An Ant Learning Algorithm for Accelerometer-based Gesture Recognition

Sichao Song
master thesis spring 2013
Abstract

Today’s emerging gesture recognition techniques have enriched the ways of human machine interaction. With the popularity of smart devices such as iPhone and iPod Touch, accelerometer-based gesture recognition for facilitating such interactions is becoming even more pervasive and promising. Accelerometer-based gesture recognition systems have been extensively discussed in many previous related work. Currently, there are several techniques being applied for recognizing gestures, most well-known algorithms are Hidden Markov Model (HMM) and Dynamic Time Warping (DTW). However, they do have shortcomings: 1) HMM requires a sizeable amount of training data, and suffers from the high computational overhead for both training and classification. 2) The processing time of DTW depends on both the length and number of templates.

In this thesis, we introduce a novel gesture recognition algorithm named the Ant Learning Algorithm (ALA), which aims at addressing some of the limitations with the currently two leading algorithms, especially HMM. It takes advantage of the pheromone mechanism from ant colony optimization and uses pheromone tables to represent gestures, which scales well with gesture complexity.

ALA requires minimal training instances and greatly reduces the computational overhead required by both training and classification. The experimental results show that ALA can achieve a high recognition accuracy of over 90% with only one training instance and exhibits good generalization.
# Contents

1 Introduction 11
   1.1 Problem Description 14
   1.2 Contribution 15
   1.3 Chapter Overview 15

2 Background 17
   2.1 Pattern Recognition 17
   2.2 Accelerometer-based Gesture Recognition 18
      2.2.1 Acceleration Data Sensing 18
      2.2.2 Open Sound Control (OSC) 19
      2.2.3 Vector Quantization 22
      2.2.4 The Two Popular Algorithms: Hidden Markov Model (HMM) and Dynamic Time Warping (DTW) 23
   2.3 Generalization and Cross Validation 28
   2.4 Ant Colony Optimization (ACO) 29
      2.4.1 One Application: Automatic Music Composition 32

3 The Ant Learning Algorithm (ALA) Based Gesture Recognition System 35
   3.1 System Overview 35
   3.2 Acceleration Data Acquisition 36
      3.2.1 Retrieving Acceleration Data from OSC Packets 37
   3.3 Data Pre-processing 38
   3.4 Vector Quantization 39
   3.5 Ant Learning Model 42
      3.5.1 Features in Gesture 44
      3.5.2 Pheromone Mechanism and Pheromone Table Update 45
   3.6 Classification Methods 49

4 Experiments and Results 55
   4.1 Experiments Setup 56
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2 Results</td>
<td>57</td>
</tr>
<tr>
<td>4.2.1 Trajectory Analysis</td>
<td>57</td>
</tr>
<tr>
<td>4.2.2 Filtering</td>
<td>60</td>
</tr>
<tr>
<td>4.2.3 Parameter Tuning</td>
<td>60</td>
</tr>
<tr>
<td>4.2.4 Cross Validation</td>
<td>63</td>
</tr>
<tr>
<td>4.2.5 Effect of Number of Training Instances</td>
<td>64</td>
</tr>
<tr>
<td>4.2.6 Effect of Starting Point</td>
<td>65</td>
</tr>
<tr>
<td>4.2.7 Execution Time (training + recognition)</td>
<td>70</td>
</tr>
<tr>
<td>4.2.8 User-dependent Tests with One-Instance Training</td>
<td>71</td>
</tr>
<tr>
<td>5 Potential Applications</td>
<td>73</td>
</tr>
<tr>
<td>6 Conclusion and Future Work</td>
<td>77</td>
</tr>
<tr>
<td>6.1 Conclusion</td>
<td>77</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>78</td>
</tr>
<tr>
<td>A Java Source Codes for Ant Learning Algorithm (ALA)</td>
<td>83</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A gesture expresses the meaning “Okey”.</td>
<td>12</td>
</tr>
<tr>
<td>1.2</td>
<td>A hand movement describing a square.</td>
<td>13</td>
</tr>
<tr>
<td>2.1</td>
<td>Axes of the iPod Touch 4’s built-in accelerometer LIS331DLH.</td>
<td>19</td>
</tr>
<tr>
<td>2.2</td>
<td>Part of the acceleration waveforms of a gesture clockwise circle.</td>
<td>20</td>
</tr>
<tr>
<td>2.3</td>
<td>Part of the acceleration waveforms of a gesture clockwise circle, with the input device held up-side down.</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>OSC Packets transmission.</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>HMM with 3 states and observation set {rain, clouds, sun, fog, snow}.</td>
<td>24</td>
</tr>
<tr>
<td>2.6</td>
<td>Matching between two time series by using Euclidean distance (left) and DTW (right).</td>
<td>26</td>
</tr>
<tr>
<td>2.7</td>
<td>Two time series and their cost matrix. The red line indicates the optimal warping path in this matrix.</td>
<td>27</td>
</tr>
<tr>
<td>2.8</td>
<td>Ants foraging behavior in nature.</td>
<td>30</td>
</tr>
<tr>
<td>2.9</td>
<td>The process of finding a good solution by using ACO, regarding the TSP problem. [4]</td>
<td>31</td>
</tr>
<tr>
<td>2.10</td>
<td>Example of graph with 7 notes: (A^{(1)}, \ldots, G^{(7)}) [5].</td>
<td>32</td>
</tr>
<tr>
<td>3.1</td>
<td>System operational pipeline for the ant learning algorithm.</td>
<td>36</td>
</tr>
<tr>
<td>3.2</td>
<td>The iPod Touch used in the project, with the HyperMusic application running.</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>Distribution of 14 characteristic vectors.</td>
<td>39</td>
</tr>
<tr>
<td>3.4</td>
<td>Integers from 1 to 14 (characteristic states) are assigned to each centre.</td>
<td>40</td>
</tr>
<tr>
<td>3.5</td>
<td>An incoming acceleration vector stream (left) and the vector stream after quantization (right).</td>
<td>42</td>
</tr>
<tr>
<td>3.6</td>
<td>Example of a graph with 14 states. States from 10 to 14 are omitted for simplification. The artificial ant can move from one state to another or stay in the same state.</td>
<td>43</td>
</tr>
<tr>
<td>3.7</td>
<td>Flow chart for a newly generated pheromone table.</td>
<td>44</td>
</tr>
<tr>
<td>3.8</td>
<td>Two features of the incoming acceleration vector stream.</td>
<td>45</td>
</tr>
</tbody>
</table>
3.9 Visualization of two pheromone tables: “Square” and “Circle”. . . 46
3.10 The pheromone table is generated according to the incoming ac-
celeration vector stream. . . . . . . . . . . . . . . . . . . . . . . . 47
3.11 Two different acceleration vector streams with the same number
of immediate repetitions of each characteristic state (↑, →, ↓ and
←) are transformed into two identical pheromone tables. . . . . . 51
3.12 Two different acceleration vector streams with the same number
of transition points between different characteristic vectors are
transformed into two identical pheromone tables. . . . . . . . . . 52
3.13 Pheromone tables adapted to different incoming acceleration vec-
tor streams, based on the hybrid classifier. . . . . . . . . . . . . 53
4.1 Working with the ALA-based system in practice. . . . . . . . . . 55
4.2 The five simple gesture types considered. . . . . . . . . . . . . 56
4.3 Trajectories of the two gestures: Square (above) and Circle (below). 58
4.4 Two gesture samples of “Circle”. The second gesture can not be
correctly recognized by ALA. . . . . . . . . . . . . . . . . . . . . . . . . 59
4.5 Average recognition results vs. Number of training instances (ALA) 64
4.6 The four gestures with starting point SP1. . . . . . . . . . . . . 65
4.7 The four gestures with starting point SP2. . . . . . . . . . . . . 65
4.8 Average recognition results vs. Number of training instances,
tested by gestures with starting points SP0, SP1 and SP2. (ALA) . 68
4.9 Average recognition results vs. Gesture types, tested by gestures
with starting points SP0, SP1 and SP2. (ALA) . . . . . . . . . . . . 68
4.10 Average recognition results vs. Number of training instances,
tested by gestures with starting points SP0 (above), SP1 (middle)
and SP2 (bottom). (ALA, HMM and DTW) . . . . . . . . . . . . 69
4.11 Average execution time vs. Number of training instances. (ALA,
HMM and DTW) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
4.12 Average recognition rates of the five gestures. (ALA) . . . . . . 72
4.13 Average recognition rates of the six tests. (ALA) . . . . . . . . 72
5.1 A gesture “Square” is used to trigger a note. . . . . . . . . . . . . 73
5.2 Two acceleration waveforms corresponding to status “Walking”
(above) and “Running” (below), respectively. . . . . . . . . . . . 75
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Features of OSC. [6]</td>
<td>21</td>
</tr>
<tr>
<td>2.2</td>
<td>A note transition table with 7 notes: A, B, .. , G.</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>Three OSC Messages with OSC Address Patterns followed by the values of acceleration $acc_x$, $acc_y$, and $acc_z$.</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>Implementation of the codebook $Y$.</td>
<td>41</td>
</tr>
<tr>
<td>3.3</td>
<td>The pheromone table with its values normalized.</td>
<td>48</td>
</tr>
<tr>
<td>3.4</td>
<td>The pheromone table that adapted to the leading diagonal distance classifier. All the values except leading diagonal values are set to 0.</td>
<td>50</td>
</tr>
<tr>
<td>3.5</td>
<td>The pheromone table that adapted to the rest-of-table distance classifier. Leading diagonal values are set to 0.</td>
<td>51</td>
</tr>
<tr>
<td>4.1</td>
<td>Experiments outline.</td>
<td>57</td>
</tr>
<tr>
<td>4.2</td>
<td>Reduction of Incoming Acceleration Vectors with Threshold Value $\Delta = 0.6g$.</td>
<td>60</td>
</tr>
<tr>
<td>4.3</td>
<td>Results for parameter tuning: pheromone value $\Delta \tau$, evaporation rate $\rho$ and classification methods. The results are averaged across the five gestures: square, circle, triangle, eight and roll.</td>
<td>61</td>
</tr>
<tr>
<td>4.4</td>
<td>Results for cross validation on the five gesture types: Square, Circle, Triangle, Eight and Roll. (ALA, HMM and DTW)</td>
<td>63</td>
</tr>
<tr>
<td>4.5</td>
<td>Evaluation results on how evaporation rate will affect the recognition accuracy, tested by gestures performed with different starting points.</td>
<td>67</td>
</tr>
</tbody>
</table>
Perface

I would like to take this opportunity to thank my supervisors: Arjun Chandra and Prof. Jim Tørresom. Without their illuminating instruction and insightful comments, this project would not have been possible.

I would further like to show my sincere appreciation to Arjun and Prof. Jim without whose meticulous revision my paper would not be accepted by CEC2013 (2013 IEEE Congress on Evolutionary Computation). This thesis is heavily based on the paper.

I would also like to thank all the people studying and working in the ROBIN research group for their support and friendship. I was quite lonely at the beginning of my study, and their sincere friendship makes me feel warm.

Thanks are also due to all my friends in Norway who have given me great support. It is always cheerful to be with them. Besides, I would especially express my gratitude to Kaihua Li who has given me many sound comments on my English witting, which helped a great deal to the final thesis.

Special thanks goes to my girl friend for her absent in the past 20 years so that I can focus on my study.

Finally, I would like to thank my family members, especially my parents, without whose support and love I may not have gone this far.
Chapter 1

Introduction

Currently, most of our interactions with computers are performed with traditional keyboards and mouses. For example, during my writing of this thesis, all the words are recorded by the laptop via its keyboard, and other actions such as scrolling down the window of the PCT-ex\(^1\) is performed via a mouse connected to the laptop. However, with the help of new technologies, a variety of emerging methods for interaction with computers are introduced into our daily life. It is common now for us to turn pages of an e-book by sweeping fingers left and right on the screen of an iPad, dialing a friend via the virtual keyboard of a smart-phone, or playing some console games like Nintendo Wii\(^2\). Likewise, gesture-based interfaces, with the advantage to enrich the interaction experience with computers, especially for people with disabilities, are also expected to be important for the next-generation human machine interaction (HCI) solutions.

A gesture is a form of non-verbal communication in which people use movements of their hands, face or other parts of their bodies to convey information when interacting with others. Thus, gestures are considered as a natural way which can be utilized in human-computer interaction. Fig. 1.1 shows a widely known gesture which expresses the meaning “Okey”.

Gesture recognition techniques have been extensively discussed in recent decades. Basically, they can be used in various categories of applications:

- **Expressive communication**: Gestures are employed to enrich the ways of communication. For instance, users are allowed to control their computers through the use of facial gestures based on the emotion recognition techniques [7].

- **Health**: Gesture recognition can be used to assist in patient rehabilitation or help the handicapped users in their interactions with computers.

