

# AUS Repository

## ZigBee-Based Tangible Input Devices

|               |   |
|---------------|---|
| Item Type     | Thesis  |
| Authors       | Khan, Waqqas Munir  |
| Download date | 2026-03-16 05:01:55   |
| Link to Item  | <a href="http://hdl.handle.net/11073/7824">http://hdl.handle.net/11073/7824</a> |

# ZIGBEE-BASED TANGIBLE INPUT DEVICES

by

Waqqas Munir Khan

A Thesis Presented to the Faculty of the  
American University of Sharjah  
College of Engineering  
In Partial Fulfillment  
of the Requirements  
for the Degree of

Master of Science in  
Mechatronics Engineering

Sharjah, United Arab Emirates

May 2015

© 2015 Waqqas Munir Khan. All rights reserved.

## Approval Signatures

We, the undersigned, approve the Master's Thesis of Waqqas Munir Khan.

Thesis Title: ZigBee-Based Tangible Input Devices

---

Dr. Imran A. Zualkernan  
Associate Professor, Department of Computer Science and Engineering  
Thesis Advisor

---

Dr. Tarik Ozkul  
Professor, Department of Computer Science and Engineering  
Thesis Committee Member

---

Dr. Rached Dhaouadi  
Professor, Department of Electrical Engineering  
Thesis Committee Member

---

Dr. Mamoun Abdel-Hafez  
Director, Mechatronics Graduate Program

---

Dr. Mohamed El Tarhuni  
Associate Dean, College of Engineering

---

Dr. Leland Blank  
Dean, College of Engineering

---

Dr. Khaled Assaleh  
Director of Graduate Studies

## **Acknowledgements**

I am thankful to my Almighty Allah for His mercy in guiding me with knowledge because of which I was able to realize the importance of education. I am also thankful to my parents and family for their love and care which helped me to recognize my potential and skills.

I would like to thank American University of Sharjah for providing me with the scholarship to work as a graduate assistant. This scholarship played a key role in achieving my educational dreams by reducing the financial burden placed on me to pursue my education.

I am extremely thankful to Dr.Imran Zualkernan for his throughout guidance, patience and support during my research. I would also like to thank all my professors who continued to enlighten me with their priceless knowledge and guidance that helped me to accomplish my research goals.

## **Dedication**

I would like to dedicate this research work to my great parents: Muhammad Munir Khan and Muzammilah Munir Khan, and to my loving wife: Javaria Waqqas.

## Abstract

ZigBee has emerged as a dominant low-power wireless standard for input devices. It has been used widely for building wireless input devices. However, the design of input devices in user environments has remained traditional and primarily consists of small keyboards and displays. The availability of cheap accelerometers and gyros has enabled the construction of gesture-based input devices. This thesis presents the design and implementation of gesture-based tangible input devices that are based on the ZigBee protocol. These ZigBee-based tangible input devices or ZTIDs are small, light-weight and graspable, and can roll, rub and tap to provide a surface gesture-based interface. This surface gesture-based interface is different from the traditional keypad-based devices because instead of a keypad, surface gestures will be used to communicate with the user environment. This type of interaction is not only natural but also creates a sense of intimacy between the user and their environment. Specialized hardware has been designed and implemented for this device. The ZTID hardware integrates the ZigBee platform, an accelerometer, a universal serial bus controller and a microcontroller with switches and an LED interface. The ZTID hardware has been tested for wireless link quality and range. It has been found that the wireless link quality is satisfactory and that ZTID can be used in star or mesh networking. The accelerometer, universal serial bus controller and a microcontroller have also been tested in the hardware. The device hardware is coupled with software implementing gesture-recognition using hidden Markov machines. Eight surface gestures have been defined for ZTID. A single Markov machine, one for each gesture, has been trained and then tested by using the HMM forward algorithm using the random sub-sampling approach. The resulting devices show an accuracy of 99.7% in gesture recognition. The ZTID gesture recognition is evaluated using receiver operating characteristics which show that the surface gestures that originate from ZTID can be used to control the user environment efficiently. Being based on ZigBee protocol, ZTID can be used as a building block for mechatronics design and also for integration with the existing products based on ZigBee.

**Search Terms:** Tangible Interface, ZigBee Network, Accelerometer, Gesture Recognition, Hidden Markov Model.

## Table of Contents

|   |           |
|---|-----------|
| <b>Abstract.....</b>                      | <b>6</b>  |
| <b>List of Figures.....</b>               | <b>9</b>  |
| <b>List of Tables.....</b>                | <b>12</b> |
| <b>Chapter 1. Introduction.....</b>       | <b>14</b> |
| 1.1. Thesis Contribution.....             | 18        |
| 1.2. Thesis Organization .....            | 19        |
| <b>Chapter 2. Literature Review .....</b> | <b>20</b> |
| 2.1. Hardware Design.....                 | 21        |
| 2.1.1. Discussion .....                   | 29        |
| 2.2. Software Design .....                | 31        |
| 2.2.1. Discussion .....                   | 36        |
| 2.3. Selected Solutions .....             | 38        |
| <b>Chapter 3. Hardware Design .....</b>   | <b>39</b> |
| 3.1. Schematic Design.....                | 39        |
| 3.2. PCB Design.....                      | 42        |
| 3.3. PCB Manufacturing and Assembly.....  | 44        |
| 3.4. Final Device Assembly .....          | 49        |
| <b>Chapter 4. Software Design.....</b>    | <b>50</b> |
| 4.1. Data Collection .....                | 50        |
| 4.2. Filtering .....                      | 52        |
| 4.3. Feature Segmentation.....            | 54        |
| 4.4. Classification.....                  | 55        |
| <b>Chapter 5. Evaluation.....</b>         | <b>60</b> |

|   |            |
|---|------------|
| 5.1. Hardware Evaluation.....                             | 60         |
| 5.2. Software Evaluation.....                             | 61         |
| <b>Chapter 6. Applications.....</b>                       | <b>66</b>  |
| <b>Chapter 7. Conclusion .....</b>                        | <b>70</b>  |
| <b>References.....</b>                                    | <b>72</b>  |
| <b>Appendices.....</b>                                    | <b>78</b>  |
| Appendix A: MC1321x Block Diagram.....                    | 78         |
| Appendix B: Network Implementation for MC1321x.....       | 79         |
| Appendix C: Casing Modification.....                      | 81         |
| Appendix D: Surface Gestures.....                         | 84         |
| Appendix E: Experimental Setup for Data Collection.....   | 87         |
| Appendix F: Filtering and Feature Segmentation Plots..... | 90         |
| Appendix G: Hidden Markov Modeling.....                   | 94         |
| Appendix H: Trained HMMs.....                             | 99         |
| Appendix I: Matlab Codes .....                            | 101        |
| Appendix J: C Codes .....                                 | 119        |
| <b>Vita.....</b>  | <b>125</b> |

