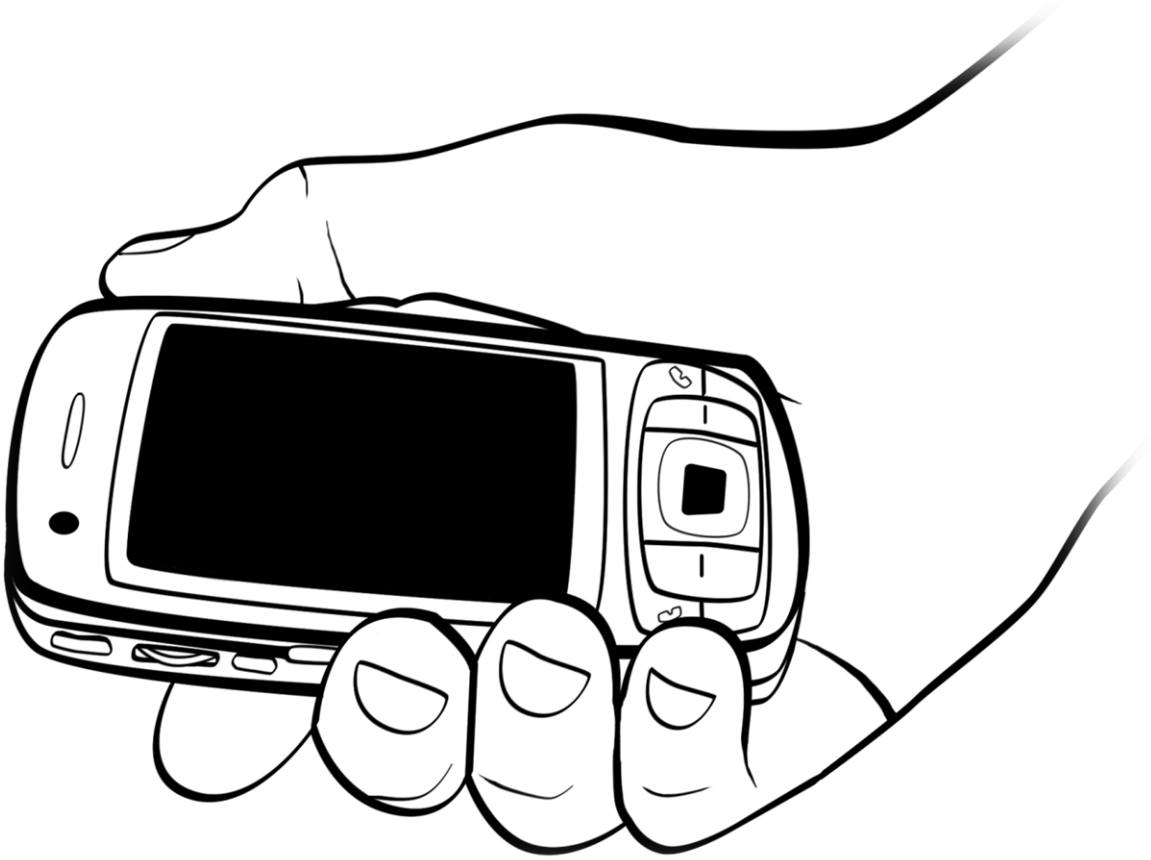

Figure 1 Human holds a mobile phone in his hand (photo made by Manuela Borza)

Here's a simple scenario that you've probably performed today. Your phone is sitting on the table and someone calls you – your phone is ringing. You pick it up half the way to look who's calling you. You push the green answer button and take the phone to your ear to talk. Don't you feel you're doing something extra that you shouldn't be doing?

Why do you press the answer button? The phone should know when to automatically answer the call because you've taken the phone to your ear. You basically do two things and one is extra. When you push the answer button, you let the phone know that you intend to talk with the other person. But wait! You're already making the gesture that you intend to answer the call by moving your phone to your ear. So why do you still need to push the answer button? This paper tries to improve the mobile experience such that you'll never have to push the answer button again!

Figure 2 shows six gestures done with a mobile phone and associates them with some possible actions, depending on the context in which the mobile phone is. Some of the presented gestures were proposed by the OpenMoko community for the Neo FreeRunner.

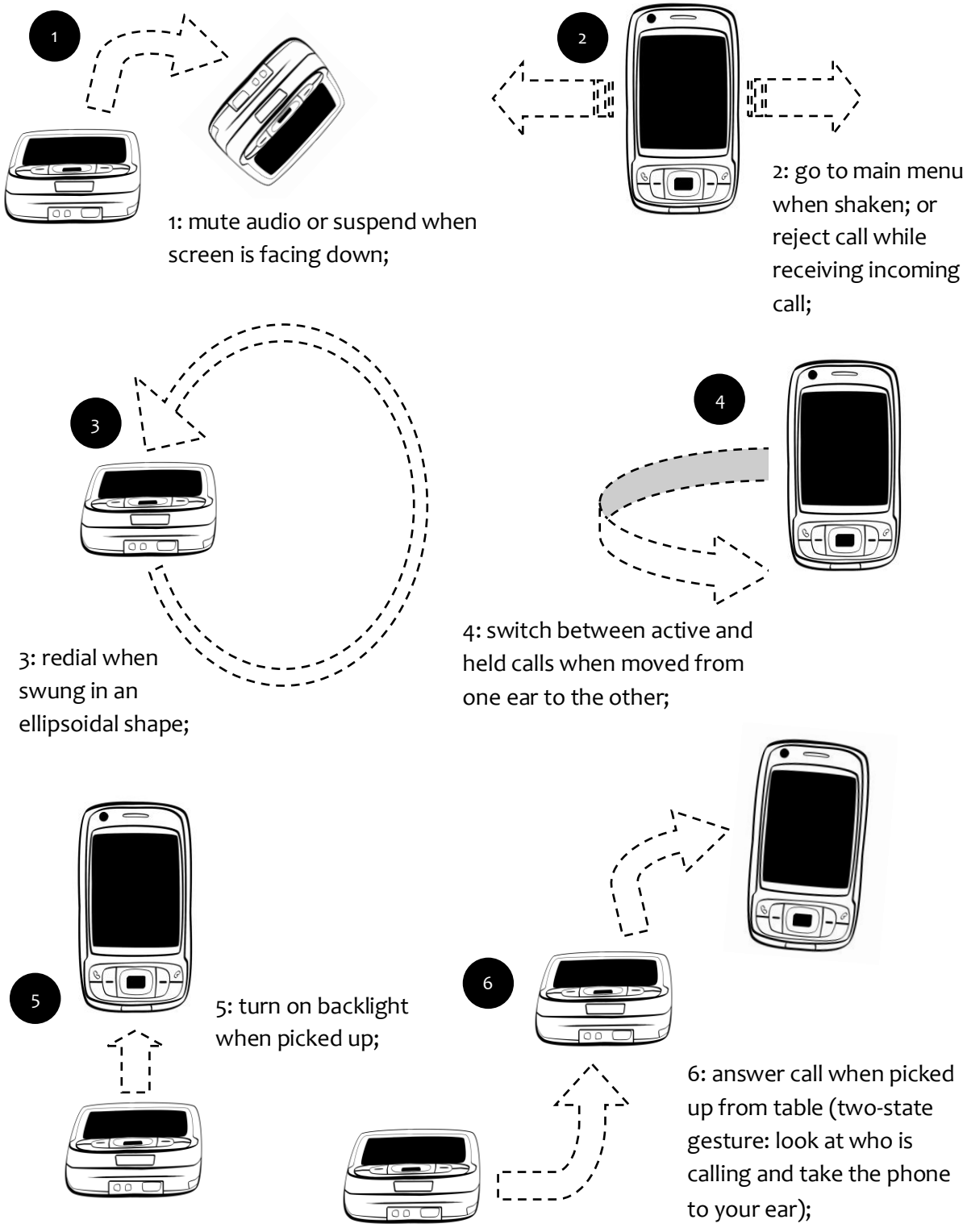


Figure 2 Mobile phone gestures and associated actions (photos made by Manuela Borza)

# Vision: TVs

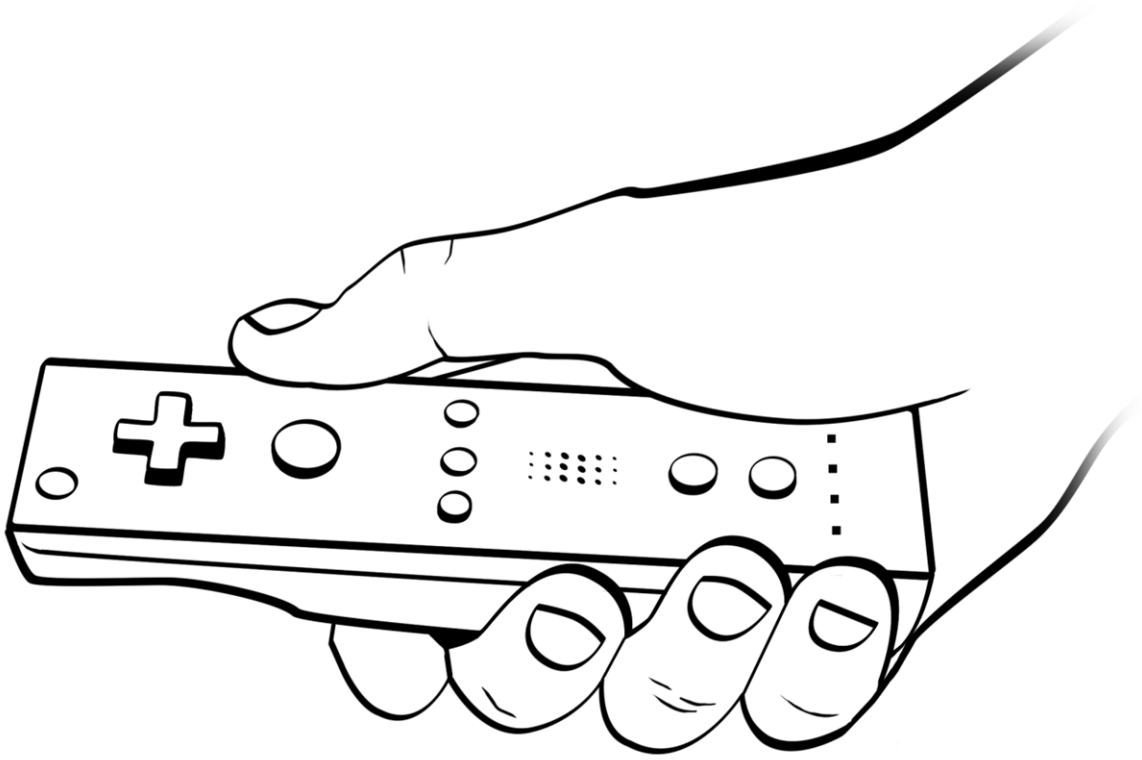

Figure 3 Human holds a remote control in his hand (photo made by Manuela Borza)

TVs have specialized remote controls that empower the user to complete simple actions within one step. However, people have always used to watch movies in dark rooms, and as a consequence, labels stamped on buttons cannot be easily seen.

Figure 4 displays nine gestures and some possible actions that can be achieved even in a dark environment with a remote control that embeds an accelerometer. To demonstrate the usefulness of such a new multi-media experience, I've used some of the gestures presented in Figure 4 to control the basic functions that Amarok exposes through `dcop`.