---

\(^1\) An integrated LATEX environment, see [http://www.pctex.com/](http://www.pctex.com/) (last access: 12.03.2013).

• **Interfaces:** Types of interfaces can be enriched. For instance, interactive game technology offers new gaming experience which allows people to play games by using their body movements [8].

• **Remote control:** Systems can be operated at a distance by using gestures such as “wave of a hand”.

Gesture recognition systems regarding the above-mentioned applications usually require one or more input devices which are used to track the user’s movements. In general, there are two types of input devices: camera and sensor (usually accelerometer and gyro). Camera-based gesture recognition systems [9, 10] have been studied for many years, and various of applications have been developed based on them. However, one limitation of such systems is that the user’s activities are restricted by the effective range of the camera. As a result, users are not free enough to perform their movements while interacting with these systems. Differing from the camera-based systems, sensor-based gesture recognition systems [11, 12, 13], especially accelerometer-based systems [14, 15, 16], seem to offer users more freedom in their interaction experience with computers, in that users can perform their movements with devices held in their hands or binded to the other parts of their bodies, without worrying about those restrictions that might appear in the camera-based systems.

Since smart devices, such as iPhone and iPod Touch, are becoming popular in our daily life, using such devices for achieving gesture recognition is becoming promising and easy to be integrated in our daily life. In general, all these devices have a few built-in sensors, for instance, a 3-axis accelerometer. They provide the possibilities for interacting with various applications using accelerometer-based gesture recognition techniques. Hence, using such techniques for facilitating human-machine interactions is becoming a hot research topic.
In accelerometer-based gesture recognition, gesture refers to an action or motion with meanings, for example, a hand movement describing a square, see Fig. 1.2. Such gestures are captured by an accelerometer when sensing the user’s hand movement, and are further represented by acceleration vector stream. Once these acceleration vector streams are obtained, they can be used for gesture recognition purposes. The core part of a gesture recognition system is the algorithm which learns the features within the input gesture acceleration streams and classifies them. Therefore, associated algorithms, also known as pattern recognition algorithms, need to be employed in the goal of classifying those gestures.

Figure 1.2: A hand movement describing a square.

Accelerometer-based gesture recognition methods have been reviewed in many publications [16, 17, 18, 19, 14, 15, 20, 21, 3, 22]. In these works, two popular algorithms, Hidden Markov Model (HMM) and Dynamic Time Warping (DTW), have been most frequently discussed. HMM is a statistical generative model for analyzing time varying signals, and DTW is the algorithm for measuring similarity between two signals which can vary in time and speed. Despite that the two algorithms have proved to be effective for gesture recognition, their limitations hinder the progress of accelerometer-based gesture recognition techniques.

Ant Colony Optimization (ACO), a probabilistic technique that finds good solutions for computational problems [23], mimics the foraging behavior of ants in nature. Martens et al. in [24] proposed a new ant-based classification approach named AntMiner+. In their work, they showed that ACO can be applied to extract
rule-based classifiers in data mining. The method took advantage of the collective behavior of ants. C. Guéret et al. [5, 25] showed an application of automatic music composition [26] that ACO can be adapted for. They studied how to generate music by simulating moves of artificial ants. Although ACO is mostly applied for solving discrete optimization problems, both of their researches bring about a new thinking: “Is ACO capable of settling other problems?” Inspired from this, we propose our question: “Can ACO be adapted for gesture recognition?”

In this thesis, we propose a novel gesture recognition algorithm, the Ant Learning Algorithm (ALA), which is inspired from the ACO meta-heuristic that is the transition rule based on the so-called pheromone mechanism, see Section 2.4. Based on some of the current research questions in the field of gesture recognition, our study aims at addressing some limitations existing in the current two leading algorithms: HMM and DTW. In addition, by presenting some ideas of potential applications which might be achieved based on ALA in Chapter 5 we show that ALA could not only be applicable for gesture recognition purposes, but could also be applied for other research interests.

1.1 Problem Description

While both HMM and DTW have proved to be effective for accelerometer-based gesture recognition, certain limitations of these algorithms have not been satisfactorily addressed in the literature. These limitations are:

- HMM suffers from high time complexity for both training and recognition [27, 21], and requires a sizable amount of training samples [28, 3].
- The processing time of DTW depends on both the length and number of gesture data (templates) [22].

Although the classical leading method, Hidden Markov Model, has proved to be effective for gesture recognition, it still has several shortcomings. HMM requires a large amount of training samples, and their computational overhead for both training and inference is high. Consequently, using HMM-based gesture recognition systems can be quite difficult in several scenarios. For example, it requires quite a long time and extensive physical work to build a gesture library with the necessary amount of gestures. As a result, user fatigue can greatly reduce the quality of training gesture instances, and lengthy training can be very tedious as well [27]. Moreover, such systems could perform weakly for real-time gesture recognition on low-end devices.

We note that dynamic time warping is a very effective technique for recognizing accelerometer-based gestures [21, 14] and has been extensively studied
recently. It requires low computational overhead and minimal training samples, which can be performed on a mobile device with promising performance. However, both the length and number of templates affect the classification time.

1.2 Contribution

This research provides mainly two contributions to the field of gesture recognition:

- **ALA requires only one training instance:** We show that while ALA has a higher recognition accuracy when more than one training instance are used, it can still achieve a satisfying recognition rate with only one training instance.

- **ALA requires extremely low computational overhead:** We show that ALA requires much shorter execution time (training + recognition) as compared to both HMM and DTW.

Part of the work in this thesis has been published and is to appear [29] in the proceedings of the IEEE Congress on Evolutionary Computation (CEC 2013) to be held in Cancun, Mexico, between 20-24 June 2013.

1.3 Chapter Overview

This thesis is composed of six chapters and one appendix. Description of remaining chapters:

**Chapter 2 Background**

Much of the background knowledge of the ALA-based gesture recognition system is considered, as well as some brief introduction to both HMM and DTW.

**Chapter 3 Methodology and Approach**

The principle of ALA-based system is described in detail. The system consists of four modules: data pre-processing, vector quantization, ant learning model and classification.

**Chapter 4 Experiments and Results**

The ALA-based system is evaluated in a systematic way. In total eight experiments were conducted and the results are presented accordingly. For some experiments, comparisons with both HMM and DTW are also provided.
Chapter 5. Potential Applications

Ideas of potential applications using ALA are discussed.

Chapter 6. Conclusion and Future Work

The entire work is concluded, and the main future work is presented.

Appendix

Source code of some important methods of ALA is provided.
Chapter 2

Background

Chapter Overview

This chapter offers some background knowledge of the techniques used in the thesis. First of all, concept of pattern recognition is introduced. Next, some theories and important things regarding an accelerometer-based gesture recognition system are offered. Specifically, basic principles of both Hidden Markov Model and Dynamic Time Warping are explained, because a comparison is made between the recognition performance of these two algorithms with that of ALA in the experiments. Finally, the basic principle of Ant Colony Optimization algorithm is stated, and a previous work which studied how to generate music by using artificial ants is introduced, as ALA was inspired from this work.

2.1 Pattern Recognition

The primary task for gesture recognition is pattern recognition. Classification, as an example of pattern recognition, assigns each input value to a set of labels (also known as classes). In this project, a gesture performed by the user is the expected input value, and a label which contains a gesture type is assigned to the input gesture as the result of classification.

Pattern recognition techniques have been widely applied in many different areas. Optical character recognition (OCR) [30, 31] is a classic application of pattern classifiers. Handwriting alphabets and numbers can be correctly recognized based on the features such as the character’s strokes, speed, pressure and so on. Another classic example is speech recognition (SR) [32, 33], which allows the system to transcribe spoken words into texts. In general, the above-mentioned systems need to be trained first by letting the user read some given texts to the systems. The user’s voice is recorded and analyzed to fine tune the recognition of his or her speech. Besides, within the domain of medical science, computer-aided di-
agnosis (CAD) \[34\] systems use pattern recognition methods to analyze patient’s symptom, thus offering the doctor suggestions of giving proper treatments to the patient.

There are many methods for pattern recognition purposes. Support Vector Machine (SVM) \[19\] is a supervised learning model which is used for analyzing and recognizing features. It is generally applied for classification and regression. Another well-known classification method is the so-called Naive Bayes classifier \[3\], which is a probabilistic model based on Bayes’ theorem. The classification result which has the highest probability is assigned to the input data. Both of these two methods are usually used for gesture recognition with associated algorithms such as Hidden Markov Model (HMM) and Dynamic Time Warping (DTW).

In accelerometer-based gesture recognition, features are hidden in the raw acceleration data. In Section \[3.5.1\] we present the features used in our ALA-based gesture recognition system. These features are simply different combinations of acceleration vectors and exhibit distinctions in different gesture types.

### 2.2 Accelerometer-based Gesture Recognition

#### 2.2.1 Acceleration Data Sensing

**Sensing Device**

In this project, an iPod Touch 4\(^{th}\) generation is chosen as the input device. It has a built-in 3-axis accelerometer, the LIS331DLH micro electro-mechanical system (MEMS) \[2\], which allows us to sense accelerations in all three dimensions. The LIS331DLH has dynamical user selectable full scales of ±2g/ ±4g/ ±8g, and is capable of measuring accelerations with output data rates from 0.5 kHz up to 1 kHz \[35\]. The directions of the incoming acceleration vectors are set based on the axes of the iPod Touch’s built-in accelerometer, see Fig. 2.1.

**Significance of Keeping the Input Device’s Orientation Consistant**

In accelerometer-based gesture recognition, the first step is sensing the user’s hand movements (gestures). The movement data is then translated into accelerations. Basically, the embedded 3-axis accelerometer senses acceleration vectors in \(x\), \(y\) and \(z\) directions expressed in the unit of gravity \(g\).

---

1. Also referred as the Maximum-likelihood Estimation (MLE)
2. Microelectromechanical systems (MEMS) is the technology of very small devices. It merges at the nano-scale into nanoelectromechanical systems (NEMS) and nanotechnology. See \url{http://en.wikipedia.org/wiki/Microelectromechanical_systems} (last access: 06.04.2013).
Fig. 2.1 shows part of the acceleration waveforms of moving the hand in a clockwise circle. Three different line types, which are the solid line, dot-square line and dashed-plus line, represent the acceleration in $x-$, $y-$ and $z-$ direction accordingly. Different gestures exhibit different acceleration waveforms.

One important thing we shall be ware is that the acceleration waveforms are sensitive to the orientation of the accelerometer. Fig. 2.2 shows the same part of the acceleration waveforms of moving the hand in a clockwise circle by the same person, but with the input device (the iPod Touch) held up-side down. In this case, although the acceleration waveform in $y-$ axis is roughly the same as that in Fig. 2.2, the other two waveforms are inverted accordingly. This highlights the fact that tilting the input device leads to different measurements.

The two waveforms shown above address the question of how orientations of the input device could affect the acceleration data corresponding to the gestures. Therefore, all experiments conducted in this project were performed with the sensing device held in the same orientation.

2.2.2 Open Sound Control (OSC)

Open Sound Control (OSC) is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology [6]. Table 2.1 provides a couple of features of OSC. OSC
Figure 2.2: Part of the acceleration waveforms of a gesture clockwise circle.

Figure 2.3: Part of the acceleration waveforms of a gesture clockwise circle, with the input device held up-side down.
has been widely applied in many fields, including new computer-based interfaces for musical expression, robotics and other applications. In this project, the acceleration data transmission is implemented based on the OSC protocol.

### Features:

Open-ended, dynamic, URL-style symbolic naming scheme  
Symbolic and high-resolution numeric argument data  
Pattern matching language to specify multiple recipients of a single message  
High resolution time tags  
“Bundles” of messages whose effects must occur simultaneously  
Query system to dynamically find out the capabilities of an OSC server and get documentation

| Table 2.1: Features of OSC. [6] |

Basically, the unit of transmission of OSC is an OSC Packet. The application that sends OSC Packets is the OSC Client, which in our case is the iPod Touch. The application which receives OSC Packets is the OSC Server, which in our case is the laptop. Fig. [2.4] gives the overview of the communication between an iPod Touch and a laptop.

![OSC Packets transmission](image)

**Figure 2.4: OSC Packets transmission.**

An OSC Packet consists of either an OSC Message or an OSC Bundle. Particularly, an OSC message consists of an OSC Address Pattern, an OSC Type
Tag String[^3] and zero or more OSC Arguments, while an OSC Bundle consists of the OSC-string “#bundle”, an OSC Time Tag and zero or more OSC Bundle Elements. It should be noted that the content of an OSC Bundle Element is either an OSC Message or an OSC Bundle, meaning that an OSC Bundle can be defined recursively.

In the OSC Server, a set of OSC Methods, which are the destinations of OSC Messages to be received, are predefined. OSC Methods are arranged in tree structures, with leaves of the trees being the OSC Methods and branches being the OSC Containers (an OSC Container is composed of an OSC Address), and can be retrieved by matching with the OSC Addresses.