## List of Figures

|  |    |
|--|----|
| Figure 1: Surface behaviors for ZTIDs .....  | 16 |
| Figure 2: Requirement of a tangible user interface.....  | 17 |
| Figure 3: Conceptual design for ZTID.....  | 17 |
| Figure 4: ZigBee network module options [19], [20], [21], and [22] .....                                     | 23 |
| Figure 5: Software solutions for ZigBee [25] .....   | 23 |
| Figure 6: Network hardware solution for MC1321x [21].....  | 27 |
| Figure 7: BDM programming port connections for MC1321x [21].....   | 28 |
| Figure 8: CP2102 interfaced with MC1321x [23] .....  | 28 |
| Figure 9: MMA7260Q interfaced with MC1321x [23] .....  | 29 |
| Figure 10: LEDs and switches interfaced with MC1321x [23].....   | 29 |
| Figure 11: ZTID device hardware architecture.....  | 39 |
| Figure 12: Schematic of RF-control-board .....   | 41 |
| Figure 13: Schematic of LED-switch-board .....   | 42 |
| Figure 14: RF-control-board PCB design files (a) CMP (b) SOL (c) CRC (d) PLC (e)<br>STC (f) STS (g) DRL..... | 43 |
| Figure 15: LED-switch-board PCB design files (a) CMP (b) SOL (c) CRC (d) PLC (e)<br>STC (f) STS (g) DRL..... | 44 |
| Figure 16: SMD part requirements and board dimensions (a) RF-control-board (b)<br>LED-switch-board.....      | 44 |
| Figure 17: RF-control-board (a) top side (b) bottom side.....  | 47 |
| Figure 18: Fully assembled RF-control-board (a) top side (b) bottom side .....                               | 48 |
| Figure 19: LED-switch-board (a) top side (b) bottom side.....  | 48 |
| Figure 20: Fully assembled LED-switch-board (a) top side (b) bottom side .....                               | 49 |

|  |    |
|--|----|
| Figure 21: Final assembly of ZTID .....  | 49 |
| Figure 22: Steps for gesture recognition .....   | 50 |
| Figure 23: ZTID stickers showing accelerometer axes.....   | 51 |
| Figure 24: Experimental setup for data collection.....   | 51 |
| Figure 25: Selecting the order of moving average filter .....  | 53 |
| Figure 26: Sample of filtered accelerometer data for gesture Roll-X.....   | 53 |
| Figure 27: Sample codebook sequences of states for gesture Roll-X using K=8.....   | 54 |
| Figure 28: Using Baum-Welch algorithm to train one HMM per gesture.....  | 56 |
| Figure 29: Using forward algorithm to classify a sequence of 1D codebook states into<br>the best matching gesture.....   | 58 |
| Figure 30: Programming the ZTID .....  | 60 |
| Figure 31: Accuracy verses the size of training sub-set for various code books.....  | 61 |
| Figure 32: ROC for eight surface gestures using codebook size K=8 (a) Roll-X (b)<br>Rub-4-X (c) Rub-4-Y (d) Tap-1 (e) Tap-2 (f) Tap-3 (g) Tap-4 (h) Tap-5 .....  | 63 |
| Figure 33: ROC for eight surface gestures using codebook size K=10 (a) Roll-X (b)<br>Rub-4-X (c) Rub-4-Y (d) Tap-1 (e) Tap-2 (f) Tap-3 (g) Tap-4 (h) Tap-5 ..... | 64 |
| Figure 34: ROC for eight surface gestures using codebook size K=14 (a) Roll-X (b)<br>Rub-4-X (c) Rub-4-Y (d) Tap-1 (e) Tap-2 (f) Tap-3 (g) Tap-4 (h) Tap-5 ..... | 65 |
| Figure 35: ZTID features (a) Hardware ports (b) I/O interfaces .....   | 66 |
| Figure 36: ZTID stickers showing accelerometer axes.....   | 69 |
| Figure 37: Block diagram of MC1321x [31] .....   | 78 |
| Figure 38: Network implementation using BeeKit GUI [24] .....  | 79 |
| Figure 39: Importing BeeKit solution in Freescale Code Warrior classic [64] .....  | 80 |
| Figure 40: Casing modifications for the LED-switch-board .....   | 81 |
| Figure 41: Casing modifications for RF-control-board .....   | 83 |

|   |    |
|---|----|
| Figure 42: Gesture Roll-X .....   | 84 |
| Figure 43: Gesture Rub-4-X .....  | 84 |
| Figure 44: Gesture Rub-4-Y .....  | 85 |
| Figure 45: Gesture Tap-1 .....  | 85 |
| Figure 46: Gesture Tap-2 .....  | 85 |
| Figure 47: Gesture Tap-3 .....  | 86 |
| Figure 48: Gesture Tap-4 .....  | 86 |
| Figure 49: Gesture Tap-5 .....  | 86 |
| Figure 50: Device setup for data collection using HyperTerminal .....     | 87 |
| Figure 51: Data in ASCII for a gesture .....                              | 89 |
| Figure 52: Un-filtered acceleration vector .....                          | 89 |
| Figure 53: Un-filtered and filtered acceleration for eight gestures ..... | 91 |
| Figure 54: Clusters for gesture Roll-X using K=8, 10 and 14 .....         | 92 |
| Figure 55: Codebooks for gesture Roll-X using K=8, 10 and 14 .....        | 93 |
| Figure 56: Example of a Markov process [69] .....                         | 94 |
| Figure 57: Example of hidden Markov modeling [69] .....                   | 95 |
| Figure 58: Example of two Gaussian mixtures [69] .....                    | 96 |