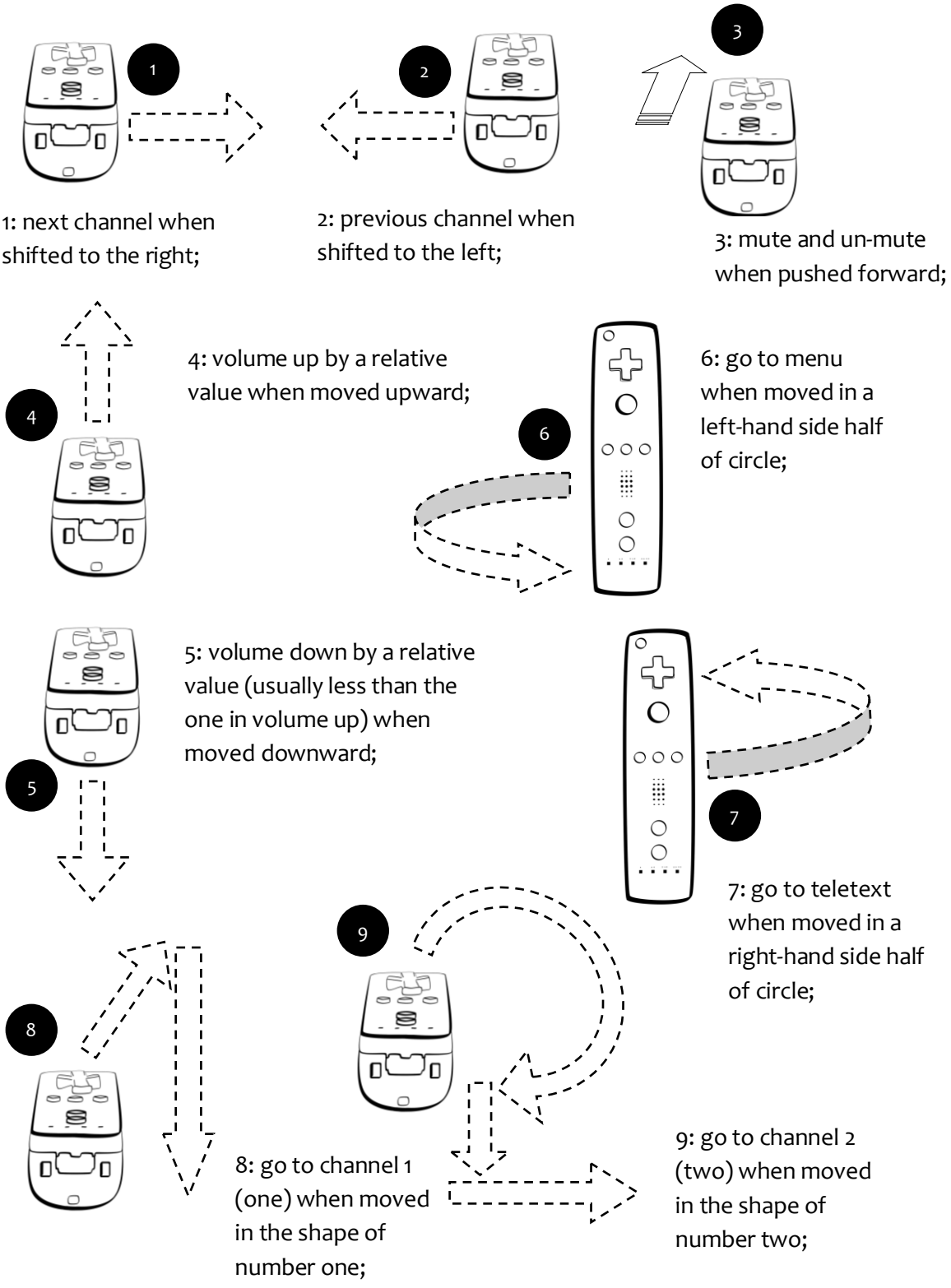


Figure 4 Remote control gestures and associated actions (photos made by Manuela Borza)

# Chapter 2. Hardware

The library can easily be adapted to use any signal source as long as the processing method from the library is called whenever a new acceleration report arrives. However, for the purposes of this demonstration, we will only use the Nintendo Wii Remote, because the Neo FreeRunner has only begun mass-production and there was a delay in the arrival of the prototype.

## Nintendo Wii Remote

As a consequence, the Nintendo Wii Remote over Bluetooth is used as the signal source. The device has an ADXL330 Small, Low Power, 3-Axis ±3G iMEMS Accelerometer manufactured by Analog Devices. Nintendo has selected the 100Hz (100 cycles per second) bandwidth for the X-, Y-, and Z-Axis. This means that the remote sends at most one hundred reports per second.

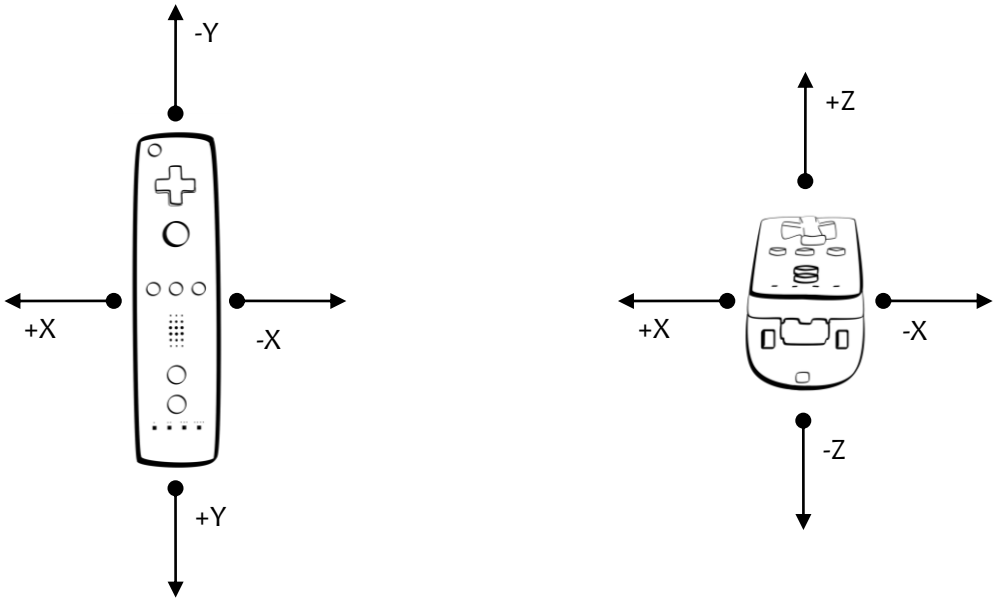


Figure 5 Nintendo Wii Remote Axes (photos made by Manuela Borza)

The Nintendo Wii Remote costs $40 and the Analog Devices ADXL330 costs $5.

The Nintendo Wii Remote communicates over Bluetooth and is designed to be used like a Bluetooth HID (2). The Bluetooth HID is based on the USB Hid. The Nintendo Wii Remote sends the Name Id *Nintendo RVL-CNT-01* with Vendor Id *0x57E* and Product Id *0x306*. The device does not require authentication. The device enters in discoverable mode when both buttons *1* and *2* are pressed.

# Chapter 3. Architecture

Similar to an automatic speech recognition system (3), a typical practical gesture recognition system consists of basic components shown in the dotted box of Figure 6. Applications interface with the decoder to get recognition results that may be used to adapt other components in the system. Acceleration models include the representation of knowledge about forces, accelerometer, and errors, etc. Language models refer to a system's knowledge of what constitutes a possible gesture, what gestures are likely to co-occur, and in what sequence. The semantics and functions related to an operation a user may wish to perform are also necessary for the language model. The motion signal is processed in the signal processing module that extracts salient feature vectors for the decoder. The decoder uses both acceleration and language models to generate the gesture sequence that has the maximum posterior probability for the input feature vectors. It can also provide information needed for the adaptation component to modify either the acceleration or language models so that improved performance can be obtained.

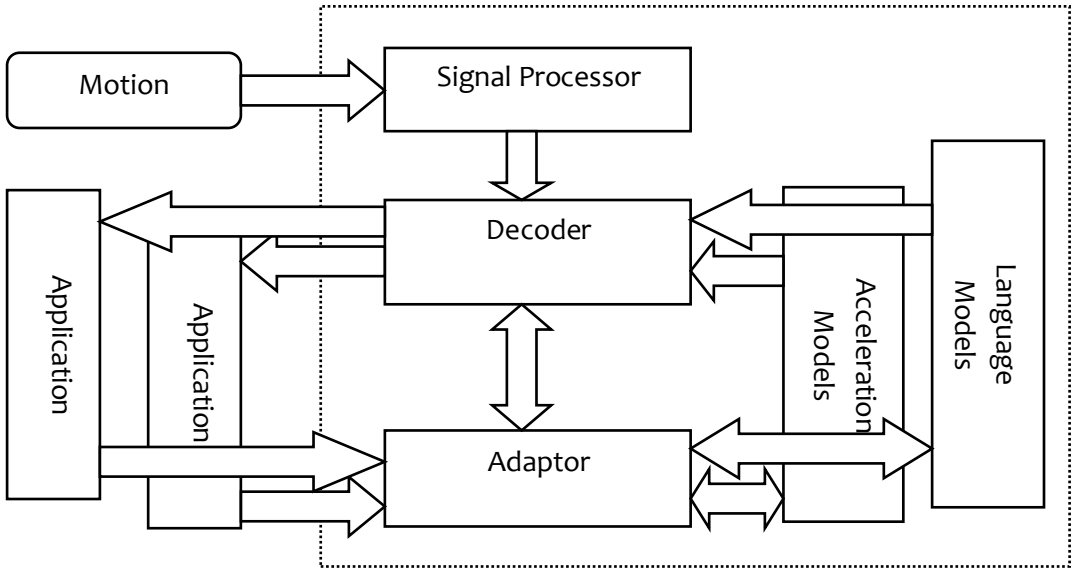


Figure 6 Basic system architecture of a gesture recognition system

Motion is read from the Wii Remote with a shared library (.so) called `wii`. The signal processor, decoder, and adaptor are implemented in another shared library (.so) called `ges`. These two libraries are used by `gesapp` which is a gesture application tool that is capable of creating, visualizing, plotting, and training Gaussian classes, or hidden Markov models. The demo application (a daemon) that also uses these two libraries is called `gesphone` and is responsible with associating each gesture with a command that is executed once a gesture is recognized. Acceleration models (.hmm and .gauss) are loaded from and saved to binary files. Language models are loaded from a human-readable text configuration file (.ges or .conf).

# Chapter 4. Signal Processing

The role of a signal processing module, as illustrated in Figure 6, is to remove static acceleration, and to extract salient features that are useful for acceleration matching.

## End-point Detection

To activate signal capture, one can use a number of modes including either push to make gesture or continuously listening for gestures. The push-to-make-gesture mode uses a special push event to activate or deactivate motion capture, which is immune to the potential unwanted gestures and can eliminate unnecessary use of processing resources to detect gesture events. This mode sometimes also requires one to push and hold while making the gesture. You push to indicate gesture's beginning and then release to indicate the end of motion capture. This mode is inappropriate to gesture recognition because we want to recognize an involuntary gesture, rather than a prepared and voluntary one. However, one could implement this mode with the help of the button A that is present on the front side of the Nintendo Wii Remote.

The continuously listening model listens all the time and automatically detects whether a dynamic motion signal is present or not; this is well suited for gestures. It needs a so-called motion end-point detector (3), which is typically based on an extremely efficient two-class pattern classifier. Such a classifier is used to filter out obvious static acceleration, but the ultimate decision on the movement boundary is left to the gesture recognizer. In comparison to the push-to-make-gesture mode, the continuously listening mode requires more processing resources, also with potential classification errors.