For instance, an OSC Message example[^6]:

```
/oscillator/4/frequency 440.0
```

consists of an OSC Address “/oscillator/4/frequency” followed by a floating point number 440.0 as the OSC Argument. The OSC Address means: the root of the tree contains an OSC Container with the name “oscillator”, followed by an OSC Container with the name “4”, followed by an OSC Method with the name “frequency”. By matching the OSC Message with the OSC Address, the value 440.0 is then retrieved.

### 2.2.3 Vector Quantization

Direction of the raw acceleration vectors sensed by the sensing device varies in three dimensions. As a result, gestures that represented by these acceleration vectors usually have a very high complexity of data representation. Such gesture data can not be processed by ALA with efficiency. Therefore, the method vector quantization is applied in the aim of reducing the complexity of the gesture data representation and at the same time has a little information loss.

In accelerometer-based gesture recognition, a vector quantizer \( Q \) maps the incoming 3-dimensional acceleration vectors \( \text{acc}^n = (\text{acc}^n_x, \text{acc}^n_y, \text{acc}^n_z) \) onto a finite set of vectors \( Y = \{y^i : i = 1, 2, ..., k\} \):

\[
Q : \mathbb{R}^3 \mapsto Y
\]

Each vector \( y^i = (y^i_x, y^i_y, y^i_z) \) is called a characteristic vector, and the set of all characteristic vectors \( Y \) is called the codebook. \( k \) is the size of the codebook. This mapping is done according to the so-called nearest-neighbor algorithm, meaning that each acceleration vector \( \text{acc}^n \in \mathbb{R}^3 \) is mapped onto its nearest characteristic vector in the codebook:

\[
Q(\text{acc}^n) = y^i, \text{if } d(\text{acc}^n, y^i) \leq d(\text{acc}^n, y^j) \forall i \neq j
\]

[^3]: The OSC Type Tag String can be omitted.
where \( d(,,) \) denotes the distance function based on Euclidean distance:

\[
d(\text{acc}, y) = \sqrt{(y_x^i - \text{acc}_x^i)^2 + (y_y^i - \text{acc}_y^i)^2 + (y_z^i - \text{acc}_z^i)^2}
\]  

(2.1)

Two problems need to be solved regarding vector quantization: 1) The number of characteristic vectors, which is the size of the codebook, needs to be decided upon. This is important and can affect the recognition result significantly, as using a small set of characteristic vectors may result in losing of essential information of a gesture, while using a large set of characteristic vectors may cause the loss of distinctions in the features. 2) The selection of characteristic vectors should also be taken into account.

### 2.2.4 The Two Popular Algorithms: Hidden Markov Model (HMM) and Dynamic Time Warping (DTW)

There are many studies on accelerometer-based gesture recognition systems which have used Hidden Markov Model in their approaches [36][16][37][20]. Pylvänäinen et al. in [36] used continuous HMMs to recognize gestures. In their work, they focused on demonstrating effects of data quality on recognition performance, which included both gesture sampling rate and vector quantization. Schlömer et al. in [16] estimated the performance of both left-to-right and ergodic HMM on 5 simple gestures, given the classic recognition pipeline. A Wii-controller was used as the input device. In their work, they evaluated effects of codebooks with different sizes \( k = 8, 14 \) and 18, respectively. They showed that, using a codebook with a size \( k = 14 \), their algorithm performed best.

Dynamic time warping has been extensively studied recently. Liu et al. in [14] proposed an efficient recognition algorithm named uWave. DTW is the core of their algorithm. They showed that DTW is effective in coping with limited training data and amenable to implementation on resource-constrained platforms. Niezen et al. in [21] evaluated recognition performance, regarding computational efficiency, recognition accuracy and storage efficiency with three popular gesture recognition techniques: Hidden Markov Model, artificial neural networks and dynamic time warping. Their experimental results showed that DTW is very effective for accelerometer-based gesture recognition, and it requires very low computational overhead and minimal training efforts.

The following theories of both HMM and DTW are described in greater detail in [32] (HMM) and [38] (DTW).
Hidden Markov Model (HMM)

A hidden Markov model is a statistical Markov model with extra unobserved (hidden) states. It can be considered as an extended version of Markov chain models, and provides the probability that a given observation (e.g. acceleration vector stream) can be generated by the model. Basically, a hidden Markov model can be denoted as $\lambda = \{S, A, B, \pi, V\}$. Fig. 2.5 describes a hidden Markov model with 3 states and observation set \{rain, clouds, sun, fog, snow\}. Based on the model, brief explanations of each component in the model are given:

- $S = \{1, 2, 3\}$: the set of 3 hidden states. This set is unobservable.
- $A = \{a_{ij}\}$: the state transition probability matrix, where $a_{ij} = P(S_j|S_i)$, $1 \leq i, j \leq 3$.
- $B = \{b_{jk}\}$: the emission probability matrix, where $b_{jk} = P(v_k|S_j)$, $1 \leq k \leq 5$.
- $\pi = \{\pi_j\}$: the set of initial state probabilities, where $\pi_j = P(S_j)$.
- $V = \{v_k\}$: the set of observations. The observation set is \{rain, clouds, sun, fog, snow\}.
HMM-based gesture recognition systems are usually divided into two parts: HMM trainer and HMM recognizer. The HMM trainer creates a hidden Markov model for each gesture type, which is trained based on multiple training samples. The HMM recognizer evaluates how well a newly input gesture data matches the available trained gesture models. More precisely, it calculates the probability that an observation sequence (acceleration vector stream) is generated by a given trained gesture model. However, these probabilities can not be directly used for classification, because the maximum probabilities of each gesture model are extremely diverging [3]. Thus, Bayes’ rule is applied to give the correct classification results. A brief explanation of how gestures can be classified with Bayes’ rule is given in the following.

Bayes’ rule can be used for computing the posterior probability of event $B$, $P(B|A)$, based on the prior knowledge about events $P(A)$ and $P(B)$ and the posterior possibility of event $A$, $P(A|B)$. It is given by:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \quad (2.2)$$

A gesture is represented as an observation sequence of length $T$:

$$O = O_1, O_2, ..., O_T \quad (2.3)$$

which is made up of the characteristic states which are the integers mapped to the acceleration vectors by a quantizer, see Section 3.4.

Assuming that we have a gesture library of $N$ hidden Markov models $\lambda_1, \lambda_2, ..., \lambda_N$, each representing a different gesture type. The goal is to find the index $c$ of model $\lambda_i$ that is most likely to generate the observation sequence $O$:

$$c = \arg\min_i P(\lambda_i|O) \quad (2.4)$$

We use Bayes’ rule to calculate $P(\lambda_i|O)$:

$$P(\lambda_i|O) = \frac{P(O|\lambda_i)P(\lambda_i)}{P(O)} \quad (2.5)$$

where $P(O)$ can be calculated given:

$$P(O) = \sum_i P(O|\lambda_i)P(\lambda_i) \quad (2.6)$$

The model $\lambda_c$ with the highest probability $P(\lambda_c|O)$ represents the classified gesture.
Dynamic Time Warping (DTW)

Dynamic time warping is an algorithm which measures similarity between two sequences of data. The sequences can vary in time or speed, thus having different lengths. Essentially, DTW is aiming to find an optimal match between two sequences, usually time series, under certain restrictions. The sequences are warped non-linearly in the time dimension, so their similarities can be measured independent of the variations in the time dimension.

Fig. 2.6 illustrates the matching between two time series by using Euclidean distance and DTW. It shows that distance measures such as Euclidean distance can not be directly applied for matching between time series. In DTW, a cost matrix is built based on the two sequences. The costs are measured by some distance measure, usually Euclidean distance. Fig. 2.7 shows the optimal warping path running along a valley (dark part in the figure, means low cost) in a cost matrix. The overall cost of the optimal warping path is the distance between the two sequences, which can be further denoted as normalized total warping distance.

The measure of similarity is called “template matching”. Input signals are first trained into template\(^4\), one representing each category (e.g. gesture type). These templates are stored as trained data sets. During classification, normalized total warping distances between the newly input signal and all trained templates are computed, and the closest matching template is the category the input signal most likely belongs to. A brief explanation of how normalized total warping distance is

\(^4\)Signals that corresponds to the same category are trained into one template. No training is required if DTW uses only one signal as training sample.
computed between two templates, and how newly input signals are classified, is given in the following. The equations are freely adapted from [38].

In accelerometer-based gesture recognition, gestures are represented by acceleration vector streams. Each vector consists of the $x$-, $y$- and $z$-axis acceleration values, i.e. $a(x, y, z)$. Hence, the acceleration vector streams are three-dimensional time-series. For example, we have two templates $A$ and $B$, where $A$ represents the test gesture and $B$ represents the trained gesture:

$A = a_1, a_2, \ldots, a_{|A|}$ \hspace{1cm} (2.7)

$B = b_1, b_2, \ldots, b_{|B|}$ \hspace{1cm} (2.8)

Given the two templates $A$ and $B$, with respective lengths $|A|$ and $|B|$, one can construct a warping path:

$W = w_1, w_2, \ldots, w_{|W|}$ \hspace{1cm} (2.9)

where the $k$th value of $W$ is:

$W_k = (A_i, B_j)$ \hspace{1cm} (2.10)

Figure 2.7: Two time series and their cost matrix. The red line indicates the optimal warping path in this matrix.
The normalized total warping distance between \( A \) and \( B \) is computed given by:

\[
DTW(A, B) = \min \frac{1}{|W|} \sum_{k=1}^{W} DIST(W_{k_i}, W_{k_j})
\] (2.11)

where \( DIST(W_{k_i}, W_{k_j}) \) is the Euclidean distance between point \( i \) in template \( A \) and point \( j \) in template \( B \), given by \( W_k \):

\[
DIST(i, j) = \sqrt{\sum_{n=1}^{3} (i_n - j_n)^2}
\] (2.12)

An newly input gesture acceleration vector sequence \( X \) can be classified by computing the normalized total warping distance between \( X \) and each of the \( M \) trained gesture templates \( G_m \). The index representing the trained gesture template which has the minimum normalized total warping distance to \( X \) is the classification result \( c \):

\[
c = \arg \min_m DTW(G_m, X)
\] (2.13)

### 2.3 Generalization and Cross Validation

**Generalization**

In machine learning, *generalization* shows a learning algorithm’s ability to provide accurate predictions based on its training data. Essentially, the objective of a learning system is to generalize from its knowledge which has been taught to the system based on the so-called “training” method. The systems usually have to extract some general knowledge from these training data in order to produce useful predictions in new cases. The reason is that such systems will exhibit poor predictive performance if they “memorize” the training gesture data.

In the scenario of gesture recognition, the systems learn the features of different gestures from the training samples, and predict (recognize) a gesture type based on the newly input gesture data. Generalization in this case is the ability of such systems to correctly recognize new gestures.

One important thing here is the selection of features. The generalization performance of a gesture recognition system can be significantly reduced if the pre-selected features are not able to represent the general information of all gesture samples well while be distinctive between different gesture types.
Cross validation

Cross validation is a means by which one measures the generalization performance of a learning algorithm. It is mainly used to evaluate the prediction accuracy of such an algorithm in practice, based on limited data samples. Generally, in each round of cross validation, the sample data is partitioned into two subsets: training set and testing set. Multiple rounds are performed using different partitions in order to reduce the variability.

Cross validation is important in some scenarios where further samples are hazardous and costly to collect. In gesture recognition, there are mainly two issues regarding the gesture samples:

- Collecting a necessary amount of gesture samples costs a lot of efforts.
- Noise is inevitably included in the gesture samples due to human factors, see Section 4.2.1

Thus, cross validation is used to estimate the overall recognition performance of a gesture recognition system.

2.4 Ant Colony Optimization (ACO)

In nature, ants wander randomly for finding food while laying down pheromone trials on their ways, and the pheromones are evaporating gradually. A path is more attractive to ants if it has larger amount of pheromone.

Fig. 2.8 gives an example of the foraging behavior of ants in nature. At the beginning, two ants start to search for food, and each of them selects a different path with equal probability. The ant on the shorter path has a shorter searching time from its nest to the food. After they reach to the food, they dynamically lay down pheromone trails on their ways back to the nest. Since the ant on the shorter path returns to the nest earlier than the other ant, its pheromone trail attracts more ants to follow the same path. As a result, the density of pheromone on the shorter path is higher because more ants passes the path. Eventually, over many iterations, the shorter path is almost selected by all the ants.

Swarm intelligence (SI) studies the collective behavior of its agents that interact locally with their environment [24]. It is inspired by biological systems, such as ant colony or bird flock [39]. Ant Colony Optimization (ACO) [40, 41], as a member of SI systems, employs artificial ants to find good solutions for computational problems.