## List of Tables

|  |    |
|--|----|
| Table 1: Existing user interfaces [13], [14], and [15] .....             | 20 |
| Table 2: Comparison of network solutions [17] .....                      | 22 |
| Table 3: ZigBee market and application areas [17] .....                  | 22 |
| Table 4: Cost comparison of ZigBee modules (Jan. 17, 2015).....          | 24 |
| Table 5: Features of software solutions for ZigBee [25] .....            | 25 |
| Table 6: Supported hardware for network solutions [25] .....             | 26 |
| Table 7: PCB form factor solutions [21], [22], and [26] .....            | 26 |
| Table 8: Gesture recognition steps and alternate methods.....            | 37 |
| Table 9: Selected solutions .....  | 38 |
| Table 10: Requirements for ZTID and the proposed hardware solutions..... | 40 |
| Table 11: PCB design files description.....                              | 42 |
| Table 12: Part list for ZTID .....                                       | 45 |
| Table 13: PCB manufacturing requirements for ZTID.....                   | 46 |
| Table 14: Manufacturing and assembly pricing.....                        | 47 |
| Table 15: ZTID surface gestures description.....                         | 51 |
| Table 16: Memory requirement for HMM storage .....                       | 57 |
| Table 17: Memory requirement for embedded coding .....                   | 58 |
| Table 18: K-Mean algorithm execution time.....                           | 59 |
| Table 19: Forward algorithm execution time.....                          | 59 |
| Table 20: Recognition accuracy and codebook size .....                   | 62 |
| Table 21: Detailed features of ZTID.....                                 | 67 |
| Table 22: HyperTerminal setup for ZTID .....                             | 88 |
| Table 23: State transition matrix for K=8 for gesture Roll-X.....        | 99 |

|   |     |
|---|-----|
| Table 24: Observations probability matrix using $K=8$ for gesture Roll-X .....  | 99  |
| Table 25: State transition matrix for $K=10$ for gesture Roll-X.....            | 99  |
| Table 26: Observations probability matrix using $K=10$ for gesture Roll-X ..... | 99  |
| Table 27: State transition matrix for $K=14$ for gesture Roll-X.....            | 100 |
| Table 28: Observations probability matrix using $K=14$ for gesture Roll-X ..... | 100 |

## Chapter 1. Introduction

ZigBee is a low-cost, low-power, wireless mesh network standard targeted at wide development of long battery life devices in wireless control and monitoring applications [1]. The protocol is used to create low-data-rate wireless personal area networks built from small, low-power digital radios that are typically integrated with microcontrollers. It operates in the industrial, scientific and medical (ISM) radio bands and has a defined rate of 250 kbit/s, best suited for intermittent data transmissions from a sensor or input device.

ZigBee builds on the physical layer and media access control defined in IEEE standard 802.15.4. The specification includes four additional key components: network layer, application layer, ZigBee device objects (ZDOs) and manufacturer-defined application objects, which allow for customization and favor total integration. ZDOs are responsible for a number of tasks including keeping track of device roles, managing requests to join a network, as well as device discovery and security. ZigBee networks are secured by 128 bit symmetric encryption keys. The ZigBee network layer natively supports both star and tree networks, and generic mesh networking. Every network must have one coordinator device tasked with its creation, the control of its parameters and basic maintenance. Within star networks, the coordinator must be the central node. Both trees and meshes allow the use of ZigBee routers to extend communication at the network level [2]. ZigBee is typically used in low-data-rate applications that require long battery life and secure networking. Applications include wireless light switches, electrical meters with in-home-displays, traffic management systems, and other consumer and industrial equipment that requires short-range low-rate wireless data transfer. Other wireless personal area network (WPAN) standards like Bluetooth and IrDA address high data rate applications such as voice, video and local area network (LAN) communications.

The IEEE 802.15.4 and the ZigBee networking standard have been used for building efficient and low cost lighting systems for buildings and home automation [3], [4]. This class of networking standards has also been used for efficient energy management systems by assigning various home network tasks to appropriate components [5], [6]. Flexible home automation architectures utilizing Wi-Fi and ZigBee have been used to monitor and control home appliances as well [7], [8].

Moreover, ZigBee has been used to control home appliances using wireless controllable power outlet systems, and to increase the efficiency of power outlet systems [9], [10]. ZigBee radios have also been used for human presence detection and for the design of smart power outlets and smart switches [11]. A flexible architecture for controlling all home appliances by using ZigBee and IR technology was also presented in [12].

The wide use of ZigBee and its derivative standards makes this protocol an excellent candidate for developing new types of user interfaces that have emerged due to the availability of low-cost microcontrollers and sensors. While low-power networking protocols like ZigBee have revolutionized the wireless aspect of remote controllers, the form of input devices and remote controllers themselves has not evolved much. Most remote controls are boxes with buttons or touch screens where buttons or touch screens can be programmed to control other devices. The availability of cheap MEMS sensors has recently enabled the construction of gesture-based input devices (GBIDs) for computer games. Unlike a keyboard, GBIDs rely on a user making physical gestures which are then used as input. Most gaming input devices use proprietary wireless network protocols and typically use sensors like cameras, accelerometers and gyros to recognize user gestures. Examples of GBIDs for gaming are Kinect for Xbox 360 [13] and Wii [14] in contrast to Xbox remote [15] which is a button-based interface.

This thesis introduces the design, implementation and evaluation of a new class of gesture-based input devices called ZigBee-based Tangible Input Devices (ZTIDs). ZTIDs are small, light-weight graspable devices that can be used to build a new class of tangible interfaces with surface behaviors like rolling, tapping and rubbing. As shown in Figure 1, a person holds such a device in their hand and enacts behavior by rolling, rubbing or tapping the device on a surface like a wall, an appliance, or a table.

These devices are fundamentally different from currently available input devices. Typical input devices normally used are wireless and graspable but not tangible. A tangible device gives physical form to digital information, employing physical artifacts both as representations and controls for computational media [16]. In typical input devices used as remote controls, physical form holds little “representational” significance. For example, turning down lights on an inner wall of

a home is done through a wireless remote control and is not ‘connected’ in any way to the ‘physicality’ of the wall. Alternatively, when using a tangible input device to turn down lights, a person would, for example, roll the device on the wall up or down to change the luminescence of the lighting connected to the wall. As the Western world ages, input devices like ZTID have the additional advantage of being geared towards senior citizens who find it difficult to use or read the small buttons on most current remote controls.