It is not critical for the automatic end-point detector to offer exact end-point accuracy. The key feature required of it is a low rejection rate (i.e. the automatic end-point detector should not interpret dynamic acceleration segments as static acceleration segments). Any false rejection leads to an error in the gesture recognizer. On the other hand, a possible false acceptance (i.e. the automatic end-point detector interprets static acceleration segments as motion segments) may be rescued by the gesture recognizer later if the recognizer has appropriate static acceleration models.

One way to implement the end-point detector is to use a simple adaptive two-class (dynamic acceleration vs. static acceleration or motion vs. noise) classifier to locate motion activities (with enough buffers at both ends) and notify the gesture recognizer for subsequent processing. For the two-class classifier, we've used a three dimensional feature vector having as components the magnitude at time $t$, the magnitude at time $t-1$ and the magnitude delta between the acceleration at time $t$ and $t-1$. Note that the last component of the vector is the magnitude delta, and not the delta magnitude. Nokia has used signed magnitudes (4). We've used:

$$magnitude = \sqrt{x^2 + y^2 + z^2}$$

where $x$, $y$, and $z$ are acceleration values recorded on X-, Y-, and Z-Axis.

In the ideal case of static acceleration, the magnitude should be equal to $1.0$.

Two Gaussian density functions $\{\Phi_1, \Phi_2\} = \Phi$ were used to model the static acceleration and dynamic acceleration, respectively. When enough frames are classified as dynamic acceleration segments by the efficient two-class classifier, the gesture recognizer is notified to start recognizing the signal. Experience shows that 20 frames (i.e. 0.2 seconds for a 100 Hz accelerometer) of dynamic acceleration should trigger the recognizer to start recognizing those 20 frames. In addition to those 20 frames, the gesture recognizer should also receive another 10 frames that were recorded before those classified as dynamic acceleration.

## Gaussian Classifiers

A Gaussian classifier is a Bayes' classifier where class-conditional probability density $p(\boldsymbol{x}|\omega_i)$ for each class $\omega_i$ is assumed to have a Gaussian distribution (bolded variables are vectors or matrices):

$$p(\boldsymbol{x}|\omega_i) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}_i|^{1/2}} e^{\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^t \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right]}$$

Note that the determinant of the covariance matrix is computed as $|\Sigma_i|^{1/2}$ and not as a square root because the determinant can be less than zero. We're using a trivariate Gaussian density, so dimension $d = 3$.

Assuming $p(\boldsymbol{x}|\omega_i)$ is a multivariate Gaussian density, a discriminant function can be written as follows:

$$d_i(\boldsymbol{x}) = \ln p(\boldsymbol{x}|\omega_i)\, P(\omega_i) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^t \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) + \ln P(\omega_i) - \frac{1}{2}\ln|\boldsymbol{\Sigma}| - \frac{d}{2}\ln 2\pi$$

Once we have the Gaussian discriminant functions (two in our case, one for the static acceleration and one for the dynamic acceleration), the decision process simply assigns data $\boldsymbol{x}$ to class $\omega_i$ if:

$$j = \underset{i}{\operatorname{argmax}}\, d_i(\boldsymbol{x})$$

The maximum likelihood estimation formulas for the multivariate Gaussian densities:

$$\widehat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{k=1}^{n} \boldsymbol{x}_k$$

$$\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_{k=1}^{n} (\boldsymbol{x}_k - \widehat{\boldsymbol{\mu}})(\boldsymbol{x}_k - \widehat{\boldsymbol{\mu}})^t$$

where $n$ is the size of the sample.

Although multivariate Gaussian pdfs provide a good level of flexibility, these are limited to a single maximum (i.e. a single bell-curve), and because of that, others use mixtures of multivariate Gaussian probability density functions. Note that by using only 1 mixture of a

multivariate Gaussian pdf, we're in the same case of a multivariate Gaussian pdf. We've also used mixtures of multivariate Gaussian pdfs, so from now on, we'll cover only the more general solution of the multivariate Gaussian mixtures, as they equally apply to multivariate Gaussian distributions.

Next section covers estimation formulas for the multivariate Gaussian mixtures that are crucial for the accuracy of the end-point detector. The following figures display the already trained classes of the classifier that we've used in the demonstration.

Trivariate Gaussian distributions can't be plotted as they would require a fourth dimension, so Figure 7 and Figure 8 show three univariate Gaussian distributions computed as parts of the trivariate Gaussian distribution for the static acceleration class and dynamic acceleration class, respectively. To achieve these, only the diagonal of the covariance matrix was used.

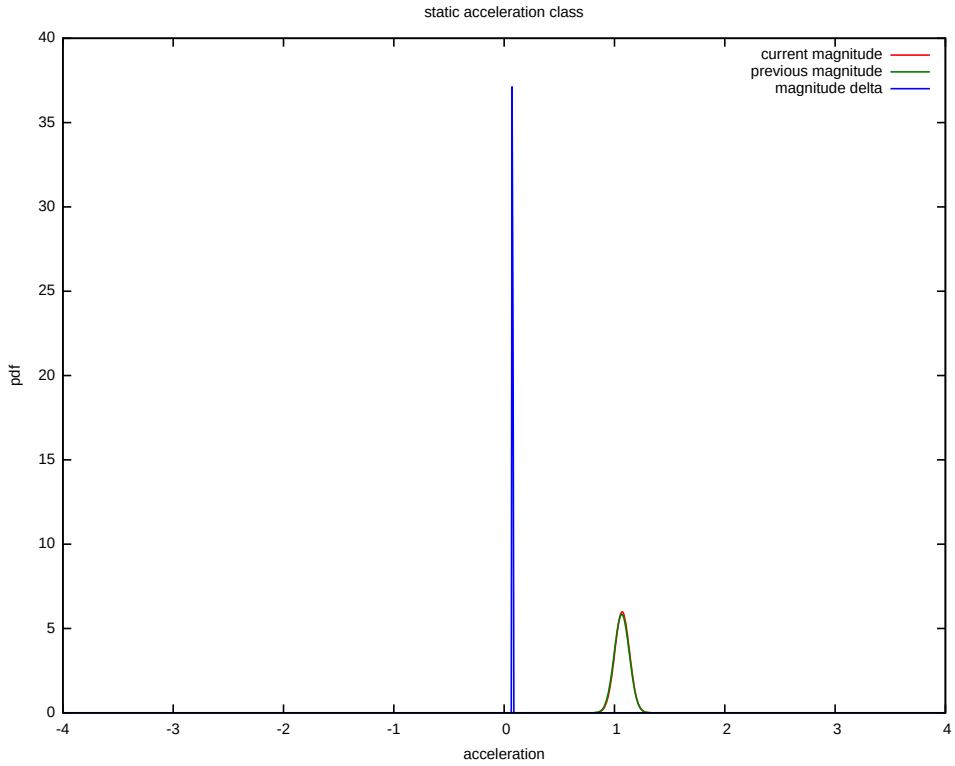


Figure 7 Gaussian probability density functions for the static acceleration class

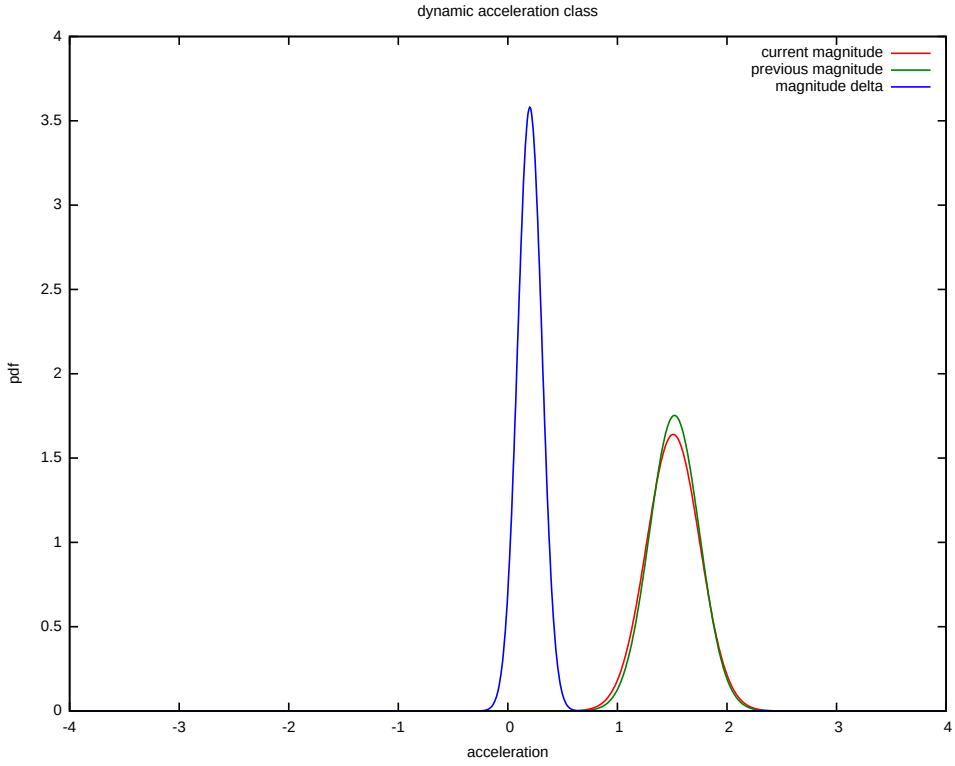


Figure 8 Gaussian probability density functions for the dynamic acceleration class

# Multivariate Gaussian Mixtures

As said before, a Gaussian classifier is usually modeled with a mixture of Gaussian probability density functions because of its generality. Mixture density estimation is a typical example of expected maximization estimation. In the mixtures of Gaussian density, the probability density for observable data $\boldsymbol{y}$ is the weighted sum of each Gaussian component:

$$p(\boldsymbol{y}|\boldsymbol{\Phi}) = \sum_{k=1}^{K} c_k p_k(\boldsymbol{y}|\boldsymbol{\Phi}_k) = \sum_{k=1}^{K} c_k N_k(\boldsymbol{y}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

where $0 \leq c_k \leq 1$ for $1 \leq k \leq K$ and $\sum_{k=1}^{K} c_k = 1.0$.

Unlike the case of a single Gaussian estimation, we also need to estimate the mixture weight $c_k$. In order to do so, we can assume that observable data $\boldsymbol{y}$ come from one of the component densities $p_X(\boldsymbol{y}|\boldsymbol{\Phi}_X)$, where $X$ is a random variable taking value from $\{1, 2, \ldots K\}$ to indicate the Gaussian component. It is clear that $x$ is unobserved and used to specify the pdf component $\boldsymbol{\Phi}_X$.