Fig. 2.9 illustrates the overall process of finding a good solution by using ACO, regarding the so-called Travelling Salesman Problem (TSP). The TSP problem can be denoted as finding the shortest round-trip which requires the agent to visit each
city exactly once, given a set of different cities and the distances between them. It is one of the hottest research problems in computational mathematics. In fact, the first ACO algorithm, also known as the Ant system [42], was aimed to solve the TSP problem. In Fig. 2.9, each point in the graph represents a state, which is a city, that the ants can visit, and each path linking two states can be passed by the ants. A solution is a round-trip that meets all the restrictions (e.g. each city can be only visited once) set by the problem. After some iterations, a good path will be chosen by most of the ants, which is then picked as the finally solution.

In ACO, artificial ants mimic their biological counterparts’ foraging behavior. Basically, a graph is built to represent the problem to be solved at the beginning, where the artificial ants iteratively construct solutions. Meanwhile, pheromone values on the paths corresponding to the solutions are dynamically updated. Path selection is a stochastically procedure based on a transition rule which is a probabilistic model. More precisely, for ant k, the probability $p_{ij}^k$ of moving from state $i$ to $j$ depends on two parameters: 1) the attractiveness of the move $\eta_{ij}$ which is computed by some heuristic indicating the priori desirability of that move and 2)
the quantity of pheromone deposited on the corresponding path $\tau_{ij}$. In general, the $k$th ant moves from state $i$ to $j$ according to:

$$p^k_{ij} = \frac{\tau^\alpha_{ij} \eta^\beta_{ij}}{\sum_{k \in \text{allowed}} \tau^\alpha_{ik} \eta^\beta_{ik}}$$

(2.14)

where $\alpha (\alpha \geq 0)$ and $\beta (\beta \geq 1)$ are the parameters to control the influence of both $\tau$ and $\eta$, respectively. $\tau^\alpha_{ik}$ and $\eta^\beta_{ik}$ represent all the possible transitions for ant $k$ at state $i$.

When an ant reaches a state, it is more likely to choose the path with higher values of both pheromone value $\tau$ and attractiveness $\eta_{ij}$. At each iteration when the ant passes a path, the pheromone value of this path is updated. Pheromone deposits on the paths happens according to:

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{m} \Delta \tau^k_{ij}, \forall i, j \in N,$$

(2.15)

and evaporates as:

$$\tau_{ij} = (1 - \rho) \tau_{ij}, \forall i, j \in N,$$

(2.16)

where $\tau_{ij}$ is the amount of pheromone already present on the path linking states $i$ and $j$, $\Delta \tau^k_{ij}$ is the amount of pheromone laid by ant $k$ on the path connecting $i$ and $j$, $m$ is the population size of the colony, $N$ is the set of states, and $\rho$ is the evaporation rate. It should be noted that $\Delta \tau^k_{ij}$ is 0 if ant $k$ does not pass the path from state $i$ to $j$.

A good solution is eventually found when the ants are converging to the path corresponding to the solution. To summarize, the design of an ACO algorithm can be described in Algorithm 1.
Algorithm 1 Ant colony optimization

while not converging do
    create ants
    let ants search for solutions, based on the above-mentioned transition rule
    update pheromone
    kill ants
end while

2.4.1 One Application: Automatic Music Composition

While the computational model of ACO is primarily used for solving optimization problems, it can be adapted for other applications as well. C. Guéret et al. [5, 25] studied how music can be automatically generated by simulating moves of artificial ants. Fig. 2.10 shows a graph of a note space where states are the notes and paths are transitions between them.

![Example of graph with 7 notes: A(1), ..., G(7)](image)

Figure 2.10: Example of graph with 7 notes: $A^{(1)}, \ldots, G^{7}$ [5]

The system takes advantage of the pheromone mechanism from ACO. In the
simplest case, only one artificial ant is used. The ant moves on the graph independently according to the same transition rule equation \ref{eq:2.14}, where $\eta_{ij}$ is set to $\frac{1}{d_{ij}+1}$ with $d_{ij}$ representing a distance corresponding to the number of half tones between note $i$ and $j$. The pheromone trails are updated based on equations \ref{eq:2.15} and \ref{eq:2.16}. In each iteration, the note selected by the ant is recorded as part of the result. After several iterations, a sequence of notes, which is the score, is generated by the system.

**Pheromone Table**

Regarding graphs such as Fig.\ref{fig:2.10}, a *pheromone table* can be used to record the pheromone values of all the paths. Table \ref{tab:2.2} shows an example of a note transition table used in the above-mentioned music composition application, which is a pheromone table with 7 notes (states) A, B, .., G.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$\tau_{11}$</td>
<td>$\tau_{12}$</td>
<td>$\tau_{13}$</td>
<td>$\tau_{14}$</td>
<td>$\tau_{15}$</td>
<td>$\tau_{16}$</td>
<td>$\tau_{17}$</td>
</tr>
<tr>
<td>B</td>
<td>$\tau_{21}$</td>
<td>$\tau_{22}$</td>
<td>$\tau_{23}$</td>
<td>$\tau_{24}$</td>
<td>$\tau_{25}$</td>
<td>$\tau_{26}$</td>
<td>$\tau_{27}$</td>
</tr>
<tr>
<td>C</td>
<td>$\tau_{31}$</td>
<td>$\tau_{32}$</td>
<td>$\tau_{33}$</td>
<td>$\tau_{34}$</td>
<td>$\tau_{35}$</td>
<td>$\tau_{36}$</td>
<td>$\tau_{37}$</td>
</tr>
<tr>
<td>D</td>
<td>$\tau_{41}$</td>
<td>$\tau_{42}$</td>
<td>$\tau_{43}$</td>
<td>$\tau_{44}$</td>
<td>$\tau_{45}$</td>
<td>$\tau_{46}$</td>
<td>$\tau_{47}$</td>
</tr>
<tr>
<td>E</td>
<td>$\tau_{51}$</td>
<td>$\tau_{52}$</td>
<td>$\tau_{53}$</td>
<td>$\tau_{54}$</td>
<td>$\tau_{55}$</td>
<td>$\tau_{56}$</td>
<td>$\tau_{57}$</td>
</tr>
<tr>
<td>F</td>
<td>$\tau_{61}$</td>
<td>$\tau_{62}$</td>
<td>$\tau_{63}$</td>
<td>$\tau_{64}$</td>
<td>$\tau_{65}$</td>
<td>$\tau_{66}$</td>
<td>$\tau_{67}$</td>
</tr>
<tr>
<td>G</td>
<td>$\tau_{71}$</td>
<td>$\tau_{72}$</td>
<td>$\tau_{73}$</td>
<td>$\tau_{74}$</td>
<td>$\tau_{75}$</td>
<td>$\tau_{76}$</td>
<td>$\tau_{77}$</td>
</tr>
</tbody>
</table>

Table 2.2: A note transition table with 7 notes: A, B, .., G.

The leading diagonal values $\{\tau_{11}, \tau_{22}, ..., \tau_{77}\}$ are the pheromone values of the paths which lead to the notes themselves, and the remaining values are the pheromone values of the paths linking two different notes. When the ant travels in the graph, values in the pheromone table are updated dynamically, depending on the ant’s moves. Based on the transition rule equation \ref{eq:2.14}, the ant selects its future moves by looking up the pheromone table and retrieve the pheromone values accordingly. Therefore, pheromone table is an effective tool to represent a graph, with all the transition information preserved.
Chapter 3

The Ant Learning Algorithm (ALA) Based Gesture Recognition System

Chapter Overview

This chapter presents the ant learning algorithm in detail. In accelerometer-based gesture recognition, each input gesture is represented by an acceleration vector stream. Therefore, a system operational pipeline is needed to pre-process the input gesture data for both training and recognition purposes. The overall system pipeline for gesture data processing and classification is explained, followed by details of each module.

The work presented in this chapter has been published and is to appear [29] in the proceedings of the IEEE Congress on Evolutionary Computation (CEC 2013).

3.1 System Overview

The ALA-based system is shown in Fig. 3.1. It consists of four main modules: data pre-processing, vector quantization, ant learning model and classification.

Once a user starts to perform a gesture, the iPod Touch continuously sends OSC packets over Wifi. The packets are received in a laptop and unpacked to get the acceleration data. After pre-processing and quantization, the gesture is then represented by a sequence of characteristic states. If the gesture is performed for training, a new pheromone table (representing characteristic state transitions) labeled with the corresponding gesture type is produced by the ant learning model and added in the gesture library for future recognition. If the gesture is performed for recognition, first a new pheromone table corresponding to the gesture is produced. This new table is then used in conjunction with the tables in the gesture library by a classifier, giving a recognition result.
3.2 Acceleration Data Acquisition

Many devices can be chosen as the input device for accelerometer-based gesture recognition systems, if these devices have built-in 3-axis accelerometers. For instance, Schlömer et al. in [16] used the Wiimote to sense acceleration data. In this project, we chose an iPod Touch 4th generation as the input device. It has a built-in 3-axis accelerometer that allows us to sense acceleration vectors in all three dimensions. The reasons for us to choose the iPod Touch are:

- It is available in our ROBIN research group.
- An application (named HyperMusic), made by the ROBIN group, is available in the device, offering a simple way to send out sensor data to a nearby computer over WiFi.
- Choosing a hand-held device, such as iPod Touch or iPhone over Wiimote,

1Robotics and Intelligent Systems, Department of Informatics, University of Oslo.
could be seen as more user-oriented. Such devices are becoming ever more popular, thus will probably have huge potential market.

Figure 3.2: The iPod Touch used in the project, with the HyperMusic application running.

The iPod Touch running the HyperMusic application used in the project is shown in Fig. 3.2. Since ALA is implemented in a laptop rather than in the device, it is not necessary to discuss about the processing power of the device in this thesis. However, in order to achieve a gesture recognition system with mobility and flexibility, implementing ALA in such hand-held devices could be considered in the future.

3.2.1 Retrieving Acceleration Data from OSC Packets

In the project, raw acceleration data is transformed into OSC Packets which are then sent over Wifi to a computer nearby. The contents of the OSC Packets are mainly the OSC Messages which consist of the OSC Address Patterns followed by the values of accelerations in all the three directions $x$–axis, $y$–axis and $z$–axis. The format of these OSC Messages is illustrated in table 3.1.

The OSC address “/Motion/Acceleration/x” means: the root of the tree contains an OSC Container with the name “Motion”; this “Motion” container contains an OSC Container with a name “Acceleration”; this “Acceleration” container contains an OSC Method with the name “x”. The acceleration data $acc_x$ can be
### Table 3.1: Three OSC Messages with OSC Address Patterns followed by the values of acceleration $acc_x$, $acc_y$ and $acc_z$.

<table>
<thead>
<tr>
<th>OSC Address Pattern</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Motion/Acceleration/x</td>
<td>$acc_x$</td>
</tr>
<tr>
<td>/Motion/Acceleration/y</td>
<td>$acc_y$</td>
</tr>
<tr>
<td>/Motion/Acceleration/z</td>
<td>$acc_z$</td>
</tr>
</tbody>
</table>

easily retrieved from the OSC Message. Likewise, the other two acceleration values, $acc_y$ and $acc_z$, can also be retrieved. As a result, an acceleration vector $(acc_x, acc_y, acc_z)$ is obtained.

### 3.3 Data Pre-processing

Differing from HMM-based methods, a “directional equivalence” filter [3, 16], which eliminates all redundant vectors that are roughly equivalent to their predecessors, is not required for ALA. On the contrary, it is very important for ALA to record the number of immediate repetitions of each characteristic state in a gesture, for both training and classification. In ALA, the number of immediate repetitions of each characteristic state is considered as one important feature of gestures. Details about selected features in ALA is presented in Section 3.5.1.

#### Idle Filter

In order to eliminate noise that may adversely affect the recognition result, an “idle” filter is used. An idle filter is simply a threshold filter eliminating all the vectors which do not contain much feature information. For instance, when the sensing device is held statically or moved in a relatively slow speed, the sensed accelerations are not strong enough to contribute to the characteristic of the gesture. Thus, these vectors can be eliminated without having significant effects on the recognition accuracy. The algorithm is shown in Algorithm 2.

For implementing the “idle” filter, we set the value of sensitivity $\Delta = 0.6g$ empirically, where $g$ represents the acceleration due to gravity. It filters out all the acceleration vectors $\vec{a}$ for $|\vec{a}| < \Delta$. With the idle filter implemented, the user might be able to make pauses during the performance of a gesture. The reduction performance of the idle filter is evaluated in Section 4.2.2.
Algorithm 2 Idle Filter

\[
\text{if vectorlength} > \text{sensitivity then}
\]

\[
\text{return vector}
\]

\[
\text{else}
\]

\[
\text{return null}
\]

end if

3.4 Vector Quantization

In ALA, pheromone tables are used to represent the acceleration vector transitions. It would be lack of efficiency to build a large size pheromone table, if the raw acceleration vectors, which vary in directions, are used directly. Therefore, a vector quantizer is implemented to reduce the complexity of acceleration data representation.