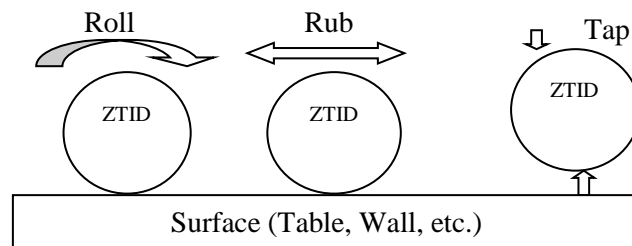


Figure 1: Surface behaviors for ZTIDs

Conceptually, ZTID can be considered a type of tangible user interface (TUI). TUIs typically couple physical representations like the ZTID device and surface with digital representations that may consist of controllers/control actuator signals. Actions on the physical representations manifest themselves as changes in the digital representation that change the state of the user environment. For example, rolling the ZTID device on an appliance (physical representation) may result in changes in the state of the appliance like raising the thermostat set-point (digital representation). As Figure 2 shows, one requirement of a tangible interface is that the physical representation embodies mechanisms for interactive control [16]. This is clearly the case for both normal remote controls as well as ZTIDs. However, for a TUI the physical representation must also be computationally and perceptually coupled with the underlying digital information [16].

These requirements are not met in a typical user input device because, for example, a button pushed on a remote control to raise a home curtain is computationally coupled but not perceptually coupled to the physical representation of an automated curtain. This is because pushing a button has no perceptual correspondence to how a curtain moves up or down in either direction. In a tangible interface, the home-owner would roll their device on the curtain in the right direction

to raise or lower the curtain by physically interacting with the curtain; both computational and perceptual coupling are present.

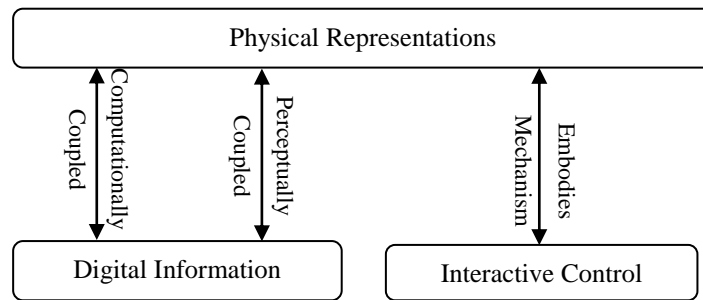


Figure 2: Requirement of a tangible user interface

Figure 3 shows the conceptual design for ZTID. Wireless networking is implemented using the ZigBee protocols while 3D accelerometer sensors are used to eventually detect the various types of rolling, rubbing and tapping behaviors.

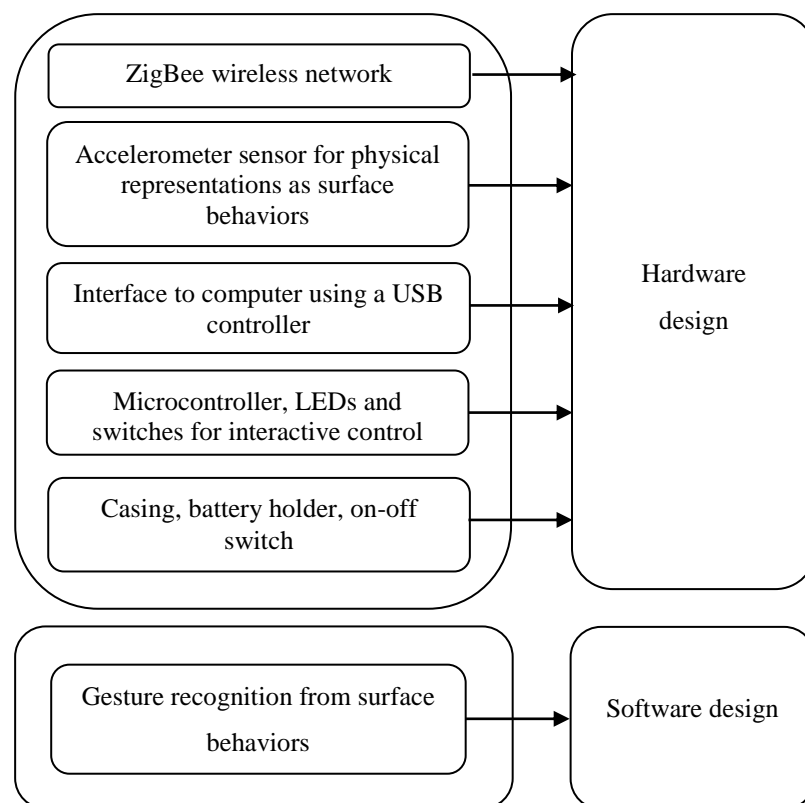


Figure 3: Conceptual design for ZTID

A universal serial bus interface (USB) is also provided. A microcontroller and LEDs to show the behavioral state of the ZTID are also included to provide direct feedback to the user. Finally, an appropriate casing is required to encase the hardware. On the software side, an efficient gesture recognition mechanism is required that uses the accelerometer data to recognize and distinguish between the various types of gestures (e.g., rolling, tapping, or rubbing). The conceptual design of ZTID is implemented in terms of hardware and software designs. The outcomes of the work done for the ZTID design are briefly described in the thesis contribution.

### **1.1. Thesis Contribution**

In this thesis, a new class of gesture-based input devices are conceived, designed, and implemented. The work in this thesis selects and utilizes a well-known machine learning technique known as hidden Markov models. The work shows that hidden Markov models can be trained from a new class of surface gestures which originates from the ZTID design. The receiver operating characteristics (ROC) of the trained hidden Markov models have a recognition accuracy of 99.7 percent. The high recognition accuracy for the unknown surface gestures provides proof that these surface gestures can be used to control other appliances (automated lights, doors, switches, and home appliances etc.) connected in a ZigBee-based wireless network. The major achievements of the thesis are:

- Hardware design, implementation and testing of ZTIDs that involves printed circuit board (PCB) designs for the smallest PCB form factor, manufacturing and assembly of PCBs, ZTID casing selection and modifications to accommodate the manufactured PCBs, final device assembly with batteries for device power and an on-off switch included and testing of four ZTIDs for a ZigBee-based wireless network, interfaces for reprogram-ability and a USB.
- Introducing a new class of surface gestures for controlling user environment and appliances. The gestures are roll, rub, tap and their derivatives like tap once, tap twice, etc.
- Software design and evaluation using HMMs that are trained for each defined surface gesture using the Baum-Welch algorithm. The trained

HMM models are evaluated using the HMM forward algorithm with a random sub-sampling approach.