Estimation formulas for the multivariate Gaussian mixture densities are:

$$\gamma_k^i = \frac{c_k p_k(\boldsymbol{y}_i|\boldsymbol{\Phi}_k)}{P(\boldsymbol{y}_i|\boldsymbol{\Phi})}$$

$$\gamma_k = \sum_{i=1}^{N} \gamma_k^i = \sum_{i=1}^{N} \frac{c_k p_k(\boldsymbol{y}_i|\boldsymbol{\Phi}_k)}{P(\boldsymbol{y}_i|\boldsymbol{\Phi})}$$

$$\hat{c} = \frac{\gamma_k}{\sum_{k=1}^{K} \gamma_k}$$

$$\widehat{\boldsymbol{\mu}}_k = \frac{\sum_{i=1}^{N} \gamma_k^i \boldsymbol{y}_i}{\sum_{i=1}^{N} \mu_k^i} = \frac{\sum_{i=1}^{N} \frac{c_k p_k(\boldsymbol{y}_i|\boldsymbol{\Phi}_k)\boldsymbol{y}_i}{P(\boldsymbol{y}_i|\boldsymbol{\Phi})}}{\sum_{k=1}^{N} \frac{c_k p_k(\boldsymbol{y}_i|\boldsymbol{\Phi}_k)}{P(\boldsymbol{y}_i|\boldsymbol{\Phi})}}$$

$$\widehat{\boldsymbol{\Sigma}}_k = \frac{\sum_{i=1}^{N} \gamma_k^i (\boldsymbol{y}_i - \boldsymbol{\mu}_k)(\boldsymbol{y}_i - \boldsymbol{\mu}_k)^t}{\sum_{i=1}^{N} \gamma_k^i} = \frac{\sum_{i=1}^{N} \frac{c_k p_k(\boldsymbol{y}_i|\boldsymbol{\Phi}_k)(\boldsymbol{y}_i - \boldsymbol{\mu}_k)(\boldsymbol{y}_i - \boldsymbol{\mu}_k)^t}{P(\boldsymbol{y}_i|\boldsymbol{\Phi})}}{\sum_{i=1}^{N} \frac{c_k p_k(\boldsymbol{y}_i|\boldsymbol{\Phi}_k)}{P(\boldsymbol{y}_i|\boldsymbol{\Phi})}}$$

The quantity $\gamma_k^i$ can be interpreted as the posterior probability that the observed data $\boldsymbol{y}_i$ belong to Gaussian component $k$. This information as to whether the observed data $\boldsymbol{y}_i$ should belong to Gaussian component $k$ is hidden and can only be observed through the hidden vaiable $x$. The EM algorithm described here (3) is used to uncover how likely the observed data $\boldsymbol{y}_i$ are expected to be in each Gaussian component.

Mixtures are covered in detail in Estimation of Finite Mixture Models (5).

# Chapter 5. Use of Hidden Markov Models in Speech

Although initially introduced and studied in the late 1960s and early 1970s, statistical methods of Markov source or hidden Markov modeling have become increasingly popular in the last several years (1). There are two strong reasons why this has occurred. First, the models are very rich in mathematical structure and hence can form the theoretical basis for use in a wide range of applications. Second, the models when applied properly, work very well in practice for several important applications.

Neither the theory of hidden Markov models nor its applications to speech recognition is new. The basic theory was published in a series of classic papers by Baum and his colleagues in the late 1960s and early 1970s and was implemented for speech applications by Baker at CMU, and by Jelinek (6) and his colleagues at IBM in the 1970s. However, widespread understanding and application of the theory of HMMs to speech processing has occurred only within the past several years. There are several reasons why this has been the case. First, the basic theory of hidden Markov models was published in mathematical journals which were not generally read by engineers working on problems in speech processing. The second reason was that the original applications of the theory to speech processing did not provide sufficient tutorial material for most readers to understand the theory and to be able to apply it to their own research. As a result, several tutorial papers were written which provided a sufficient level of detail for a number of research labs to begin work using HMMs in individual speech processing applications.

Both Carnegie Mellon University's open speech software http://www.speech.cs.cmu.edu/sphinx and Cambridge University's HTK http://htk.eng.cam.ac.uk are a good starting point for those interested in using the existing tools for running experiments. Hidden Markov models have become the most prominent techniques for speech recognition today. Most of the state-of-the-art speech recognition systems on the market are based on HMMs.

Microsoft's Whisper engine offers general-purpose speaker-independent continuous speech recognition (3). Whisper can be used for command and control, dictation, and conversational applications. Whisper offers many features such as continuous speech recognition, speaker-independence with adaptation, and dynamic vocabulary. Whisper has a unified architecture that can be scaled to meet different application and platform requirements. The hidden Markov model topology is a three-state left-to-right model for each phone. The language model used in Whisper can be either the trigram or the context-free grammar. The difference is largely related to the decoder algorithm. The trigram lexicon has 60,000 most-frequent words extracted from a large text corpus. Word selection is based on both the frequency and the word's part-of-speech information. For example, verbs and the inflected forms have a higher weight than proper nous in the selection process. Whisper's overall word recognition error rate for speaker-independent continuous recognition is about 7% for the standard DARPA business-news dictation test set. For isolated dictation with similar materials, the error is less than 3%. If speaker-dependent data are available, it can further reduce the error rate, with less than 30 minutes' speech from each person.

# Chapter 6. Alternatives

## Dynamic Time Warping

Popular in late 1970s to mid 1980s, dynamic time warping is an algorithm that measures the similarity between two sequences (one is called the test pattern and the other is called the reference pattern) which can vary in time or speed. The DTW method can warp two motion templates in the time dimension to alleviate nonlinear distortion. Dynamic programming principles can drastically reduce the amount of computation by avoiding the enumeration of sequences that cannot possibly be optimal.

Here is how a pseudo-code algorithm would look like for dynamic time warping:

```
double DTW(double O1[1...T1], double O2[1...T2])
       declare double DTW[0...T1, 0...T2]
       declare integer i, j

       for i := 1 to T1
              DTW[i, 0] := positive infinity
       for j := 1 to T2
              DTW[0, j] := positive infinity
       DTW[0, 0] := 0

       for i := 1 to T1
              for j := 1 to T2
                     DTW[i, j] := distance(O1[i], O2[j]) +
                       minimum(DTW[i - 1, j], DTW[i, j - 1], DTW[i - 1, j - 1])

       return DTW[T1, T2]
```

For the distance method, one can use the Minkowski distance (e.g. Manhattan distance). The advantage of the dynamic programming lies in the fact that once a sub-problem is solved, the partial result can be stored and never needs to be recalculated; this is a very important principle. Motion recognition based on DTW is simple to implement and fairly effective for small-vocabulary motion recognition. Dynamic programming can temporally align patterns to account for differences in moving rates across different humans. However, it does not have a principled way to derive an averaged template for each pattern from a large amount of training samples. A multiplicity of reference training tokens is typically required to characterize the variation among different repetitions. As such, the hidden Markov models are a much better alternative for motion-based recognition.

# Chapter 7. Hidden Markov Models

The hidden Markov model is a very powerful statistical method of characterizing the observed data samples of a discrete-time series. Three basic problems of interest have to be addressed to be able to apply these models to real-world applications: the evaluation problem, the decoding problem, and the learning problem. But first, let's start with some definitions.

## Markov Chain

A Markov chain models a class of random processes that incorporates a minimum amount of memory without being completely memory-less (3). Let $\boldsymbol{X} = X_1, X_2, \dots, X_n$ be a sequence of random variables from a finite discrete alphabet $O = \{o_1, o_2, \dots, o_M\}$. Based on Bayes' rule, we have:

$$P(X_1, X_2, \dots, X_n) = P(X_1) \prod_{i=2}^{n} P\left(X_i \middle| X_1^{i-1}\right)$$

where $X_1^{i-1} = X_1, X_2, \dots, X_{i-1}$. The random variables $X$ are said to form a first-order Markov chain, provided that:

$$P\left(X_i \middle| X_1^{i-1}\right) = P(X_i | X_{i-1})$$

This equation is also known as the Markov assumption. This assumption uses very little memory to model dynamic data sequences: the probability of the random variable at a given time depends only on the value at the preceding time.

As a consequence, for the first-order Markov chain, the new equation is:

$$P(X_1, X_2, \dots, X_n) = P(X_1) \prod_{i=2}^{n} P(X_i | X_{i-1})$$

The Markov chain can be used to model time-invariant events if we discard the time index $i$:

$$P(X_i = s | X_{i-1} = s') = P(s | s')$$

If we associate $X_i$ to a state, the Markov chain can be represented by a finite state process with transition between states specified by the probability function $P(s|s')$. Using this finite state representation, the Markov assumption is translated to the following: the probability that the Markov chain will be in a particular state at a given time depends only on the state of the Markov chain at the previous time.

# Definition

A hidden Markov model is a collection of states connected by transitions. Each state is characterized by two sets of probabilities: a transition probability, and either a discrete output probability distribution (used in HMMs) or continuous output probability density function (used in CDHMMs) which, given the state, defines the conditional probability of emitting each output symbol from a finite alphabet or a continuous random vector.

The hidden Markov model can be viewed as a double-embedded stochastic process with an underlying stochastic process (the state sequence) not directly observable. This underlying process can only be probabilistically associated with another observable stochastic process producing the sequence of features we can observe. Formally speaking (3), a hidden Markov model is defined by:

- $O = \{o_1, o_2, \dots, o_M\}$ is an output observation alphabet (strictly for definition purposes, as we'll use continuous random variables, and not discrete ones). The observation symbols correspond to the physical output of the system being modeled, in our case a 3-dimensional vector with acceleration values. The observations are not characterized by discrete symbols chosen from a finite alphabet, and therefore we cannot use a discrete probability density within each state. For our problem, the observations are continuous signals, or vectors that have multiple components. Although it is possible to quantize such continuous signals via codebooks etc. there is serious degradation associated with such quantization. Hence, it's more advantageous to use HMMs with continuous observation densities.
- $\Omega = \{1,2, \dots, N\}$ is a set of states representing the state sequence. These states are hidden and for many practical applications there is often some physical significance attached to the states of the model. Generally, the states are interconnected in such a way that any state can be reached from any other state (e.g. ergodic model).
- $A = \{a_{ij}\}$ is a transition probability matrix, where $a_{ij}$ is the probability of taking a transition from state $i$ to state $j$, i.e.

$$a_{ij} = P(s_{t+1} = j | s_t = i)$$

$$1 \leq i, j \leq N$$

$$1 \leq t \leq T - 1$$

- $B = \{b_j(k)\}$ is an output probability matrix, where $b_j(k)$ is the probability of emitting symbol $o_k$ when state $j$ is entered, i.e.

$$b_j(k) = P(X_t = o_k | s_t = j)$$

$$1 \leq j \leq N$$

$$1 \leq t \leq T$$

$$1 \leq k \leq M$$

- $\pi = \{\pi_i\}$ is an initial state distribution, where:

$$\pi_i = P(s_1 = i)$$

$$1 \leq i \leq N$$

Since these are all probabilities, they must satisfy the following properties (it will be seen that after the re-estimation of the parameters, some of the values will exceed the imposed boundaries):

$$\sum_{j=1}^{N} a_{ij} = 1.0$$

$$\sum_{k=1}^{M} b_j(k) = 1.0$$

$$\sum_{i=1}^{N} \pi_i = 1.0$$

For convenience, we will use the notation $\Phi = (A, B, \pi)$.

## Definition of Continuous Mixture Density Hidden Markov Models

If the observation does not come from a finite set, but from a continuous space, the discrete output distribution discussed in the previous section needs to be modified. The difference between the discrete and the continuous HMM lies in the different form of output probability functions.

In choosing continuous output probability density functions $b_j(\boldsymbol{x})$, the first candidate is multivariate Gaussian mixture density functions (as the ones used in the classifier); this is because they can approximate any continuous probability density function. With $M$ Gaussian mixture density functions, we have:

$$b_j(\boldsymbol{x}) = \sum_{k=1}^{M} c_{jk}\, N(\boldsymbol{x}, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}) = \sum_{k=1}^{M} c_{jk}\, b_{jk}(\boldsymbol{x})$$

Where $\boldsymbol{x}$ is the vector being modeled, $c_{jk}$ is the mixture coefficient for the $k$th mixture in state $j$ and $N$ denotes a single Gaussian density function with mean vector $\boldsymbol{\mu}_{jk}$ and covariance matrix $\boldsymbol{\Sigma}_{jk}$ for the $k$th mixture component in state $j$.