Schlómer et al. in [16] evaluated effects of codebooks with different sizes \(k = 8, 14\) and 18 respectively. In their work, the \(k\)–mean algorithm is applied with \(k\) being the size of the codebook in order to cluster the acceleration data. Basically, \(k = 8\) was identified empirically for gestures in a 2-dimensional space. All the
characteristic vectors in this codebook was distributed in a plane, thus not suitable for the gestures which are performed in a 3-dimensional space. Besides, they also found that using \( k = 18 \) can result in “over-trained” HMMs. To a conclusion, by using a codebook with size \( k = 14 \), their algorithm was able to achieve a satisfying recognition result. In accordance with their work, we also choose a codebook with size \( k = 14 \), see Fig. 3.3.

The centres can be decided in the following steps:

- **Step 1**: Decide a circle in XY-plane.
- **Step 2**: 8 cluster centres can be decided by distributing them uniformly on the circle.
- **Step 3**: Decide another circle in YZ-plane, intersected orthogonal to the circle in XY-plane.
- **Step 4**: Another 6 centres can then be decided same as in step 2.

Once the distribution of characteristic vectors is decided, each characteristic vector is then labeled with a characteristic state, see Fig. 3.4.

The codebook \( Y \) can be easily implemented by pre-defining a set of characteristic vectors. Table 3.2 shows the 14 characteristic vectors used in this project. Each vector has a length of 1, which is also referred to as the unit vector.
<table>
<thead>
<tr>
<th>Number of state</th>
<th>Characteristic Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1, 0, 0</td>
</tr>
<tr>
<td>1</td>
<td>$\frac{\sqrt{2}}{2}$, 0, $-\frac{\sqrt{2}}{2}$</td>
</tr>
<tr>
<td>2</td>
<td>0, 0, $-1$</td>
</tr>
<tr>
<td>3</td>
<td>$-\frac{\sqrt{2}}{2}$, 0, $-\frac{\sqrt{2}}{2}$</td>
</tr>
<tr>
<td>4</td>
<td>$-1$, 0, 0</td>
</tr>
<tr>
<td>5</td>
<td>$-\frac{\sqrt{2}}{2}$, 0, $\frac{\sqrt{2}}{2}$</td>
</tr>
<tr>
<td>6</td>
<td>0, 0, 1</td>
</tr>
<tr>
<td>7</td>
<td>$\frac{\sqrt{2}}{2}$, 0, $\frac{\sqrt{2}}{2}$</td>
</tr>
<tr>
<td>8</td>
<td>0, $-1$, 0</td>
</tr>
<tr>
<td>9</td>
<td>0, $-\frac{\sqrt{2}}{2}$, $-\frac{\sqrt{2}}{2}$</td>
</tr>
<tr>
<td>10</td>
<td>0, $\frac{\sqrt{2}}{2}$, $-\frac{\sqrt{2}}{2}$</td>
</tr>
<tr>
<td>11</td>
<td>0, 1, 0</td>
</tr>
<tr>
<td>12</td>
<td>0, $\frac{\sqrt{2}}{2}$, $\frac{\sqrt{2}}{2}$</td>
</tr>
<tr>
<td>13</td>
<td>0, $-\frac{\sqrt{2}}{2}$, $\frac{\sqrt{2}}{2}$</td>
</tr>
</tbody>
</table>

Table 3.2: Implementation of the codebook $Y$. 
Figure 3.5: An incoming acceleration vector stream (left) and the vector stream after quantization (right).

Once the codebook is decided, each incoming acceleration vector can then be quantized to the nearest characteristic vector according to the distance equation 2.1. An acceleration vector $\text{acc}^n$ is mapped onto a characteristic state $s$ given by:

$$s = \arg\min_m \sqrt{d(\text{acc}^n, y^i)}$$  \hspace{1cm} (3.1)

where state $m \in M$, $M$ being the set of states $\{1,2,3,\ldots,14\}$. As a consequence, the gesture is eventually represented by a sequence of characteristic states after quantization. It hence significantly reduces the gesture complexity.

To illustrate, Fig. 3.5 shows an example of a vector quantizer $Q$ using a codebook with four characteristic vectors: $Q: \{1 \uparrow, 2 \downarrow, 3 \leftarrow, 4 \rightarrow\}$. As shown in the example, it quantizes all the acceleration vectors within the incoming acceleration vector stream (left) to the four characteristic vectors (right). The quantized vector stream is then represented by a sequence of characteristic states: $\{1,1,1,4,4,4,2,2,2,2,3,3,3,3\}$.

### 3.5 Ant Learning Model

Inspired by [41, 5, 25], we introduce ACO’s pheromone mechanism to our learning model. One major task of the ant learning model is to observe and record characteristic state transitions within each input gesture data.

In ALA, an artificial ant moves in the state graph which is shown in Fig. 3.6 and deposits pheromone on paths linking these states. Differing from ACO, there is no transition rule for the ant to choose its future moves. Instead, the ant moves whenever a new acceleration vector is received. More precisely, whenever ALA
Figure 3.6: Example of a graph with 14 states. States from 10 to 14 are omitted for simplification. The artificial ant can move from one state to another or stay in the same state.
receives a new input acceleration vector, it first quantizes the vector to a corresponding characteristic state, then the ant moves from its current state to the new input state accordingly.

Pheromone tables are used to represent the characteristic state transitions. The size of the pheromone table is decided as $14 - by - 14$, because a codebook with size $k = 14$ is chosen. Values in the table are updated dynamically, depending on the ant’s moves. Each input gesture generates a corresponding pheromone table, and the table is further sent to: 1) Gesture library, if the gesture is performed as a training sample. 2) Classifier, if the gesture is performed for recognition. Fig. 3.7 illustrates the process.

### 3.5.1 Features in Gesture

We stress on two features of gestures:

- **Feature I:** Immediate repetitions of each acceleration vector.
- **Feature II:** Transitions between different acceleration vectors.

A simple example shown in Fig. 3.8 illustrates the two features. The incoming acceleration vector stream consists of four vector types: $\uparrow$, $\downarrow$, $\leftarrow$ and $\rightarrow$. The blue
ellipse indicates the feature of immediate repetitions of the vector ↓ (Feature I), and the red circle indicates the feature of a transition point between vector ↑ and → (Feature II). These two features can be represented by separate sets of values in the pheromone table. There are two types of values in a pheromone table:

- **Leading diagonal values**: These values represent the number of immediate repetitions of each characteristic vector.

- **Rest-of-table values**: These values represent the number of transitions between different characteristic vectors.

One effective way to observe the two features in a gesture is to visualize the pheromone table corresponding to the gesture. Fig. 3.9 shows the visualization of two pheromone tables corresponding to two gesture types: “Square” and “Circle”. According to the color-bars, red cells in these two tables exhibit higher proportion of pheromone values than blue cells. It can be seen that leading diagonal values are distributed more intensive than rest-of-table values. In addition, both of the two types of values are distributed differently from one table to the other. As a result, different gestures are able to be distinguished and correctly classified based on the two features.

### 3.5.2 Pheromone Mechanism and Pheromone Table Update

Basically, during each transition from state $i$ to $j$, the ant lays a pheromone value $\Delta \tau$ on the path linking the previous state and the current state. Since only one
Figure 3.9: Visualization of two pheromone tables: “Square” and “Circle”.

46
artificial ant is used in the system, pheromone deposits on the edges happens according to:
\[ \tau_{ij} = \tau_{ij} + \Delta \tau, \forall i, j \in N \]  
(3.2)

This is simpler compared with equation 2.15. Pheromone evaporates according to equation 2.16, which is the same as in ACO. \( \Delta \tau \) is set to a default value based on extensive experiments (see Section 4.2.3).

Once the user starts to perform a gesture, a corresponding pheromone table is built. The initial values in the pheromone table are set to zero. All values in the pheromone table are updated every time when a new acceleration vector is received and quantized. To illustrate, Fig. 3.10 shows an example of a pheromone table with a codebook sized \( k = 4 \), which is produced corresponding to the acceleration vectors. All the vectors follow a chronological order.

<table>
<thead>
<tr>
<th></th>
<th>↑</th>
<th>→</th>
<th>↓</th>
<th>←</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>→</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↓</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>←</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 3.10: The pheromone table is generated according to the incoming acceleration vector stream.

The leading diagonal values show the number of immediate repetitions of each acceleration vector and the remaining values show the number of each transition point. The sequence of vectors starts from the orange point and we assume that the initial state is ↑. A pheromone value \( \Delta \tau = 1 \) is added whenever the system receives a new vector. Each arrow listed in the first column represents the previously received vector and each arrow in the first row represents the currently received vector. The blue circle shows a transition point between vector ↑ and →, so it adds a value of 1 in the corresponding cell in the table. The red ellipse shows
A series of three vectors ↓. The first vector produces a transition point between → and ↓, so it does not update the pheromone value in the cell pointed to by the red arrow. After this, the system receives two more vectors ↓. As a result, a total value of 2 is added to the corresponding cell in the table.

Once the recording of the gesture is finished, both leading diagonal values and the remaining values in the table are normalized separately, because each of them represents a different feature. The pheromone table Table 3.3 is the normalized table of that in Fig. 3.10.

<table>
<thead>
<tr>
<th></th>
<th>↑</th>
<th>→</th>
<th>↓</th>
<th>←</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>0.2857</td>
<td>0.2000</td>
<td>0.0000</td>
<td>0.2000</td>
</tr>
<tr>
<td>→</td>
<td>0.0000</td>
<td>0.2857</td>
<td>0.2000</td>
<td>0.2000</td>
</tr>
<tr>
<td>↓</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.2857</td>
<td>0.2000</td>
</tr>
<tr>
<td>←</td>
<td>0.2000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1429</td>
</tr>
</tbody>
</table>

Table 3.3: The pheromone table with its values normalized.

We see this normalization as a way of affecting generalization, as normalized values are transform-independent, meaning that gesture type will be represented by corresponding pheromone table, independent of its size and performing speed.

A gesture library is used to stores all the trained gestures. Each newly trained table is stored in the gesture library and available in the operational pipeline for recognition purpose. In this thesis, values in a trained table will not be further modified once they are normalized. However, they can potentially be continuously updated by a recognized gesture which is the same gesture type as the one that produced the pheromone table. This could make ALA an online gesture learning and recognition algorithm, and will be studied as part of our future work.

To summarize, the design of the ant learning model is given in Algorithm 3.

**Algorithm 3 Ant Learning Model**

create an ant

initialize a pheromone table

while gesture is not done do
  ant moves from the previous state to current state
  update pheromone
end while

normalize the pheromone table

The speed should be greater than a certain value, otherwise the input acceleration vectors will be filtered out by the "idle" filter.
3.6 Classification Methods

A new pheromone table is generated as soon as the user starts to perform a gesture. Once the gesture is done, a classifier based on Euclidean distance is then used to match this new table to each of the trained gesture tables. Distance \( c^m \) is calculated as:

\[
c^m = \sum_{i=1, j=1}^{k=14} (\tau_{i,j}^1 - \tau_{i,j}^2)^2
\]

(3.3)

where \( \tau_{i,j}^1 \) and \( \tau_{i,j}^2 \) represent the pheromone values in the corresponding two tables, the gesture that corresponds to the closest matching gesture table \( m \), is the recognition result. In this project, we set \( m \in \{1\text{-Square}, 2\text{-Circle}, 3\text{-Triangle}, 4\text{-Eight}, 5\text{-Roll}\} \) according to the gestures used in our experiment. We propose three classification methods in this thesis:

- **Leading diagonal distance classifier**: It takes the leading diagonal values from the pheromone table into account. All other values are set to 0s. It examines the similarities between two gestures based on Feature I according to equation 3.3 with \( i = j \).

- **Rest-of-table distance classifier**: It evaluates all the values except for the leading diagonal values in the pheromone table. It evaluates the similarities with regards to Feature II according to equation 3.3 with \( i \neq j \).

- **Hybrid classifier**: It is a combination of the two methods whereby, both \( i = j \) and \( i \neq j \) are used.

The reason for considering the first two classification methods is that it is interesting to observe whether or not gestures can be classified correctly based on minimal features (Feature I or Feature II). Each recognized gesture is mapped to a trained gesture type which has the shortest distance between them. Since we do not set any threshold for this distance, even gestures which have not been trained will still be recognized to the most similar trained gesture type. One advantage of doing so is that it provides tolerance for noise in gesture data. For example, it is not necessary to do a standard square in order to be correctly recognized, as long as the gesture has the shortest distance to the trained “Square” sample. As a result, ALA might have good performance in user-independent cases. This is out the scope of this thesis but will be part of our future work.