## **1.2. Thesis Organization**

Chapter 2 contains a literature review where the high-level technology requirements for a tangible user interface are identified and the software and hardware solutions to implement these technology requirements for ZTID are selected. Chapter 3 explains the hardware design using the selected casing, PCB designs, surface mount parts, assembly requirements, outsourcing and manufacturing of PCBs and the final ZTID assembly. Chapter 4 is about the software design. This chapter explains how the selected software methods were used in the software design. The selected methods are for filtering, feature extraction and segmentation and for the classification problem. Chapter 5 is about device hardware and software evaluation and the results obtained from the evaluation. Chapter 6 discusses ZTID applications. In Chapter 7 a conclusion is given with some recommendations. The thesis supporting material and codes are given in the Appendices.

## Chapter 2. Literature Review

In this chapter the high level technology requirements for a tangible user interface are identified and the software and hardware solutions to implement these technology requirements for ZTID are selected. To identify the requirements for a tangible user interface, some existing user interfaces are explored. The literature review is categorized in to the review for hardware and software design solutions. The selected solutions to meet the ZTID conceptual design are based on the discussions given in this chapter. The chapter ends with a summary of selected solutions that are later implemented for ZTID design.

Kinect for Xbox 360 is game console that has a 3D camera to map the exact position of user hands, fingers, feet, head, nose and everything in a 3D map [13]. This allows the user to control the game by using only the body, in great detail, and no controller is needed. It incorporates gesture recognition, voice and facial recognition and supports complex video chat.

Wii is a video game console by Nintendo [14]. A distinguishing feature of Wii is a wireless remote controller, the Wii remote which can be used as handheld pointing device and detects movements in three dimensions. Wii remote uses accelerometers to detect motions in three dimensions. The gaming user interface Xbox 360 uses an Xbox remote to control the game environment [15]. The Xbox remote is a button-based interface that uses Wi-Fi for the wireless network. Some existing user interfaces are given in Table 1.

Table 1: Existing user interfaces [13], [14], and [15]

| <b>Manufacturer</b> | <b>Video game console</b> | <b>Remote</b> | <b>User interface/technique</b>  | <b>Connectivity</b> |
|---------------------|---------------------------|---------------|----------------------------------|---------------------|
| Microsoft           | Xbox 360                  | Xbox remote   | Button based: use buttons        | Wi-Fi               |
| Microsoft           | Kinect for Xbox 360       |               | Gesture based: use 3D camera     |                     |
| Nintendo            | Wii                       | Wii remote    | Gesture based: use motion sensor | Bluetooth           |

Wii remote and Kinect for Xbox 360 are examples of gesture-based input devices (GBIDs). Xbox remote is an example of non-gesture-based device that uses buttons to control the game environment. For a tangible user interface (TUI), the physical representation must be computationally and perceptually coupled with the underlying digital information [16]. These requirements are not met in Xbox remote. A button pushed on Xbox remote to control the game environment is computationally coupled but not perceptually coupled to the controls in a game environment. Pushing buttons has no perceptual correspondence to how the game user presses the buttons. However in a Wii controller, a game user can interact with the game by physically grabbing the device and controlling the game actions for example, cut with sword, push back, move an object by physically grabbing and moving the Wii controller; thus, both computational and perceptual coupling are present.

The Wii remote uses a motion sensor for gesture recognition and the Bluetooth network for wireless connectivity. The example of Wii controllers shows that graspable TUI design poses a problem of wireless network and gesture recognition that needs to be solved on the hardware and software level. The goal of ZTID conceptual design was to achieve a graspable TUI that is surface-based. The review of literature for ZTID design is therefore related to hardware and software designs for wireless networks and gesture recognition.

## **2.1. Hardware Design**

This section describes the hardware design options that were considered for the ZTID design. The hardware design options were considered for the wireless network design and implementation. These options are for selecting the network, network hardware and software, PCB designs and the schematic solutions for network implementation. The selected solutions are described in the discussion section. There are several network options available like Bluetooth, Wi-Fi and ZigBee.

Table 2 shows a comparison of network solutions. The comparison of network solutions shows that ZigBee is used for sensor-based remote controls. Table 2 shows that ZigBee offers very low standby current and the number of devices can be added easily to enlarge the network. Hence, it is best suited for remote control devices that use small batteries for power. Table 3 shows the ZigBee market and application areas [17].

Table 2: Comparison of network solutions [17]

| <b>Features</b>     | <b>ZigBee</b>   | <b>Bluetooth</b>  | <b>Wi-Fi</b>  |
|---------------------|---|---|---|
| Category            | WPAN  | WPAN  | WLAN  |
| Standard            | 802.15.4  | 802.15.1  | 802.11  |
| Data rate           | 250 kbps  | 750 kbps  | Up to 54 Mbps   |
| Transmit current    | 30 mA   | 40 mA   | 400+ mA   |
| Standby current     | 3 uA  | 200 uA  | 20 mA   |
| Memory              | 32-55 KB<br>Memory  | 200+ KB<br>Memory   | 100+KB Memory   |
| Devices per network | 2 <sup>16</sup>   | 8   |   |
| Main applications   | Lighting,<br>sensors,<br>remote<br>controls,<br>peripherals | Cable<br>replacement for<br>voice/audio<br>phones, laptops,<br>headsets | Wireless<br>networking for<br>enterprise and home<br>, intranet, internet |

Table 3: ZigBee market and application areas [17]

| <b>ZigBee market</b> | <b>Applications</b>  |
|----------------------|--|
| Building automation  | HVAC control, asset tracking, building lighting                          |
| Personal health care | Monitoring, fitness, personal, hospital, patient                         |
| Industrial control   | Process control, environmental monitor, energy management, metering      |
| Commercial security  | Home security, lawn irrigation, home remote control, home remote monitor |
| PC and peripherals   | Mouse, keyboard, joystick  |
| Consumer electronics | TV, VCR, DVD/CD, remote control, interactive toys                        |

ZigBee is the best among all the wireless technologies because of its low cost and low power consumption [18]. There are several ZigBee modules available from different manufacturers. Figure 4 shows three ZigBee modules from XBee, Freescale and Atmel.



Figure 4: ZigBee network module options [19], [20], [21], and [22]

To find the cheapest ZigBee module among XBee, Freescale and Atmel, a part level cost comparison was carried out that is given in Table 4. The cost comparison shows that Freescale offers the cheapest ZigBee solution. Freescale also provides extensive development support for ZigBee applications. Based on the analysis, a Freescale ZigBee network starter kit [23] was ordered to explore options for network development.

Freescale provides various software solutions for network development. The software solutions typically consist of a hardware layer, physical layer, media access control layer, network layer and application layer as shown in Figure 5. The SMAC supports point-to-point and star networks and has a very low memory requirement. Moreover SMAC is easy to implement and has no software development fee.