The mixture gains satisfy the stochastic constraint:

$$\sum_{k=1}^{M} c_{jk} = 1.0$$

$$c_{jk} \geq 0.0$$

$$1 \leq j \leq N$$

$$1 \leq k \leq M$$

so that the pdf is properly normalized, i.e.

$$\int_{-\infty}^{+\infty} b_j(x)dx = 1.0$$

$$1 \le j \le N$$

## Choice of Model for an Accelerometer

The practical model depends on the source that sends the signal. The accelerometer has three axes and thus sends symbols that are composed of three values:

$$\boldsymbol{x}_t = [x_t, y_t, z_t]$$

Mean vector:

$$\boldsymbol{\mu} = [\mu_x, \mu_y, \mu_z]$$

Covariance matrix:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xz}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yz}^2 \\ \sigma_{zx}^2 & \sigma_{zy}^2 & \sigma_{zz}^2 \end{bmatrix}$$

The formula to compute the determinant of the covariance matrix is:

$$|\Sigma| = \begin{vmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xz}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yz}^2 \\ \sigma_{zx}^2 & \sigma_{zy}^2 & \sigma_{zz}^2 \end{vmatrix}$$
$$= \sigma_{xx}^2\sigma_{yy}^2\sigma_{zz}^2 + \sigma_{xy}^2\sigma_{yz}^2\sigma_{zx}^2 + \sigma_{xz}^2\sigma_{yx}^2\sigma_{zy}^2 - \sigma_{xz}^2\sigma_{yy}^2\sigma_{zx}^2 - \sigma_{yx}^2\sigma_{xy}^2\sigma_{zz}^2 - \sigma_{xx}^2\sigma_{zy}^2\sigma_{yz}^2$$

and the formula to compute the inverse of the covariance matrix is:

$$\Sigma^{-1} = \frac{1}{|\Sigma|}\begin{bmatrix} \begin{vmatrix} \sigma_{yy}^2 & \sigma_{yz}^2 \\ \sigma_{zy}^2 & \sigma_{zz}^2 \end{vmatrix} & \begin{vmatrix} \sigma_{xz}^2 & \sigma_{xy}^2 \\ \sigma_{zz}^2 & \sigma_{zy}^2 \end{vmatrix} & \begin{vmatrix} \sigma_{xy}^2 & \sigma_{xz}^2 \\ \sigma_{yy}^2 & \sigma_{yz}^2 \end{vmatrix} \\ \begin{vmatrix} \sigma_{yz}^2 & \sigma_{yx}^2 \\ \sigma_{zz}^2 & \sigma_{zx}^2 \end{vmatrix} & \begin{vmatrix} \sigma_{xx}^2 & \sigma_{xz}^2 \\ \sigma_{zx}^2 & \sigma_{zz}^2 \end{vmatrix} & \begin{vmatrix} \sigma_{xz}^2 & \sigma_{xx}^2 \\ \sigma_{yz}^2 & \sigma_{yx}^2 \end{vmatrix} \\ \begin{vmatrix} \sigma_{yx}^2 & \sigma_{yy}^2 \\ \sigma_{zx}^2 & \sigma_{zy}^2 \end{vmatrix} & \begin{vmatrix} \sigma_{xy}^2 & \sigma_{xx}^2 \\ \sigma_{zy}^2 & \sigma_{zx}^2 \end{vmatrix} & \begin{vmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 \end{vmatrix} \end{bmatrix}$$
$$= \frac{1}{|\Sigma|}\begin{bmatrix} \sigma_{yy}^2\sigma_{zz}^2 - \sigma_{yz}^2\sigma_{zy}^2 & \sigma_{xz}^2\sigma_{zy}^2 - \sigma_{xy}^2\sigma_{zz}^2 & \sigma_{xy}^2\sigma_{yz}^2 - \sigma_{xz}^2\sigma_{yy}^2 \\ \sigma_{yz}^2\sigma_{zx}^2 - \sigma_{yx}^2\sigma_{zz}^2 & \sigma_{xx}^2\sigma_{zz}^2 - \sigma_{xz}^2\sigma_{zx}^2 & \sigma_{xz}^2\sigma_{yx}^2 - \sigma_{xx}^2\sigma_{yz}^2 \\ \sigma_{yx}^2\sigma_{zy}^2 - \sigma_{yy}^2\sigma_{zx}^2 & \sigma_{xy}^2\sigma_{zx}^2 - \sigma_{xx}^2\sigma_{zy}^2 & \sigma_{xx}^2\sigma_{yy}^2 - \sigma_{xy}^2\sigma_{yx}^2 \end{bmatrix}$$

More details can be found in Chapter 9. Usage and Scenarios on how to choose the states and the initial estimates for the mean vector and the covariance matrix.

# The Evaluation Problem

Given the observation sequence $\boldsymbol{x} = \boldsymbol{x}_1\boldsymbol{x}_2 \cdots \boldsymbol{x}_T$, and a model $\Phi = (A, B, \pi)$, how do we efficiently compute $P(\boldsymbol{x}|\Phi)$, the probability of the observation sequence, given the model?

## Forward-Backward Procedure

Consider the forward variable $\alpha_t(i)$ defined as

$$\alpha_t(i) = P(\boldsymbol{x}_1\boldsymbol{x}_2 \cdots \boldsymbol{x}_t, s_t = i|\ \Phi)$$

i.e. the probability of the partial observation sequence $\boldsymbol{x}_1\boldsymbol{x}_2 \cdots \boldsymbol{x}_t$, and state $i$ at time $t$, given the model $\Phi$. We can solve for $\alpha_t(i)$ inductively, as follows:

Initialization:

$$\alpha_1(i) = \pi_i b_i(\boldsymbol{x}_1)$$

$$1 \le i \le N$$

Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right] b_j(\boldsymbol{x}_{t+1})$$

$$1 \le t \le T-1$$

$$1 \le j \le N$$

Solution:

$$P(\boldsymbol{x}|\Phi) = \sum_{i=1}^{N} \alpha_T(i)$$

Strictly speaking, we only need the forward part of the forward-backward procedure to solve the evaluation problem. We will introduce the backward part of the procedure in this section since it will be used to help solve the learning problem.

In a similar manner, we can consider a backward variable $\beta_t(i)$, defined as:

$$\beta_t(i) = P(\boldsymbol{x}_{t+1}\boldsymbol{x}_{t+2} \cdots \boldsymbol{x}_T|s_t = i, \Phi)$$

i.e. the probability of the partial observation sequence $\boldsymbol{x}_{t+1}\boldsymbol{x}_{t+2} \cdots \boldsymbol{x}_T$, given state $i$ at time $t$ and model $\Phi$. Again, we can solve for $\beta_t(i)$ inductively, as follows:

Initialization:

$$\beta_T = 1.0$$

$$1 \le i \le N$$

Induction:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(\boldsymbol{x}_{t+1})\, \beta_{t+1}(j)$$

$$T - 1 \geq t \geq 1$$

$$1 \leq i \leq N$$

# The Decoding Problem

Given the observation sequence $\boldsymbol{x} = \boldsymbol{x}_1 \boldsymbol{x}_2 \cdots \boldsymbol{x}_T$ and the model $\Phi = (A, B, \pi)$ how do we choose a corresponding state sequence $Q = s_1 s_2 \cdots s_T$ which is optimal is some meaningful sense, i.e. best explains the observations?

## Viterbi Procedure

To find the single best state sequence $Q = s_1 s_2 \cdots s_T$ for the given observation sequence $\boldsymbol{x} = \boldsymbol{x}_1 \boldsymbol{x}_2 \cdots \boldsymbol{x}_T$ we need to define a quantity:

$$\delta_t(i) = \max_{s_1, s_2, \cdots, s_{t-1}} P(s_1 s_2 \cdots s_t = i, \boldsymbol{x}_1 \boldsymbol{x}_2 \cdots \boldsymbol{x}_t | \Phi)$$

i.e. the best score (highest probability) along a single path, at time $t$, which accounts for the first $t$ observations and ends in state $i$. To actually retrieve the state sequence, we need to keep track of the argument which maximized the previous quantity, for each $t$ and $i$. We do this via a variable:

$$\psi_t(i)$$

Initialization:

$$\delta_1(i) = \pi_i b_i(\boldsymbol{x}_1)$$

$$\psi_1(i) = 0$$

$$1 \leq i \leq N$$

Induction:

$$\delta_t(j) = \max_{1 \leq i \leq N} \left[\delta_{t-1}(i) a_{ij}\right] b_j(\boldsymbol{x}_t)$$

$$\psi_t(j) = \operatorname*{argmax}_{1 \leq i \leq N} \left[\delta_{t-1}(i) a_{ij}\right]$$

$$2 \leq t \leq T$$

$$1 \leq j \leq N$$

Solution:

$$P^* = \max_{1 \le i \le N} [\delta_T(i)]$$

$$s_T^* = \operatorname*{argmax}_{1 \le i \le N} [\delta_T(i)]$$

Sequence:

$$s_t^* = \psi_{t+1}(s_{t+1}^*)$$

$$T - 1 \ge t \ge 1$$

# The Learning Problem

The most difficult problem of hidden Markov models is to determine a method to adjust the model parameters $(A, B, \pi)$ to maximize the probability of the observation sequence given the model. There is no known way to analytically solve for the model which maximizes the probability of the observation sequence. In fact, given any finite observation sequence as training data, there is no optimal way of estimating the model parameters. We can, however, choose $\Phi = (A, B, \pi)$ such that $P(\boldsymbol{x}|\Phi)$ is locally maximized using an iterative procedure such as the Baum-Welch method.

## Baum-Welch Procedure

In order to describe the procedure for re-estimation (iterative update and improvement) of HMM parameters, we first define $\xi_t(i,j)$, the probability of being in state $i$ at time $t$, and state $j$ at time $t+1$, given the model and the observation sequence, i.e.

$$\xi_t(i,j) = P(s_t = i, s_{t+1} = j | \boldsymbol{x}, \Phi)$$

We can write $\xi_t(i,j)$ as:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(\boldsymbol{x}_{t+1}) \beta_{t+1}(j)}{P(\boldsymbol{x}|\Phi)} = \frac{\alpha_t(i) a_{ij} b_j(\boldsymbol{x}_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(\boldsymbol{x}_{t+1}) \beta_{t+1}(j)}$$

where the numerator term is just $P(s_t = i, s_{t+1} = j | \boldsymbol{x}, \Phi)$ and the division by $P(\boldsymbol{x}|\Phi)$ gives the desired probability measure.

We can also define $\gamma_t(i)$ as the probability of being in state $i$ at time $t$, given the observation sequence and the model; hence, we can relate $\gamma_t(i)$ to $\xi_t(i,j)$ by summing over $j$, giving:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

If we sum $\gamma_t(i)$ over the time index $t$, we get a quantity which can be interpreted as the expected number of times that state $i$ is visited, or equivalently, the expected number of

transitions made from state $i$. Similarly, summation of $\xi_t(i,j)$ over $t$ can be interpreted as the expected number of transitions made from state $i$ to state $j$. That is:

$$\sum_{t=1}^{T-1} \gamma_t(i) = expected\ number\ of\ transitions\ from\ state\ i$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = expected\ number\ of\ transitions\ from\ state\ i\ to\ state\ j$$

Using the above formulas, we can give a method for re-estimation of the parameters of an HMM:

$$\hat{\pi}_i = \gamma_1(i)$$

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

$$\hat{b}_j(k) = \frac{\sum_{t=1}^{T} \gamma_t(j)\ s.t.\ x_t = o_k}{\sum_{t=1}^{T} \gamma_t(j)}$$

## Continuous Case

$$\gamma_t(j,k) = \left[\frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}\right]\left[\frac{c_{jk}\, N(\boldsymbol{x}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk})}{\sum_{m=1}^{M} c_{jm}\, N(\boldsymbol{x}_t, \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})}\right]$$

$$\hat{c}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \gamma_t(j,k)}$$

$$\hat{\boldsymbol{\mu}}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)\boldsymbol{x}_t}{\sum_{t=1}^{T} \gamma_t(j,k)}$$

$$\hat{\boldsymbol{\Sigma}}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)\,(\boldsymbol{x}_t - \boldsymbol{\mu}_{jk})(\boldsymbol{x}_t - \boldsymbol{\mu}_{jk})^t}{\sum_{t=1}^{T} \gamma_t(j,k)}$$

# Chapter 8. Implementation Issues and Limitations

If the reader thinks that implementation issues should be overlooked, he/she is wrong, because HMMs simply don't work well (or at all) in practice without these workarounds.