Some examples used in the remaining section are based on the pheromone table which was shown in Table 3.3.
Leading Diagonal Distance Classifier

Only Feature I is observed by the leading diagonal distance classifier. The normalized pheromone table shown in Table 3.3 can be further transformed into the table shown in Table 3.4. In this case, once the user has performed a new gesture for recognition, a corresponding pheromone table which only contains the leading diagonal values is built. According to equation 3.3 with $i = j$, the Euclidean distances from this newly built table to all the tables stored in the gesture library are computed. A label $m$, which represents a gesture type, is then assigned to the newly built table if trained gesture sample labeled $m$ has the shortest distance from this table.

<table>
<thead>
<tr>
<th></th>
<th>↑</th>
<th>→</th>
<th>↓</th>
<th>←</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>0.2857</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>→</td>
<td>0.0000</td>
<td>0.2857</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>↓</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.2857</td>
<td>0.0000</td>
</tr>
<tr>
<td>←</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1429</td>
</tr>
</tbody>
</table>

Table 3.4: The pheromone table that adapted to the leading diagonal distance classifier. All the values except leading diagonal values are set to 0.

Since the leading diagonal distance classifier only examines the similarities between two gestures based on Feature I, it can not distinguish between two gestures which have the same number of immediate repetitions of each characteristic state but different number of transition points.

An example is given in Fig. 3.11. Both of the two acceleration vector streams start from the orange point and we assume that their initial states are ↑. A pheromone value $\Delta \tau = 1$ is added whenever the system receives a new vector. Both of them have 3 immediate repetitions of the acceleration vector ↑, and 2 of immediate repetitions of the remaining three vectors →, ↓, and ←. Since the numbers of transition points in these two acceleration vector streams are ignored by the classifier, the two identical pheromone tables are then produced. As a result, the leading diagonal distance classifier is not able to distinguish such acceleration vector streams produced by the corresponding gestures.

Rest-of-table Distance Classifier

Differing from the leading diagonal distance classifier, only Feature II is observed by the rest-of-table classifier. The pheromone table shown in Table 3.3 can be further transformed into the table shown in Table 3.5. In this case, once the user
Figure 3.11: Two different acceleration vector streams with the same number of immediate repetitions of each characteristic state (↑, →, ↓ and ←) are transformed into two identical pheromone tables.

has performed a new gesture, a corresponding pheromone table which contains all the remaining values, except the leading diagonal values, is built. According to equation 3.3, with $i \neq j$, the Euclidean distances from the new table to all the trained tables stored in the gesture library are computed, and label $m$ is then assigned to the table.

![Diagram of acceleration vector streams](image)

Table 3.5: The pheromone table that adapted to the rest-of-table distance classifier. Leading diagonal values are set to 0.

<table>
<thead>
<tr>
<th></th>
<th>↑</th>
<th>→</th>
<th>↓</th>
<th>←</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>0.3333</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>→</td>
<td>0.0000</td>
<td>0.2222</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>↓</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.2222</td>
<td>0.0000</td>
</tr>
<tr>
<td>←</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.2222</td>
</tr>
</tbody>
</table>

The rest-of-table classifier can not distinguish between two gestures which have the same number of transition points between different characteristic vectors but different number of immediate repetitions of each characteristic vector. An
example is given in Fig. 3.12. Both of the two acceleration vector streams have 1 transition from ↑ to →, → to ↓ and ↓ to ←, respectively. Since the two identical pheromone tables are produced, the rest-of-table distance classifier is not able to distinguish such acceleration vector streams.

**Hybrid Classifier**

Since both of the two above-mentioned classifiers have their own shortcomings, a combined method, named the hybrid classifier, is then proposed. It classifies gestures based on both of the two features (Feature I and Feature II). Thus, all the values in the pheromone table shown in Table 3.3 are used.

According to equation 3.3 with both \( i = j \) and \( i \neq j \), the Euclidean distances from a newly built pheromone table to all the trained tables stored in the gesture library are computed. The example in Fig. 3.13 indicates that the hybrid classifier is able to distinguish gestures which can not be correctly classified by the two above-mentioned classifiers. The experimental results shown in Section 4.2.3 signify that the hybrid classifier leads to the highest average recognition rates.
Figure 3.13: Pheromone tables adapted to different incoming acceleration vector streams, based on the hybrid classifier.
Chapter 4

Experiments and Results

Chapter Overview

This chapter gives a systematic evaluation of the ALA-based gesture recognition system. Setup of the experiments are described at first, followed by the eight separate tests presented in detail. In some of the experimental results, comparison among the three algorithms, ALA, HMM and DTW are also provided.

The work presented in Sections 4.2.3, 4.2.4, 4.2.7 and 4.2.8 in this chapter has been published and is to appear [29] in the proceedings of the IEEE Congress on Evolutionary Computation (CEC 2013).

Figure 4.1: Working with the ALA-based system in practice.
4.1 Experiments Setup

Fig. 4.1 shows how our system is operated in practice. The window with red background color will indicate the system when to start or end the recording of a gesture sample through a simple click. In order to ensure platform independence, we implemented ALA and the algorithms (HMM and DTW) to which we compare ALA, in Java. These implementations were done within the NetBeans IDE 7.2.1 environment. A Processing library OSC P5\textsuperscript{1} is used to process the OSC Packets and retrieve acceleration data sent from the iPod touch.

Five gesture types are considered for the experiments, see Fig. 4.2. All instances of each gesture should be performed using the same orientation of the sensing device.

![Figure 4.2: The five simple gesture types considered.](image)

Basically, two gesture training methods can be used in ALA:

- **Type I training**: Immediately repeat a gesture $t$ times during one training.

- **Type II training**: Record $t$ separate training samples of a gesture.

In Type I training, only one training sample is needed for each gesture type. Each training sample contains $t$ training instances (repetitions). In Type II training, $t$ training samples are recorded for each gesture type, and each sample contains only one instance (repetition) of the corresponding gesture. Type II training is also used in HMM-based and DTW-based systems. Since we are interested in observing ALA’s performance with minimal training instances, we stress on recognizing gestures with one-instance training ($t = 1$) in the experiments. It should be noted that in the case of one-instance training, Type I training is the same as Type II training.

Both HMM and DTW were tested, and their recognition performance were

\textsuperscript{1}oscP5 is an OSC implementation for the programming environment Processing. It can be used for Java based projects.
compared with ALA's. In this project, HMM was implemented based on the Wiigee library, and DTW was implemented based on [38].

Table 4.1 gives an outline of all the experiments. It should be noted that Type II training were used in order to keep consistent with the training methods used for both HMM and DTW, when a number of \( t > 1 \) training instances were required.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Training Method</th>
<th>No. of Training Instances ( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Trajectory Analysis</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2) Filtering</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3) Parameter Tuning</td>
<td>Type I</td>
<td>1 to 15</td>
</tr>
<tr>
<td>4) Cross Validation</td>
<td>Type I/II</td>
<td>1</td>
</tr>
<tr>
<td>5) Effect of Number of Training Instances</td>
<td>Type II</td>
<td>1 to 15</td>
</tr>
<tr>
<td>6) Effect of Starting Points</td>
<td>Type II</td>
<td>1 to 15</td>
</tr>
<tr>
<td>7) Execution Time (training + recognition)</td>
<td>Type II</td>
<td>1 to 15</td>
</tr>
<tr>
<td>8) User-dependent Tests with One-Instance Training</td>
<td>Type I/II</td>
<td>1</td>
</tr>
</tbody>
</table>

\(^1\) For each gesture type, \( t \) training instances are required.
\(^2\) Comparison with HMM and DTW are provided in experiments 4), 6) and 7).

Table 4.1: Experiments outline.

### 4.2 Results

#### 4.2.1 Trajectory Analysis

Trajectories of gestures are produced by visualizing the corresponding acceleration vector streams. By analyzing the trajectories, two facts can be observed: 1) Visualized acceleration vector streams can have quite different shapes compared with the corresponding gestures. 2) For the same gesture type, different gesture samples can be made up of quite different acceleration vectors.

Fig. 4.3 shows the trajectories of two gestures: Square and Circle. Although we can still recognize the second trajectory as a circle, it is impossible for us to tell which gesture type the first trajectory belongs to. Basically, acceleration is the rate at which the velocity changes with time. For a human user, the velocity of his or her hand changes non-homogeneously during the performance of a gesture. Thus, both acceleration and deceleration happen alternately, especially for non-smooth gestures such as “Square”.

\(^2\) A Java-based gesture recognition library for the Wii remote, see [http://www.wiigee.org/](http://www.wiigee.org/) (last access: 28.04.2013). Based on this, we implemented a HMM-based gesture recognition system, with the iPod Touch used as the input device.
The two trajectories in Fig. 4.3 exhibit quite different shapes, even though they belong to the same gesture “Circle” and were performed by the same person. The reason is: it is impossible for a person to move his or her hand in a perfect shape, for example “Square” or “Circle”. In this case, ALA did not correctly recognize the second gesture (below) as “Circle”, with the first gesture sample (above) being used as training sample. Basically, the unconscious vibration of the hand brings in noise to the gesture data, which constitute part of the corresponding acceleration vector stream. This is mainly the reason why misclassification occurs.
Figure 4.4: Two gesture samples of “Circle”. The second gesture cannot be correctly recognized by ALA.
4.2.2 Filtering

The threshold value $\Delta$ is empirically set to 0.6g. For each gesture type, 5 gesture trails were performed. An overall reduction rate is averaged across the five gesture types. Table 4.2 indicates that the idle filter reduces the number of input acceleration vectors by 23.6% on average.

<table>
<thead>
<tr>
<th></th>
<th>Trail 1</th>
<th>Trail 2</th>
<th>Trail 3</th>
<th>Trail 4</th>
<th>Trail 5</th>
<th>Average</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square</td>
<td>unfiltered</td>
<td>105</td>
<td>105</td>
<td>96</td>
<td>99</td>
<td>99</td>
<td>100.8</td>
</tr>
<tr>
<td></td>
<td>filtered</td>
<td>69</td>
<td>76</td>
<td>64</td>
<td>66</td>
<td>72</td>
<td>69.4</td>
</tr>
<tr>
<td>Circle</td>
<td>unfiltered</td>
<td>68</td>
<td>70</td>
<td>72</td>
<td>68</td>
<td>90</td>
<td>73.6</td>
</tr>
<tr>
<td></td>
<td>filtered</td>
<td>58</td>
<td>60</td>
<td>62</td>
<td>58</td>
<td>64</td>
<td>63.4</td>
</tr>
<tr>
<td>Triangle</td>
<td>unfiltered</td>
<td>84</td>
<td>86</td>
<td>87</td>
<td>90</td>
<td>87</td>
<td>86.8</td>
</tr>
<tr>
<td></td>
<td>filtered</td>
<td>58</td>
<td>67</td>
<td>67</td>
<td>69</td>
<td>56</td>
<td>63.4</td>
</tr>
<tr>
<td>Eight</td>
<td>unfiltered</td>
<td>100</td>
<td>96</td>
<td>98</td>
<td>102</td>
<td>101</td>
<td>99.4</td>
</tr>
<tr>
<td></td>
<td>filtered</td>
<td>88</td>
<td>83</td>
<td>86</td>
<td>87</td>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td>Roll</td>
<td>unfiltered</td>
<td>66</td>
<td>69</td>
<td>65</td>
<td>74</td>
<td>66</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>filtered</td>
<td>42</td>
<td>48</td>
<td>47</td>
<td>53</td>
<td>50</td>
<td>48</td>
</tr>
</tbody>
</table>

Table 4.2: Reduction of Incoming Acceleration Vectors with Threshold Value $\Delta = 0.6g$.