Freescale provides BeeKit as a free GUI for network development [24]. In the BeeKit GUI, the user can select any of the network solutions given in Figure 5 and can configure the network using BeeKit. The features offered by the network software solutions in Figure 5 are given in Table 5.

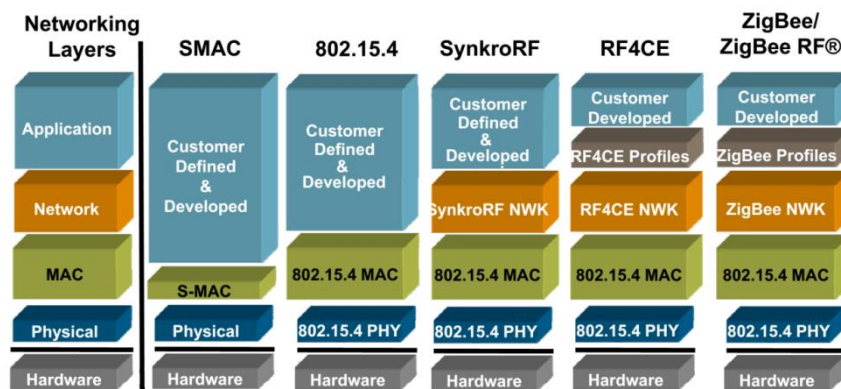


Figure 5: Software solutions for ZigBee [25]

Table 4: Cost comparison of ZigBee modules (Jan. 17, 2015)

|                         | Qty                 | Part             | Value          | Description        | Unit Price      | Price per Qty |             |
|-------------------------|---------------------|------------------|----------------|--------------------|-----------------|---------------|-------------|
| <b>XBee</b>             | 1                   | XBee             | XBP24-AWI-001  | ZigBee platform    | 19              | 19            |             |
|                         | 1                   | U1               | Atmega328      | controller         | 3.04            | 3.04          |             |
|                         | 1                   | Y1               | 20MHZ          | Crystal            | 0.4             | 0.4           |             |
|                         | 1                   | C3               | 10uF           | Capacitor          | 0.39            | 0.39          |             |
|                         | 2                   | C1,C4            | 0.1uF          | Capacitor          | 0.06            | 0.13          |             |
|                         | 1                   | R2               | 10K            | Resistor           | 0.03            | 0.03          |             |
|                         | <b>Total (\$)</b>   |                  |                |                    |                 |               | <b>23</b>   |
| <b>Freescale-ZigBee</b> | 5                   | C2,C5,C6,C12,C13 | 0.1uf          | Capacitor          | 0.33            | 1.65          |             |
|                         | 2                   | C3,C4            | 8pf            | Capacitor          | 0.01            | 0.03          |             |
|                         | 1                   | Y1               | 16Mhz          | Crystal            | 0.68            | 0.68          |             |
|                         | 1                   | L3               | 8.2nH          | Inductor           | 0.12            | 0.12          |             |
|                         | 1                   | L1               | 4.7nH          | Inductor           | 0.05            | 0.05          |             |
|                         | 1                   | T1               | 50 ohm         | Balun              | 0.28            | 0.28          |             |
|                         | 1                   | C8f              | 0.5pF          | Capacitor          | 0.03            | 0.03          |             |
|                         | 2                   | C15,C16          | 10pF           | Capacitor          | 0.06            | 0.12          |             |
|                         | 1                   | U1               | MC13213        | ZigBee platform    | 4.72            | 4.72          |             |
|                         | 1                   | A1               | chip           | antenna            | 1.05            | 1.05          |             |
|                         | 1                   | R7               | 470K           | Resistor           | 0.02            | 0.02          |             |
|                         | 1                   | C19              | 10uf           | Capacitor          | 0.15            | 0.15          |             |
|                         | <b>Total (\$)</b>   |                  |                |                    |                 |               | <b>8.94</b> |
|                         | <b>Atmel-ZigBee</b> | 1                | ATmega128R FA1 | ATmega128R FA 1-ZU | ZigBee platform | 5.73          | 5.73        |
| 1                       |                     | B1               | 2.4GHz         | SMD balun          | 0.84            | 0.84          |             |
| 4                       |                     | CB1,CB2,CB3,CB4  | 1uF            | Capacitor          | 0.07            | 0.30          |             |
| 4                       |                     | CX1,CX2,CX3,CX4  | 12pF           | Capacitor          | 0.06            | 0.24          |             |
| 2                       |                     | C1n,C2n          | 22pF           | Capacitor          | 0.07            | 0.14          |             |
| 1                       |                     | C4n (optional)   | 0.47pF         | Capacitor          | 0.03            | 0.03          |             |
| 1                       |                     | R1n              | 680            | Resistor           | 0.07            | 0.07          |             |
| 1                       |                     | XTAL1            | 16MHz          | Crystal            | 2.19            | 2.19          |             |
| 1                       |                     | XTAL2            | 32Khz          | Crystal            | 1.88            | 1.88          |             |
| <b>Total (\$)</b>       |                     |                  |                |                    |                 | <b>11.43</b>  |             |

Table 5: Features of software solutions for ZigBee [25]

| Features                 | SMAC                    | 802.15.4 MAC           | SynkroRF                 | ZigBee RF4CE                   | ZigBee/ZigBee Pro   |
|--------------------------|-------------------------|------------------------|--------------------------|--------------------------------|---------------------|
| Typical applications     | Cable replacement       | Wireless control       | Cable replacement        | RF remote control              | Home automation     |
|                          | Wireless toys and games | Wireless automation    | Wireless control         | Home entertainment and control | Smart energy        |
|                          |                         | Wireless meter reading |                          | Home automation                | Building automation |
|                          |                         |                        |                          |                                | Health care         |
| Network stack            | No                      | No                     | Yes                      | Yes                            | Yes                 |
| Software development fee | No                      | No                     | No                       | Yes                            | Yes                 |
| Network profiles         | No                      | No                     | No                       | Yes                            | Yes                 |
| Memory requirements      | 4-8K                    | 40-50K                 | 32K                      | <40K                           | 50-100K             |
| Network topology         | Point to point          | Peer to peer           | Co-existing star         | Co-existing star               | Tree                |
|                          | Star                    | Tree                   |                          |                                | Mesh                |
|                          |                         | Mesh                   |                          |                                |                     |
| Typical # of nodes       | 2-100                   | 2-1000                 | 32 per controlled device | 32 per target device           | 2-250 ZigBee        |
|                          |                         |                        |                          |                                | 2-1000 ZigBee Pro   |
| Typical IC cost          | \$1-2                   | \$2-3                  | \$2-3                    | \$2-3                          | \$3-5               |
| Typical data throughput  | 50-115K                 | 90-115K                | 70-100K                  | 70-100K                        | 30-70K              |

BeeKit supports solutions for three main Freescale platforms, i.e., MC1321x, MC1323x and MC1322x as given in Table 6. The hardware platforms can be selected in BeeKit. Both MC1321x and MC1323x have an 8-bit micro controller integrated with a 2.4 GHz radio integrated in a single low-grid array (LGA) package. MC1322x has a 32 bit ARM 7 micro controller integrated with a 2.4 GHz radio in a single LGA package.