## Initial Estimates

In theory, the re-estimation algorithm of the HMM should reach a local maximum for the likelihood function. A key question is how to choose the right initial estimates of the HMM parameters so that the local maximum becomes the global maximum. In our case where we've used continuous mixture density HMMs, good initial estimates for the Gaussian density functions are essential (1). Next chapter explains how to segment the acceleration signal and how to take initial estimates.

## Left-to-Right Model Topology

Motion is a time-evolving non-stationary signal. Each HMM state has the ability to capture some quasi-stationary segment in the non-stationary motion signal. A left-to-right topology is a natural candidate to model the acceleration signal. It has a self-transition to each state that can be used to model contiguous motion features belonging to the same state. When the quasi-stationary motion segment evolves, the left-to-right transition enables a natural progression of such evolution. In such a topology, each state has a state dependent output probability distribution that can be used to interpret the observable motion signal. This topology is, in fact, one of the most popular HMM structures used in state-of-the-art speech recognition systems (3).

The fundamental property of all left-to-right HMMs is that the state transition coefficients have the property:

$$a_{ij} = 0.0$$

$$1 \leq j < i \leq N$$

i.e. no transitions are allowed to states whose indices are lower than the current state. Furthermore, the initial state probabilities have the property:

$$\pi_i = 0.0 \; if \; i \neq 1$$

$$\pi_i = 1.0 \; if \; i = 1$$

$$1 \leq i \leq N$$

since the state sequence must begin in state 1, and end in state N. Often, with left-to-right models, additional constraints are placed on the state transition coefficients to make sure that large changes do not occur, i.e.

$$a_{ij} = 0.0$$

$$j > i + \Delta$$

is often used. In particular, we've taken $\Delta = 1$, and the form of the state transition matrix looks like:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0.0 & 0.0 \\ 0.0 & a_{22} & a_{23} & 0.0 \\ 0.0 & 0.0 & a_{33} & a_{34} \\ 0.0 & 0.0 & 0.0 & a_{44} \end{bmatrix}$$

It should be clear that, for the last state in a left-to-right model, the state transition coefficients are specified as:

$$a_{NN} = 1.0$$

$$a_{Ni} = 0.0$$

$$1 \leq i < N$$

## Deleted Interpolation

A problem associated with training HMM parameters via re-estimation methods is that the observation sequence used for training is, of necessity, finite. Thus there is often an insufficient number of occurrences of different model events to give good estimates of the model parameters. A solution to this problem is to interpolate one set of parameter estimates with another set of parameter estimates from a model for which an adequate amount of training data exists. Thus, if we have estimates for the parameters of the model $\lambda = (A, B, \pi)$, and as well as for the parameters of the model $\lambda' = (A', B', \pi')$, then the interpolated model $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$, is obtained as:

$$\bar{\lambda} = \varepsilon\lambda + (1 - \varepsilon)\lambda'$$

Where $\varepsilon$ represents the weighting of the parameters of the first model, and $(1 - \varepsilon)$ represents the weighting of the parameters of the second model. We've used and would recommend using this technique when new estimates are given with the Baum-Welch algorithm. Hence, the re-estimated model will approach the global maximum slower.

When there is insufficient training data, one should add extra constraints to the model parameters to ensure that no model parameter estimate falls below a specified level. For example, we might specify a constraint, for a discrete symbol model, that:

$$b_j(k) \geq \delta$$

or, for a continuous distribution model (as ours), that:

$$\mathbf{\Sigma}_{jk}(r, r) \geq \delta$$

The constraints should be applied as a postprocessor to the re-estimation equations such that if a constraint is violated, the relevant parameter is manually corrected, and all remaining parameters are rescaled so that the densities obey the required stochastic constraints. We've taken $\delta = 0.001$ in our experiments with the Nintendo Wii Remote.

# Probability Representations

When we compute the forward and backward probabilities in the forward-backward algorithm, they will approach zero in exponential fashion if the length of the observation sequence becomes large enough. For sufficiently large $T$, the dynamic range of these probabilities will exceed the precision range of essentially any machine. Thus, in practice, it will result in underflow on the computer if probabilities are represented directly. We can solve this implementation problem by scaling these probabilities with some scaling coefficient so that they remain within the dynamic range of the computer. All of these scaling coefficients can be removed at the end of the computation without affecting the overall precision. We've used scaling on double precision. As a note, single precision (i.e. float) should never be used to represent probabilities.

The forward variable $\alpha_t(i)$ should be multiplied by a scaling coefficient, $S_t$ like:

$$S_t = 1.0/\sum_i \alpha_t(i)$$

The backward variable $\beta_t(i)$ should also be multiplied by $S_t$ for $1 \leq t \leq T$.

Notice that $\alpha_t(i)$ and $\beta_t(i)$ are computed recursively in exponential fashion; therefore, at time $t$, the total scale factor applied to the forward variable $\alpha_t(i)$ is:

$$Scale_\alpha(t) = \prod_{k=1}^{t} S_k$$

and the total scale factor applied to the backward variable $\beta_t(i)$ is:

$$Scale_\beta = \prod_{k=t}^{T} S_k$$

Equation:

$$\sum_i \alpha_T^{'} = Scale_\alpha(T) \sum_i \alpha_T(i) = Scale_\alpha(T) P(\boldsymbol{x}|\boldsymbol{\Phi})$$

The scaled intermediate probability, $\gamma_t^{'}(i,j)$, can be written as:

$$\gamma_t^{'}(i,j) = \frac{Scale_\alpha(t-1)\alpha_{t-1}(i)a_{ij}b_j(\boldsymbol{x}_t)\beta_t(j)Scale_\beta(t)}{Scale_\alpha(T)\sum_{i=1}^{N}\alpha_T(i)} = \gamma_t(i,j)$$

Thus, the intermediate probabilities can be used in the same way as the unscaled probabilities, because the scaling factor is cancelled out. Therefore, re-estimation formulas can be kept exactly exccept that $P(\boldsymbol{x}|\boldsymbol{\Phi})$ should be computed according to:

$$P(x|\boldsymbol{\Phi}) = \sum_i \frac{\alpha_T^{'}}{Scale_\alpha(T)}$$

The above technique should be used for the forward-backward algorithm. However, for the Viterbi algorithm, we've avoided undeflow by using a logarithmic representation for all the probabilities. This not only ensures that scaling is unnecessary, as undeflow cannot happen, but also offers the benefit that integers can be used to represent the logarithmic values, thereby changing floating point operations to fixed point ones, which is particulary suitable for Viterbi-style computation. Integer values should be used instead of floating point values when one needs extra performance that can only be achieved on integer operations. This improvement with integer operations will be implemented for the Neo FreeRunner as performance is a de-facto requirement.

# Limitations

## Duration Modeling

One major weakness of conventional HMMs is that they do not provide an adequate representation of the temporal structure of motion. This is because the probability of state occupancy decreases exponentially with time, as:

$$d_i(t) = a_{ij}^t \left(1 - a_{ij}\right)$$

The probability of $t$ consecutive observations in state $i$ is the probability of taking the self-loop at state $i$ for $t$ times.

In practice, duration models offer only modest improvement for independent continuous recognition. Nevertheless, duration information is very effective for pruning unlikely candidates during the motion decoding process.

# Chapter 9. Usage and Scenarios

This chapter focuses on the creation, visualization, and adaptation of a gesture with the tools I've developed. As part of this tutorial, we will create a gesture that has the shape of the letter Z. In addition, we will also present the mechanism that allows us to control the basic functions in Amarok.

## How to Create a Gesture

- `./gesapp.sh --show-accel`
  - Press 1 and 2 on the Wii.

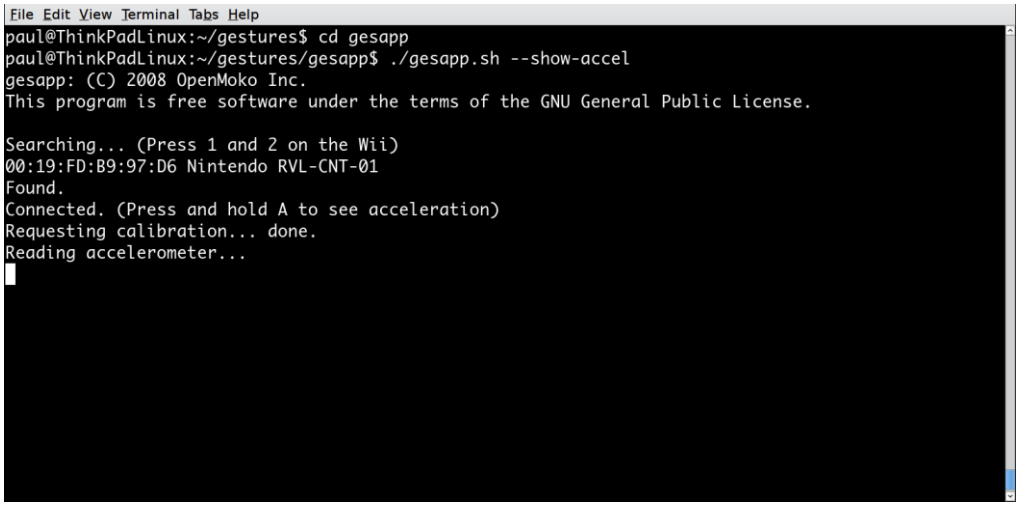


Figure 9 Gesture application in show acceleration mode before making a gesture

  - Press and hold button A on the Wii Remote while you make a Z in the air. Release when done.

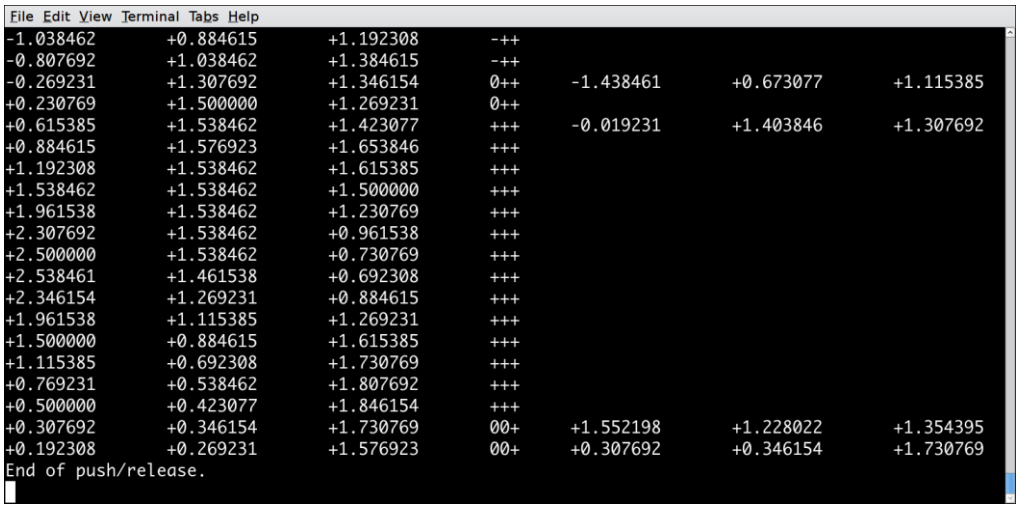


Figure 10 Gesture application in show acceleration mode after making a gesture

## Initial Estimates

- Observe the values in the right-hand side of the screen. When the accelerometer changes the axes on which it accelerates, the program computes the average of the previous values while it remained in the same state. These are helpful when you have to decide on the number of states for a certain gesture and on the initial estimates for the parameters of the model.