4.2.3 Parameter Tuning

As in ACO, different values of parameters have impacts on its performance, so it is important to find satisfying parameters. Similarly, ALA parameters need to be tuned first in order to guarantee a good overall performance. Main parameters of ALA are the following:

- Codebook size $k = 14$ (fixed in our case)
- Pheromone value $\Delta \tau$
- Evaporation rate $\rho$
- Classifier

The parameter tuning experiment was divided into three parts based on the different classification methods: leading diagonal distance classifier, rest-of-table distance classifier, and hybrid classifier. In each part of the test, the average recognition rates across pheromone value $\Delta \tau \in \{0.01, 0.05, 0.1, 0.5, 1, 2, 5, 10\}$ and
<table>
<thead>
<tr>
<th>ρ</th>
<th>Δτ</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
<th>0.5</th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Leading diagonal Distance Classifier</strong></td>
<td>0.1</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
<td>82.0%</td>
</tr>
<tr>
<td><strong>Rest-of-table Distance Classifier</strong></td>
<td>0.1</td>
<td>86.8%</td>
<td>91.4%</td>
<td>91.4%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>86.8%</td>
<td>91.4%</td>
<td>91.4%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
<td>91.6%</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>91.4%</td>
<td>91.4%</td>
<td>91.4%</td>
<td>91.4%</td>
<td>91.4%</td>
<td>91.4%</td>
<td>91.4%</td>
<td>91.6%</td>
</tr>
<tr>
<td><strong>Hybrid Classifier</strong></td>
<td>0.1</td>
<td>93.6%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
</tr>
</tbody>
</table>

Table 4.3: Results for parameter tuning: pheromone value Δτ, evaporation rate ρ and classification methods. The results are averaged across the five gestures: square, circle, triangle, eight and roll.
evaporation rate $\rho \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ were examined. In order to keep the gesture samples consistent for all the tests, we first recorded the raw acceleration vectors during performing the gestures, and loaded them into the system operational pipeline afterwards. For the training dataset, each gesture was performed with different number of training instances $t \in \{1, 2, 5, 10, 15\}$. For the recognition dataset, each gesture was performed with $t = 1$. In the experiment, each gesture type was tested using 20 separately recorded gesture samples in the recognition dataset and a recognition rate was averaged over them. Both training and recognition samples were performed by the same person.

Table [4.3] shows results for ALA parameter tuning. It is shown that ALA achieved much higher average recognition rates with hybrid classifier rather than the other two classifiers. This was expected since the hybrid classifier uses both information of repetitions of each character state (leading diagonal values) and numbers of transition points (rest-of-table values), which makes it generalize to different gesture types. As a result, the hybrid classifier is selected for ALA.

It is also noticed from the table that ALA achieved almost the same recognition results independent of both pheromone value $\Delta \tau$ and evaporation rate $\rho$. However, ALA tended to achieve higher recognition rates with larger $\Delta \tau$ and in-between values of evaporation rates. We assume that lower evaporation rates allow ALA to “forget” previously incoming acceleration vectors of a gesture, so they contribute less to the pheromone table as compared to the acceleration vectors which are received later. The side-effect of evaporation is that it makes the recognizer rely more on the latter part of gestures. This might strengthen the online learning ability of ALA since the newly received acceleration vectors could refine the pheromone table accordingly. Evaluation of ALA’s online learning ability will be carried out in the future work.

According to the results, the final parameters of ALA are set as following:

- Codebook size $k = 14$ (it is fixed in our project)
- Pheromone value $\Delta \tau = 10$
- Evaporation rate $\rho = 0.5$
- Classifier=Hybrid classifier

This set of parameters was used in the remaining experiments in this chapter. Although the parameters are decided based on extensive tests, they are not optimal, because we did not exhaustively enumerate all the combination of $\Delta \tau$ and $\rho$ in the experiment. Nevertheless, it shall be feasible to adjust both $\Delta \tau$ and $\rho$ according to the practical need.
4.2.4 Cross Validation

In cross validation, for each gesture type, 20 gesture samples recorded. Of the 20 samples, a single recorded sample was picked as training data and the remaining 19 samples were used as the test data. This was then repeated 20 times, with each sample used exactly once as the training data. An average recognition rate (across the 19 test samples) was computed after each test run, and the final result was averaged over the 20 runs. It should be noted that all the gesture samples were performed with only one immediate repetition \((t = 1)\), so the training here is referred to as one-instance training.

<table>
<thead>
<tr>
<th></th>
<th>ALA</th>
<th>HMM</th>
<th>DTW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square</td>
<td>97.11%±10.73%</td>
<td>30.79%±26.14%</td>
<td>100.0%±0.0%</td>
</tr>
<tr>
<td>Circle</td>
<td>98.95%±3.66%</td>
<td>92.89%±9.7%</td>
<td>100.0%±0.0%</td>
</tr>
<tr>
<td>Triangle</td>
<td>96.84%±6.48%</td>
<td>81.32%±26.09%</td>
<td>100.0%±0.0%</td>
</tr>
<tr>
<td>Eight</td>
<td>100.0%±0.0%</td>
<td>68.95%±27.69%</td>
<td>100.0%±0.0%</td>
</tr>
<tr>
<td>Roll</td>
<td>100.0%±0.0%</td>
<td>60.0%±32.6%</td>
<td>100.0%±0.0%</td>
</tr>
</tbody>
</table>

Table 4.4: Results for cross validation on the five gesture types: Square, Circle, Triangle, Eight and Roll. (ALA, HMM and DTW)

Results in table 4.4 indicate that, among the three algorithms, DTW achieved the best recognition performance with all five gestures always being correctly recognized. While ALA achieved a lower recognition accuracy compared with DTW, it still showed a recognition rate of 98.58% averaged across the five test gesture types. Both ALA and DTW performed much better than HMM regarding one-instance training. As a result, we show that both ALA and DTW have very good generalization performance with only one training instance, thus able to produce precise predictions of new gesture samples.

From the cross validation results of ALA, it is noted that the quality of training samples might affect the recognition results significantly. We find that in some specific tests, the recognition rates are much below the average. For example, from the cross validation result of the gesture type “Square”, it is observed that for one of the training instances, ALA achieved an average recognition rate of 52.63%, which is much lower than the average 97.11%. This indicates that this particular instance could be very different from other instances. In fact, this training sample contained more noise than the average, thus was inconsistent with the others. Using such instances as training data could lead to very high misclassification rates. Therefore, gestures need to have some degree of consistency and such inconsistent gesture samples should not be used as training instances.
4.2.5 Effect of Number of Training Instances

We have evaluated ALA’s generalization performance with only one training instance by using cross validation. In this experiment, the effects of number of training instances were examined. It is interesting to observe how ALA’s recognition rate varies when different number of training instances are used.

For each gesture type, 15 training samples \((t = 1)\) were recorded independently. For recognition, the same gesture samples (20 samples for each gesture type) as in Section 4.2.4 were used.

![Figure 4.5: Average recognition results vs. Number of training instances (ALA)](image)

Fig. 4.5 shows the average recognition rates across the five gesture types, depending on the number of training instances. The results indicate that ALA achieved the lowest recognition rate of 88.0% with only one training instance, and it achieved a much higher recognition rate of 97.0% when one more training instance was added. The average recognition rate mainly stays above 96.0%, with the number of training instances \(t\) ranging from 2 to 15. Therefore, in general it is not necessary for ALA to use more than two training instances.
4.2.6 Effect of Starting Point

The flexibility of gesture’s starting point can be essential in many scenarios such as video game playing and active music performance. In these cases, users do not want to be much restricted when performing gestures. They would like to perform a gesture that can correctly trigger the corresponding event (e.g., a “jump” in a video game) even when it starts from an arbitrary point. To the best of our knowledge, both HMM and DTW are sensitive to the starting point of gestures, if they are not trained with gesture samples performed with different starting points.

Whether the starting point of gestures can be flexible for ALA is of interest to us. Theoretically, the starting point is flexible for all gestures in Fig. 4.2 except (e) only if an evaporation rate $\rho = 1$ is used. The effect of a $\rho < 1$ is that the acceleration vectors constituting the ending part of a gesture contribute more to the pheromone accumulated in the edges, as compared to those constituting the starting part. As a result, the values in the pheromone table for the same gesture type can vary due to different starting points. However, if a gesture is trained with many instances, the evaporation mechanism may not have significant effects on the recognition accuracy, because every new input instance during the training will re-update the pheromone table. In other words, the starting point might be flexible for all gestures except (e) in such cases.

Two tests were conducted to study the effects of starting point:
• Evaluate how evaporation rate will affect the recognition accuracy on gestures with different starting points.

• Evaluate how the number of training instances will affect the flexibility of performing gestures with different starting points.

The four gesture types: “Square”, “Circle”, “Triangle” and “Eight”, were considered for the two tests. To make it simple, we define that all the four gesture types in Fig. 4.2 were performed with a starting point SP0 (the original starting point), while those in Fig. 4.6 and 4.7 were performed with different starting points SP1 and SP2. Compared with starting point SP2, SP1 was designed to be closer (in the time series of a gesture) to SP0.

For training, the same gesture samples from Section 4.2.5 were used. For recognition, 20 gesture samples were performed and recorded for each gesture type with different starting points SP1 and SP2, respectively.

Evaluation on Evaporation Rate

In the test, all the four gesture types were trained with only one training instance. Gestures performed with both starting point SP1 and SP2 were tested on the same training samples. An evaporation rate $\rho \in \{0.1, 0.2, 0.3, ..., 1.0\}$ was used for each test.

Table 4.5 shows the evaluation results. Surprisingly, the evaporation rate $\rho$ did not show any effect on the recognition rate for recognizing gestures with different starting points. The possible reason for this might be that, although an evaporation rate $\rho < 1$ changes the values in the pheromone table of a gesture, feature information of each gesture type still maintained. As a result, a lower evaporation rate will not affect the recognition results.

Evaluation on Number of Training Instances

The same gesture samples, both for training and recognition, were used for ALA as well as HMM and DTW. Since we have observed that the evaporation rate does not affect the recognition accuracy, $\rho = 0.5$ was then selected according to the parameter tuning result, see Section 4.2.3

Fig. 4.8 shows ALA’s average recognition rates across the four gesture types performed with starting points SP0, SP1 and SP2. The results indicate: 1) For all the tests, with number of training instance $t$ ranging from 1 to 15, the recognition accuracy for gestures with different starting points (SP1 and SP2) is lower than that for gestures with the original starting point (SP0). 2) For training gesture samples with only one training instance $t = 1$, ALA achieved the lowest average
Table 4.5: Evaluation results on how evaporation rate will affect the recognition accuracy, tested by gestures performed with different starting points.
recognition rate in all three cases (SP0, SP1 and SP2). For training gesture samples with the number of instances \( t \) from 2 to 15, the average recognition rate stayed roughly the same. 3) In general, recognizing gestures with starting point SP1 achieved roughly the same average recognition rates as those with starting point SP2.

Fig. 4.9 shows the average recognition rates across different number of training instances \( t \in \{1, 2, 3, ..., 15\} \) for all the four gesture types, “Square”, “Circle”, “Triangle” and “Eight”. It is interesting to see: 1) For some gesture types such as “Square” and “Eight”, the average recognition rates were not affected by the
Figure 4.10: Average recognition results vs. Number of training instances, tested by gestures with starting points SP0 (above), SP1 (middle) and SP2 (bottom). (ALA, HMM and DTW)
starting point. 2) For some gesture types such as “Circle” and “Eight” (especially “Circle”), the average recognition rates highly depended on the starting point.

A comparison among the three algorithms with test gestures starting from SP0, SP1 and SP2 is shown in Fig. 4.10. Two facts can be drawn from the results: 1) For all the three algorithms, the average recognition rate decreased when gestures with different starting points (SP1 and SP2) were tested. 2) Both ALA and DTW achieved similar recognition accuracy and outperformed HMM.

4.2.7 Execution Time (training+recognition)

The execution time of the three algorithms was measured on a Thinkpad E420 114139c laptop with an Intel(R) Core(TM) i5-2410M Dual-core processor (2.30GHz, 2.30GHz) with 4GB RAM. The test was performed on Windows 7 (64-bits). The execution time provided in the result consists of both training and recognition.

Each algorithm was tested on gestures with different number of training instances \( t \in \{1, 2, 3, ..., 15\} \). For each number of training instance \( t \), the execution time was averaged across all the five gestures.

![Figure 4.11: Average execution time vs. Number of training instances. (ALA, HMM and DTW)](image)

Fig. 4.11 indicates that, compared with both HMM and DTW, ALA’s average execution time was much shorter, and remained more or less the same, independent of the number of training instances. While DTW had similar execution time with HMM when the number of training instances was below 7, it performed much slower than HMM when more than 8 training instances were used. Therefore, the result shows that ALA requires extremely low computational overhead, and outperforms both HMM and DTW with regards to execution time.
Basically, the ALA-based system can be divided into two parts: gesture training and gesture recognition. The main task in gesture training is to transform the input gestures into corresponding pheromone tables. To be specific, each input acceleration vector is quantized to a characteristic state, and then a pheromone value $\Delta \tau$ is added to the corresponding cell, which represents the transition from the previous characteristic state to the current one, in the pheromone table. The training process stops immediately when a gesture is performed. As for the main task in gesture recognition, it is to classify new input gestures and then give the recognized gesture types. The classifier computes the Euclidean distances between this newly built pheromone table and all the trained tables that stored in the gesture library. Since the size of the pheromone tables is fixed (14 by 14) in this project, the classification process requires extremely low computations.

4.2.8 User-dependent Tests with One-Instance Training

In order to evaluate the performance of our system in practice, we conducted six separate tests, in the aim of reducing variability. All the tests were carried out by the same person in different periods during two days: three tests were performed separately in the morning, afternoon and evening in the first day, and the other three tests were performed in the same periods during the second day, respectively.