Freescale provides PCB reference designs for its three main ZigBee platforms i.e. MC1321x, MC1323x and MC1322x. The MC13213 platform supports all main

network solutions given in Table 6. Moreover, the MC1321x platform is provided with the widest support for network development in terms of both hardware and software reference designs. For MC1321x technology, Freescale provides five PCB solutions that include the PCB designs and the schematics. The three PCB solutions ICB [21], IBP [22], and UCB [26] (no name) are form factor reference designs. The other two PCB solutions are network coordinator board (NCB) and sensor reference board (SRB) [23]. The PCB solutions are provided as reference designs. The alternate PCB form factor solutions are given in Table 7.

Table 6: Supported hardware for network solutions [25]

| Hardware Platform | SMAC | IEEE 802.15.4 | SynkroRF  | ZigBee RF4CE | ZigBee    |            |
|-------------------|------|---------------|-----------|--------------|-----------|------------|
|                   |      |               |           |              | ZigBee    | ZigBee Pro |
| MC13201           | Yes  | No            | No        | No           | No        | No         |
| MC13202           | Yes  | HCS08 MCU     | HCS08 MCU | HCS08 MCU    | HCS08 MCU | No         |
| MC13211           | Yes  | No            | No        | No           | No        | No         |
| MC13212           | Yes  | No            | No        | No           | No        | No         |
| MC13213           | Yes  | Yes           | Yes       | Yes          | Yes       | No         |
| MC13224           | Yes  | Yes           | Yes       | Yes          | Yes       | Yes        |
| MC1323x           | Yes  | Yes           | Yes       | Yes          | Yes       | Yes        |

Table 7: PCB form factor solutions [21], [22], and [26]

| Features     | ICB   | IBP  | UCB   |
|--------------|---|--|---|
| Layers       | 2   | 4  | 4   |
| PCB          | FR4   | FR4  | FR4   |
| Antenna      | Chip  | Printed F  | Chip  |
| Output power | +2.5 dBm  | +2.5 dBm   | +2.5 dBm  |
| Range        | 190m line of sight  | 190m line of sight   | 130 m line of sight   |
| Dimensions   | 30.5x21.6 mm  | 35.6x25.4 mm   | 53.34x16.8 mm   |
| Image        |  |  |  |

The IBP provides the network solution for MC1321x with printed F-antennae. The ICB and UCB provide the network solution for MC1321x with chip antennae [27]. ICB is a two-layer design compared to UCB that has four layers. The ICB design offers good range and output power compared to UCB as given in the comparison of Table 7. The schematic for the ICB network solution is given in Figure 6.

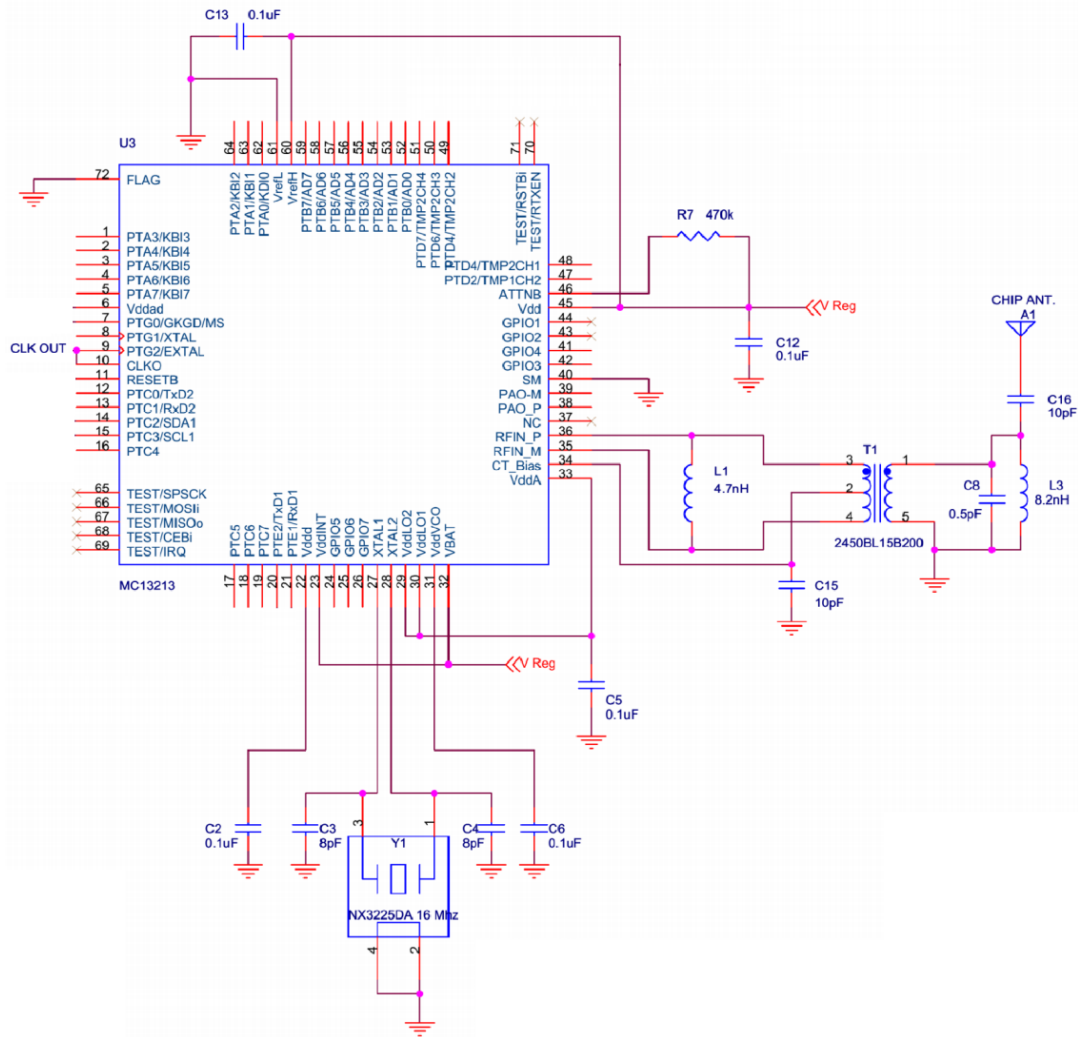


Figure 6: Network hardware solution for MC1321x [21]