- `./gesapp.sh --new-model z.hmm`
  - For the Z-like gesture, we will use 5 states. When a new model is created, the left-to-right hidden Markov model gets its transition matrix pre-populated with a uniform distribution and allows only transitions to the next state. Follow the instructions on the screen, and when prompted, enter the following values:

| State 0, Mixture 0 | X | Y | Z |
|---|---|---|---|
| Means | -0.25 | +0.37 | +0.51 |
| Covariances (on diagonal) | +0.05 | +0.05 | +0.05 |
| State 1, Mixture 0 | X | Y | Z |
| Means | -1.29 | +0.64 | +0.80 |
| Covariances (on diagonal) | +0.10 | +0.05 | +0.05 |
| State 2, Mixture 0 | X | Y | Z |
| Means | +1.8 | +1.51 | +0.97 |
| Covariances (on diagonal) | +0.08 | +0.08 | +0.05 |
| State 3, Mixture 0 | X | Y | Z |
| Means | -1.97 | +1.63 | +1.19 |
| Covariances (on diagonal) | +0.1 | +0.1 | +0.1 |
| State 4, Mixture 0 | X | Y | Z |
| Means | +1.3 | +1.9 | +1.5 |
| Covariances (on diagonal) | +0.08 | +0.08 | +0.08 |

- `./gesapp.sh --view-model z.hmm`
  - Several scalable vector graphics (.svg) are automatically generated with `gnuplot`, one for each state. Figure 13, Figure 14, Figure 15, Figure 16, and Figure 17 display the distributions with the initial estimates, i.e. before training.

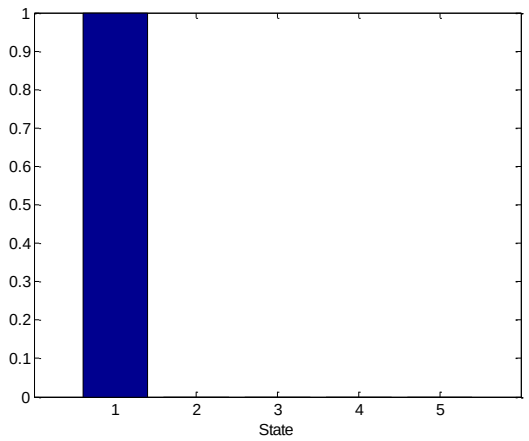


Figure 11 Initial state probabilities before training for gesture Z

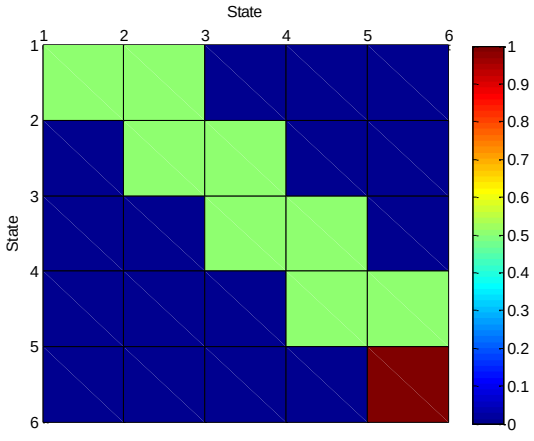


Figure 12 State transition probabilities before training for gesture Z

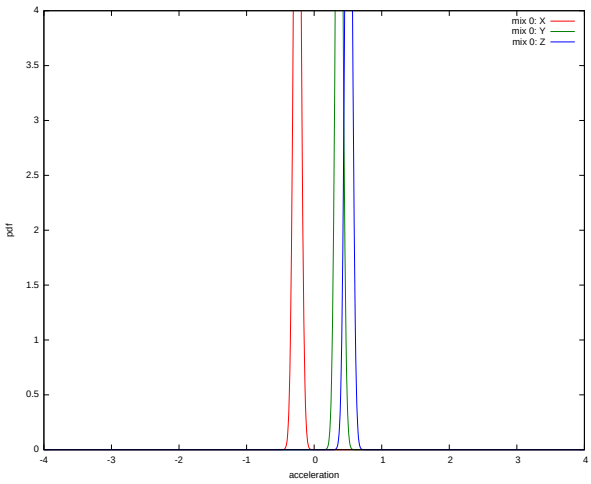


Figure 13 Gaussian probability density functions in state 0 before training for gesture Z

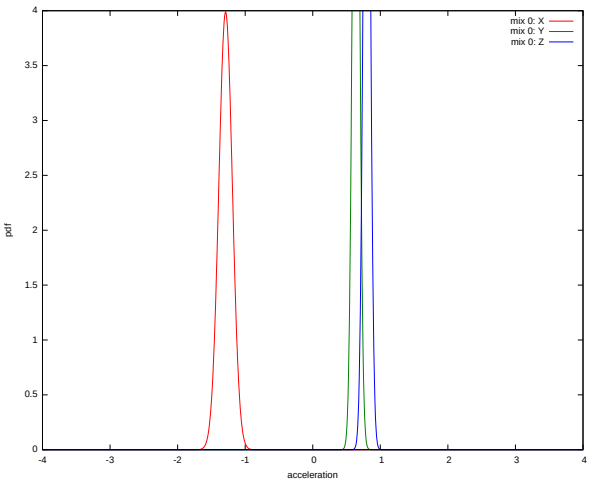


Figure 14 Gaussian probability density functions in state 1 before training for gesture Z

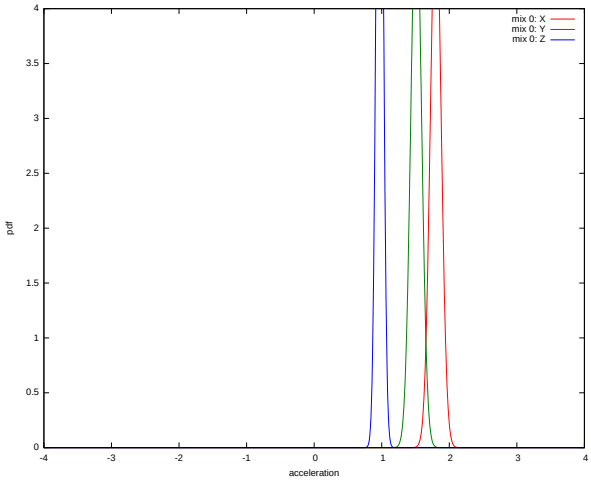


Figure 15 Gaussian probability density functions in state 2 before training for gesture Z

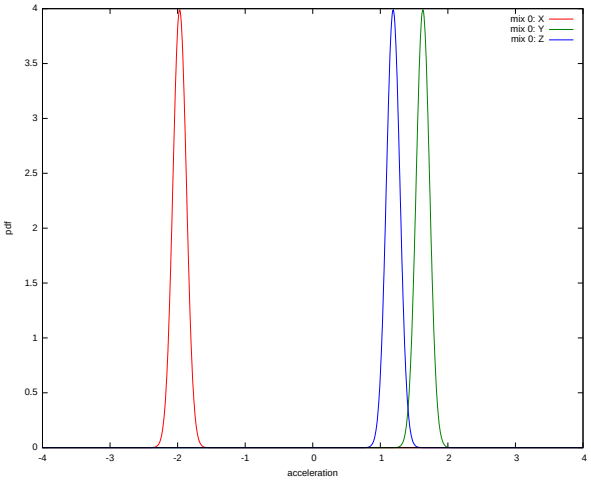


Figure 16 Gaussian probability density functions in state 3 before training for gesture Z

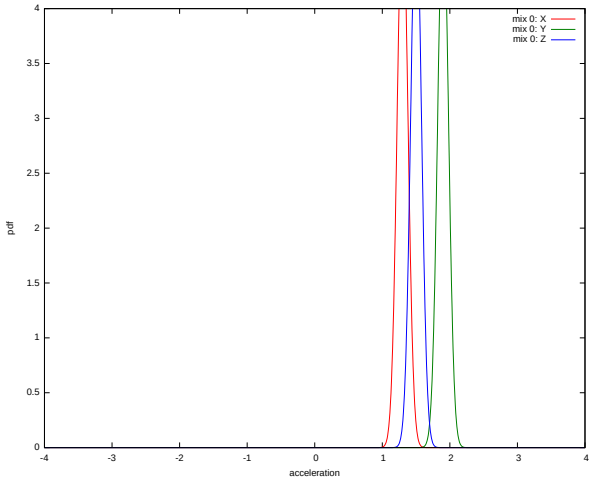


Figure 17 Gaussian probability density functions in state 4 before training for gesture Z

## Training

- `./gesapp.sh --train-model z.hmm`
  - Follow the instructions on the screen on how to train the gesture; it's basically very easy, as you just have to repeat the same gesture again and the hidden Markov model will adapt itself.
- `./gesapp.sh --view-model z.hmm`
  - Figure 20, Figure 21, Figure 22, Figure 23, and Figure 24 show the new estimates for the hidden Markov model, i.e. after training. Figure 11 and Figure 18 are the same because the initial state distribution remains unchanged due to the fact that the model needs to start in state 1 and end in state 5. However, the state transition probability matrix suffers a complete change after it has been re-estimated with the Baum-Welch procedure, as seen in Figure 19 (compare with Figure 12).

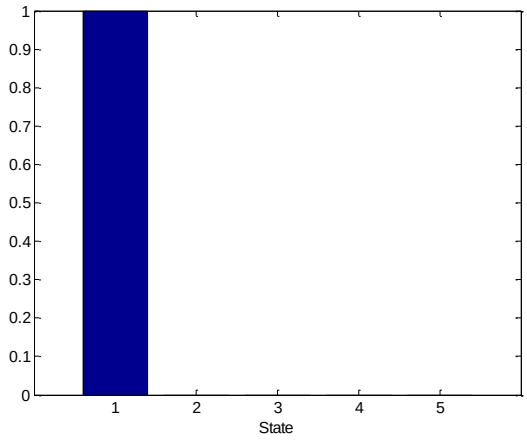


Figure 18 Initial state probabilities after training for gesture Z

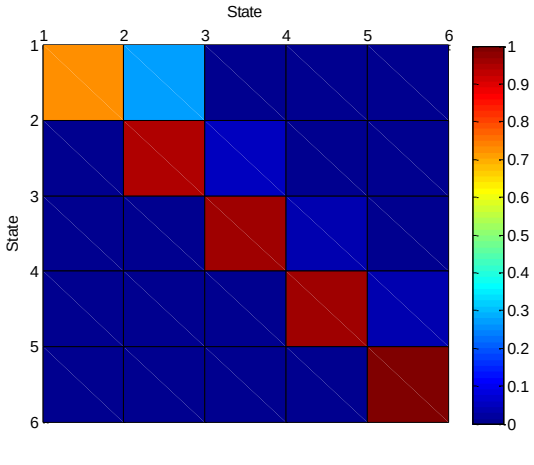


Figure 19 State transition probabilities after training for gesture Z

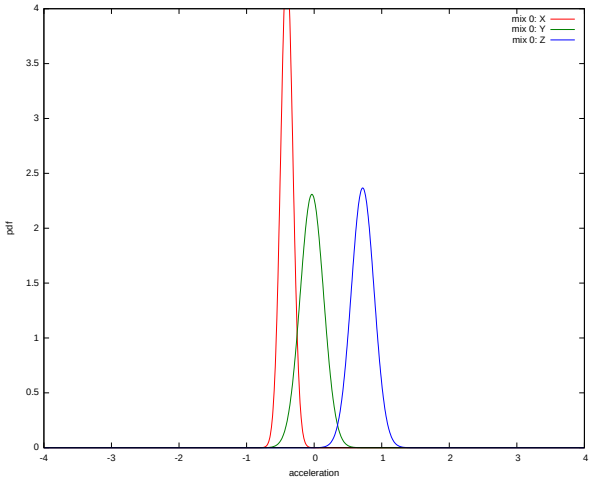


Figure 20 Gaussian probability density functions in state 0 after training for gesture Z