The same five gestures as shown in Fig. 4.2 were used for the evaluation. 100 gesture trails were performed in each test with 20 trails for each gesture. As a result, in total 600 gesture trails were conducted in the experiment. An average recognition rate of 91.3% across the five gestures was observed. The average recognition rate for each gesture is shown in Fig. 4.12 and the average recognition rate for each test is shown in Fig. 4.13.

Results in Fig. 4.13 show that the recognition rates varied corresponding to different training samples (gestures were re-trained in each test), even though they were trained by the same person. It indicates that the quality of training data can affect the recognition accuracy significantly.
Figure 4.12: Average recognition rates of the five gestures. (ALA)

Figure 4.13: Average recognition rates of the six tests. (ALA)
Chapter 5

Potential Applications

In this chapter, ideas of three potential applications which could be achieved based on the ant learning system are presented.

Active Music Performance

The recognition of gestures in a musical domain requires a low-latency, highly robust and user-configurable recognition system. ALA can be used for such systems due to its efficiency. Since ALA can achieve a high recognition accuracy with minimal training instances, music performers can easily retrain a previous gesture or add new gestures to the system with minimal training efforts. Besides, ALA requires very low computational overhead, making it suitable for real-time gesture recognition tasks.

Basically, the performers need to train some gestures according to his or her preference and demand. Each trained gesture needs to be pre-defined and mapped with a specific function, such as adding one filter to the system or simply playing a note “C”, see from Fig. 5.1.

![Figure 5.1: A gesture “Square” is used to trigger a note.](image)

As the active music performance requires a real-time and efficient gesture
Motion Detection for a Smart Music Player

The scenario considered in this case deals with the issue of a smart music player that automatically selects music with suitable rhythms according to the user’s status when e.g. he or she is running in a gym. Such status could be “Running”, “Jogging” and “Walking”.

It can be clearly seen from Fig. 5.2 that the acceleration waveforms of the two statues “Walking” and “Running” differ widely in magnitude and shape, which indicates that they can be easily distinguished from each other.

Basically, these acceleration waveforms can be trained and transformed into corresponding pheromone tables based on a different codebook. Once the training is done, these pheromone tables are stored and thus available for recognizing the user’s status. As a result, the smart music player can then select music based on the user’s current status, for example, music with fast rhythm for running or music with slow rhythm for walking.

Interactive Music Composition

We have introduced an automatic music composition system in Section 2.4. This system takes advantage of the ant’s collective behavior, and automatically produces music (a sequence of nodes) by its artificial ants. Essentially, a pheromone table, which is a note transition table that represents the strength values of the link between two notes, can be used to lead the ants’ moves. See table 2.2.

By using the ant learning model, a same table (pheromone table) can be built by using a codebook with size $k = 7$. One thing that should be addressed regarding the table is how to select the seven characteristic vectors. To conclude, ALA can be used to modify the values in such a transition table on-the-fly by sensing the user’s motion (for example, hand motion), thus expressing the user’s motion with the produced music.

---

1 It might require a codebook or not.
Figure 5.2: Two acceleration waveforms corresponding to status “Walking” (above) and “Running” (below), respectively.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we propose the Ant Learning Algorithm (ALA) for accelerometer-based gesture recognition systems. It is motivated by the study on the two current popular algorithms: Hidden Markov Model (HMM) and Dynamic Time Warping (DTW). We aim at addressing the limitations of these algorithms, which have not been satisfactorily addressed in previous work.

We show that ALA requires minimal training effort. The cross validation results (Section 4.2.4) indicate that ALA exhibited good generalization performance with only one training instance. Besides, results of the user-dependent experiment (Section 4.2.8) show that ALA achieved an average recognition rate of 91.3% with one-instance training in practice, which is promising but leaves room for further improvement.

By evaluating ALA’s execution time (Section 4.2.7), we show that ALA requires extremely low computational overhead. The evaluation results (training time + recognition time) demonstrate that ALA ran much faster than both HMM and DTW. Devices with low computing power could thus benefit from it and achieve real-time gesture recognition.

Recognition performance was also tested by gestures performed with different starting points (Section 4.2.6). For some gesture types, ALA is robust with recognizing gestures performed with different starting points. This is desirable in terms of enhancing the interactive experience through gestures. However, for some other gesture types, the recognition accuracy highly depends on the starting point.
6.2 Future Work

Future work could be observing how ALA performs in online learning cases. We plan to evaluate the recognition results on-the-fly, where trained pheromone tables are re-updated during the process. It is expected that a trained gesture type can be modified to another inconsistent type after several trials during the online learning.

User-independent experiments will be carried out to see how ALA performs in such cases, as it is important for a gesture recognition system to have competitive accuracy in universal cases regardless of the user.

Currently, ALA classifies gestures simply by computing the Euclidean distances between pheromone tables, using all the values in the tables (the hybrid classifier). A better classification method could be developed, depending on a better understanding of how to distinguish among different gesture types based on the two features (see Section 3.5.1).

It is also worthy of exploring the applicability of ALA to other pattern recognition problems. To be specific, it is possible to have one or more pheromone tables representing extended features which could be learned and recorded in such tables for further recognition.
Bibliography


Appendix A

Java Source Codes for Ant Learning Algorithm (ALA)

Receiving OSC Messages

```java
package aladatacollect;

/**
 * @author Sichao Song
 */
import netP5.NetAddress;
import oscP5.OscMessage;
import oscP5.OscP5;

public class OscReceiver {
    OscP5 oscP5;
    NetAddress myRemoteLocation;
    float accx, accy, accz;
    public float Rx, Ry, Rz, R;

    OscReceiver(String remoteIp, int port) {
        oscP5 = new OscP5(this, port);
        myRemoteLocation = new NetAddress(remoteIp, port);
    }

    public void oscEvent(OscMessage msg) {
```
String addr = msg.addrPattern();

if (addr.equals("/Motion/Acceleration/x")) {
    accx = msg.get(0).floatValue();
} else if (addr.equals("/Motion/Acceleration/y") ) {
    accy = msg.get(0).floatValue();
} else if (addr.equals("/Motion/Acceleration/z") ) {
    accz = msg.get(0).floatValue();
}

Rx = accx;
Ry = accy;
Rz = accz;

Idle Filter

package aladatacollect;

/**
 * @author Sichao Song
 */
public abstract class Filter {

    public double[] filter(double[] vector){
        if(vector==null){
            return null;
        } else{
            return filterAlgorithm(vector);
        }
    }

    abstract public double[] filterAlgorithm(double[] vector);

    abstract public void reset();
package aladatacollect;

/**
 * @author Sichao Song
 */
public class IdleFilter extends Filter {

double sensivity;

public IdleFilter() {
    super();
    this.sensivity = 0.0;
}

@Override
public void reset() {
}

@Override
public double[] filterAlgorithm(double[] R) {

    if ((absValue > (0.6 + this.sensivity))) {
        return R;
    } else {
        return null;
    }
}

public void setSensivity(double sensivity) {
    this.sensivity = sensivity;
}

public double getSensivity() {
    return this.sensivity;
}
}
Vector Quantizer

package alatraining;

import java.util.Vector;

/**
 * @author Sichao Song
 */
public class FeatureMap {

double[][] fvBook = {{1, 0, 0},
   {Math.sqrt(2) / 2, 0, -Math.sqrt(2) / 2},
   {0, 0, -1},
   {-Math.sqrt(2) / 2, 0, -Math.sqrt(2) / 2},
   {-1, 0, 0},
   {-Math.sqrt(2) / 2, 0, Math.sqrt(2) / 2},
   {0, 0, 1},
   {Math.sqrt(2) / 2, 0, Math.sqrt(2) / 2},
   {0, -1, 0},
   {0, -Math.sqrt(2) / 2, -Math.sqrt(2) / 2},
   {0, Math.sqrt(2) / 2, -Math.sqrt(2) / 2},
   {0, 1, 0},
   {0, Math.sqrt(2) / 2, Math.sqrt(2) / 2},
   {0, -Math.sqrt(2) / 2, Math.sqrt(2) / 2}};

double[] R = new double[3];
int state;
Vector<Integer> sList = new Vector<Integer>();

FeatureMap() {
}

double dist(double[] Rv1, double[] Rv2) {
}

86
```java
int featureMap(double[] Rv) {
    double minDist = dist(Rv, fvBook[0]);
    state = 0;

    for (int i = 1; i < 14; i++) {
        if (dist(Rv, fvBook[i]) < minDist) {
            minDist = dist(Rv, fvBook[i]);
            state = i;
        }
    }
    sList.add(state);

    return state;
}

Vector<Integer> stateList() {
    return sList;
}

Ant Learning Model

package alatraining;

/**
 * @author Sichao Song
 */
public class AntLearning {
    double[][] pTable = new double[14][14];
    int state, oldState;

    AntLearning(int s) {
        state = s;
        oldState = s;

        for (int i = 0; i < 14; i++) {
            for (int j = 0; j < 14; j++) {
                pTable[i][j] = 0.0;
            }
        }
    }
```
```java
void clear()
{
    for (int i = 0; i < 14; i++) {
        for (int j = 0; j < 14; j++) {
            pTable[i][j] = 0.0;
        }
    }
}

double[][] pTableUpdate(int s, float pheromone) {
    state = s;
    pTable[oldState][state] += (double) pheromone;
    oldState = state;
    return pTable;
}

package alatraining;

/**
 * @author Sichao Song
 */
public class GestureTraining {

double[][] pTable = new double[14][14];
float pheromone;

GestureTraining(float pheromone) {
    this.pheromone = pheromone;
}

void gestureTraining(AntLearning aLearning, int state) {
    for (int i = 0; i < 14; i++) {
        for (int j = 0; j < 14; j++) {
            pTable[i][j] = aLearning.pTableUpdate(
```
```c
state, pheromone)[i][j];
}

void evaporation(double e) {
    for (int i = 0; i < 14; i++) {
        for (int j = 0; j < 14; j++) {
            pTable[i][j] *= e;
        }
    }
}

double[[[]] getPTable(int flag) {
    if (flag == 1) {
        double sumDiag = 0.0, sumRest = 0.0;
        for (int n = 0; n < 14; n++) {
            sumDiag += pTable[n][n];
        }
        for (int i = 0; i < 14; i++) {
            for (int j = 0; j < 14; j++) {
                pTable[i][j] /= sumDiag;
            }
        }
        for (int i = 0; i < 14; i++) {
            for (int j = 0; j < 14; j++) {
                if (i != j) {
                    sumRest += pTable[i][j];
                }
            }
        }
        for (int i = 0; i < 14; i++) {
            for (int j = 0; j < 14; j++) {
                if (i != j) {
                    pTable[i][j] = pTable[i][j] / sumRest;
                }
            }
        }
    }
}
```
return pTable;
}

void clear() {
    for (int i = 0; i < 14; i++) {
        for (int j = 0; j < 14; j++) {
            pTable[i][j] = 0.0;
        }
    }
}

void printTable() {
    System.out.println("The pheromone table is as below:");
    for (int i = 0; i < 14; i++) {
        for (int j = 0; j < 14; j++) {
            System.out.print(pTable[i][j] + "");
        }
        System.out.println();
    }
}

Classifier

package alarecognition;

import java.util.Vector;

/**
 * @author Sichao Song
 */
public class Classifier {
    Vector<double[][]> pVector;
    double [][] pTable;
    double [] euclideanDist;
}
Classifier(Vector<double>[], pVector, double[][] pTable) {
    this.pVector = pVector;
    this.pTable = pTable;
    euclideanDist = new double[pVector.size()];
}

double euclideanDistance(double[][] table, int mode) {
    double dist = 0.0;

    if (mode == 1) { // mode1: diag
        for (int n = 0; n < 14; n++) {
            dist += (pTable[n][n] - table[n][n]) * (pTable[n][n] - table[n][n]);
        }
    } else if (mode == 2) { // mode2: rest
        for (int i = 0; i < 14; i++) {
            for (int j = 0; j < 14; j++) {
                if (i != j) {
                    dist += (pTable[i][j] - table[i][j]) * (pTable[i][j] - table[i][j]);
                }
            }
        }
    } else if (mode == 3) { // mode3: hybrid
        for (int i = 0; i < 14; i++) {
            for (int j = 0; j < 14; j++) {
                dist += (pTable[i][j] - table[i][j]) * (pTable[i][j] - table[i][j]);
            }
        }
    }

    return dist;
}

double[] distanceVector(Vector<double>[] pVector, int method) {
for (int n = 0; n < pVector.size(); n++) {
    euclideanDist[n] = this.euclideanDistance(pVector.get(n), method); // 1—diag; 2—rest; 3—hybrid
}
return euclideanDist;
}

void printDistVector() {
    for (int n = 0; n < pVector.size(); n++) {
        System.out.print(euclideanDist[n] + " ");
    }
}
}