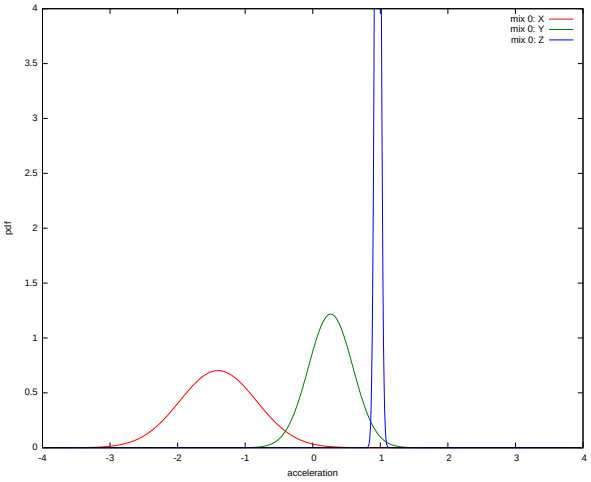


Figure 21 Gaussian probability density functions in state 1 after training for gesture Z

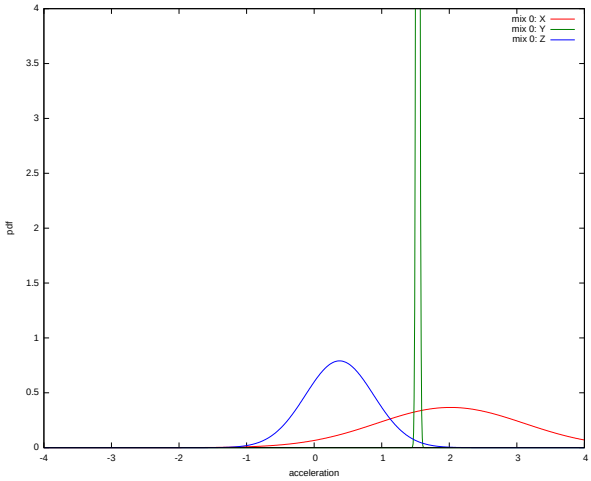


Figure 22 Gaussian probability density functions in state 2 after training for gesture Z

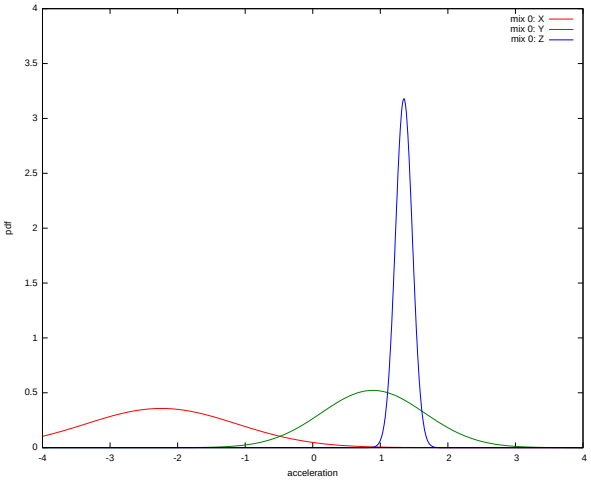


Figure 23 Gaussian probability density functions in state 3 after training for gesture Z

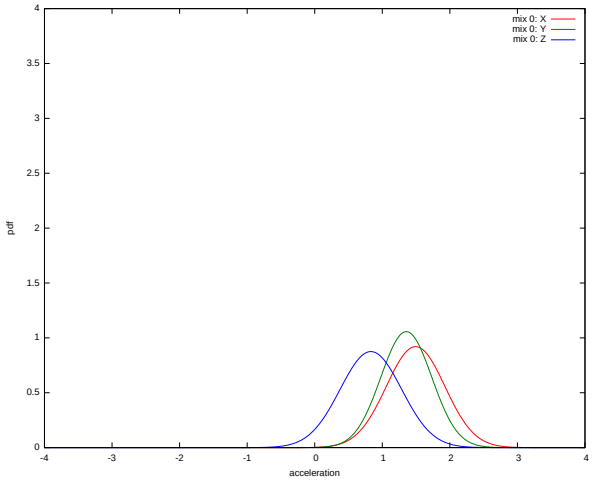


Figure 24 Gaussian probability density functions in state 4 after training for gesture Z

## How to Use a Gesture

The classes of the classifier and the models of the grammar must be specified in a configuration file (amarok.ges) that has the following format:

```
# accelerometer-based gestures configuration file

# classes (two required)
noise       noise.gauss
motion      motion.gauss

# models (minimum two required)
# add a new model like: hmm\tcommand with spaces\tfile_name
hmm   dcop amarok player next right.hmm
hmm   dcop amarok player prev left.hmm
hmm   dcop amarok player setVolumeRelative 50   up.hmm
hmm   dcop amarok player setVolumeRelative -30  down.hmm
hmm   dcop amarok player mute forward.hmm
hmm   dcop amarok player playPause  z.hmm
hmm   dcop amarok player stop triangle.hmm
```

- `./gesphone.sh --load amarok.ges`
  - Make a Z in the air without pressing any buttons (the application is in continuously listening mode) and you'll see something like this:

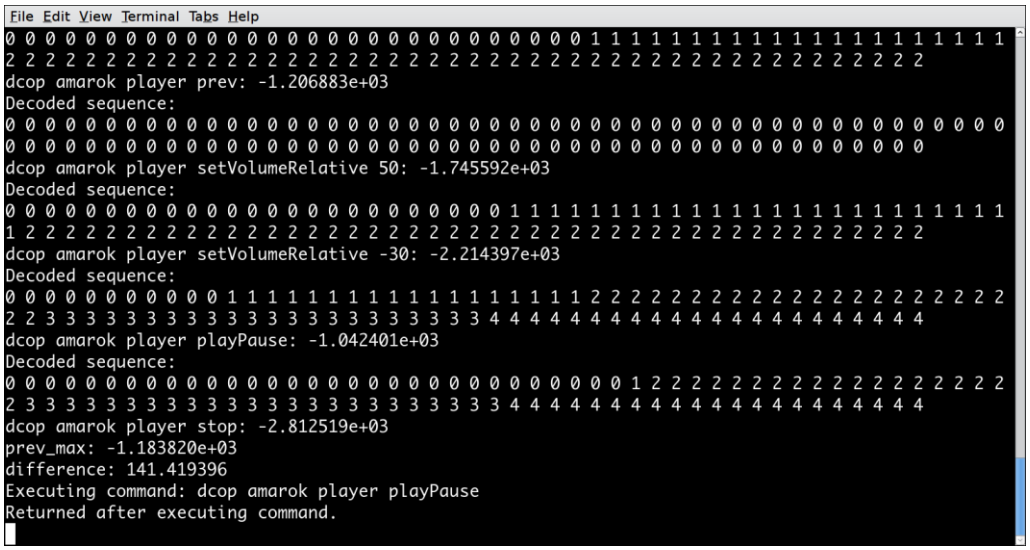


Figure 25 Gesture application that controls Amarok

# Chapter 10. Future Work and Conclusions

Hidden Markov models have well known limitations in practice, but these can be overcome with some workarounds that alter the fundamental assumptions they were built on. Advanced projects on gesture recognition using HMMs are being developed at Microsoft Research (7) and Carnegie Mellon University, Robotics Institute (8).

I will continue to improve the accelerometer-based gesture recognition framework, as my work will be incorporated in the Neo FreeRunner mobile phone which runs on the OpenMoko platform. The next key-point is the inclusion of connected recognition while continuously listening for gestures. Connected recognition will be implemented with the time-synchronous Viterbi beam search and will run on a composite hidden Markov model, which is composed of the union of all the states from the other hidden Markov models (i.e. a hidden Markov model of all the gestures).

To achieve better results, we have to somehow help the decoder prune unpromising models. One variant is to associate time durations to each model and automatically discard models that don't fit the time duration of the observed motion sequence. Also, the number of appearances of each state should be proportional with the length of the observation sequence.

Hidden Markov models are widely used in state-of-the-art speech recognizers like Carnegie Mellon University's Sphinx. Hidden Markov models are a good choice for modeling an accelerometer if understood and implemented correctly. In my humble opinion, one can spend its entire lifetime learning and researching this domain of machine learning techniques – pattern and model recognition. The performance of a hidden Markov model framework greatly depends on its design and its parameters (i.e. probabilities).

# Works Cited

1. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.* **Rabiner, Lawrence R.** 1989. IEEE. pp. 257-286.

2. **Peek, Brian.** Managed Library for Nintendo's Wiimote. *MSDN Coding4Fun.* [Online] 03 14, 2007. http://blogs.msdn.com/coding4fun/archive/2007/03/14/1879033.aspx.

3. **Huang, Xuedong, Acero, Alex and Hon, Hsiao Wuen.** *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development.* Upper Saddle River, New Jersey 07458 : Prentice Hall PTR, 2001. ISBN 0-13-022616-5.

4. *Recognizing Movements of a Handheld Device using Symbolic Representation and Coding of Sensor Signals.* **Flanagan, Adrian, et al.** Oulu, Finland : Nokia, 2002.

5. **Rouse, David Marshall.** Estimation of Finite Mixture Models. *North Carolina State University.* [Online] 2005. http://www.lib.ncsu.edu/theses/available/etd-11282005-140114/unrestricted/etd.pdf.

6. **Jelinek, Frederick.** *Statistical Methods for Speech Recognition (Language, Speech, and Communication).* United States of America : The MIT Press, 1998. ISBN 0-262-10066-5.

7. **Wilson, Andrew and Shafer, Steven.** XWand: UI for Intelligent Spaces. *Microsoft Research.* [Online] April 5, 2003. http://research.microsoft.com/~awilson/papers/CHI%202003%20XWand.pdf.

8. **Wilson, Daniel and Wilson, Andy.** Gesture Recognition Using the XWand. *Carnegie Mellon University, Robotics Institute.* [Online] April 2002. http://www.cs.cmu.edu/~dwilson/papers/xwand.pdf. Technical Report CMU-RI-TR-04-57.

# Bio

Paul-Valentin Borza was born in Sebeş, Alba, Romania on April 30, 1986. From 2005 to 2007 he participated in the Software Design Invitational, Imagine Cup where he won the National Finals two years in a row and represented Romania to the Central and Eastern Europe Regional Finals in Maribor, Slovenia and to the Worldwide Finals in Seoul, Korea. He writes and publishes articles on MSDN Coding4Fun whenever he wishes. Paul was invited in 2005 in the Microsoft Academic Program and is a Microsoft Student Partner. He is now wrapping up his sixth semester at the Babeş-Bolyai University, Faculty of Mathematics and Computer Science and is developing accelerometer-based gestures for OpenMoko Inc. within Google Summer of Code 2008. He will spend his summer holiday as an intern at the Jozef Stefan Institute in Ljubljana, Slovenia.