MC1321x can be reprogrammed for any supported network solution using a BDM programming port. The BDM port provides re-programming of the MC1321x solution using its back ground debug module [28]. The BDM port option is available on all reference designs NCB, SRB, ICB, UCB and IBP for MC1321x. The schematic for the BDM port on ICB is given in Figure 7.



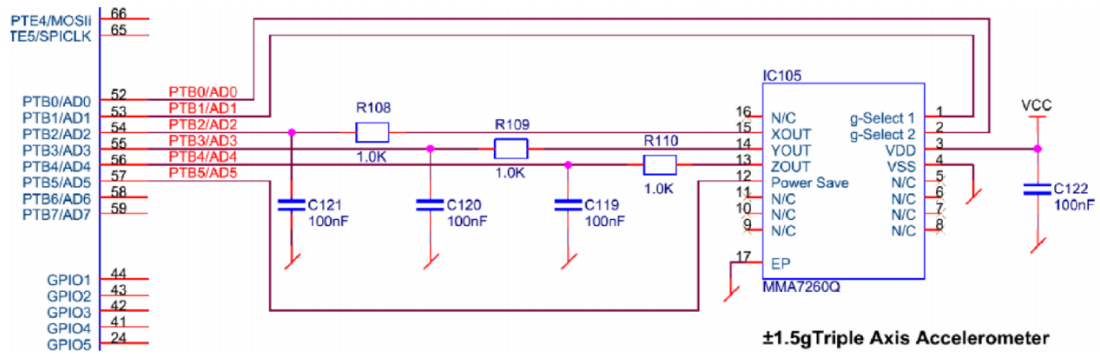


Figure 9: MMA7260Q interfaced with MC1321x [23]

The SRB [23] uses four switches and four LEDs interfaced with the MC1321x GPIO. The schematic for the switches and LED interface is given in Figure 10.

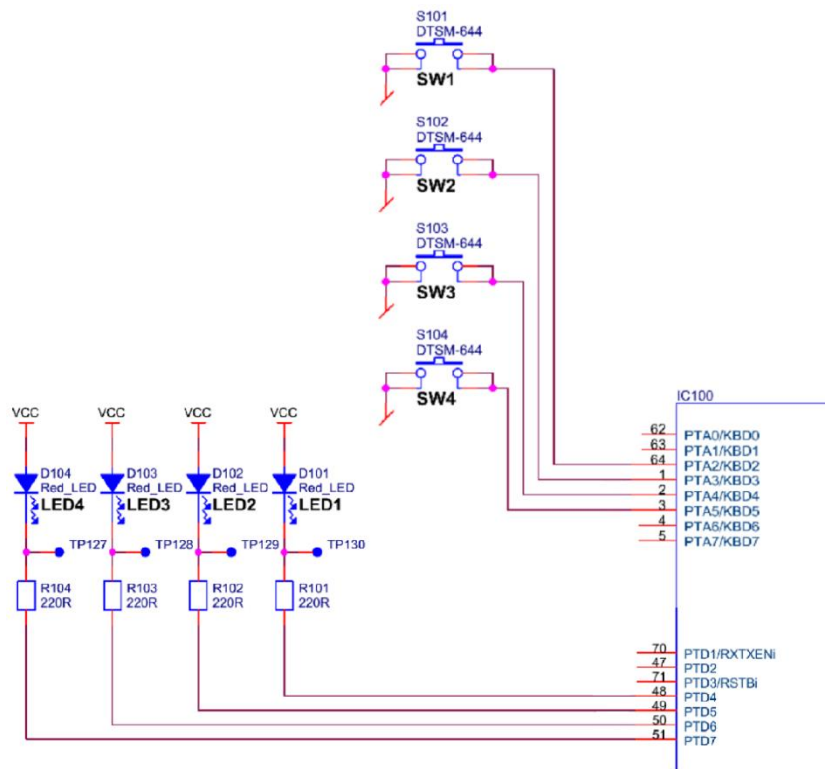


Figure 10: LEDs and switches interfaced with MC1321x [23]

### 2.1.1. Discussion

The comparison of network solutions in Table 2 shows that ZigBee offers low standby current and the number of devices can be added to enlarge the network. The comparison also shows that ZigBee is used for sensor-based remote controls. Table 3

shows that ZigBee is used for home and building automation, industrial control, health care, consumer electronics and commercial security. ZigBee has emerged as a dominant network solution especially for low-power remote control applications. Thus, we decided to use ZigBee for the ZTID design.

Among the ZigBee modules, many options are available like Atmel ZigBee, Freescale and XBee. A cost comparison on the part level of these three ZigBee modules is given in Table 4. The cost comparison shows that Freescale offers the cheapest ZigBee solution. Freescale also provides extensive support for development for ZigBee hardware and software applications [25]. Based on the cost analysis, hardware and software reference design availability and extensive support for network development, Freescale ZigBee network solution MC1321x [31] is selected for ZTID.

Freescale MC1321x comes in three packages, MC13211, MC13212 and MC13213 depending on the Flash and the RAM size. MC13213 offers the full ZigBee solution. For MC13213 the flash size is 60K and the RAM size is 4KB. MC13213 is a good option among the other ZigBee platforms given in Table 6 because it is not only capable of implementing all network designs given in Table 5 but also comes with extensive design support. Moreover this platform is a low grid array (LGA) package. LGA packages can be implemented on a two-layer PCB. Ball grid array packages (BGAs) need more than two PCB layers. Based on this analysis, MC13213 is selected for ZTID design. The block diagram of MC13213 is given in Appendix A. For network implementation, the SMAC solution from Table 5 is selected for ZTID. The SMAC solution is easy to implement, offers low memory requirements with point-to-point and star networking, comes with source code and has no software development fee. The network implementation method for the selected SMAC network solution using the selected hardware platform MC13213 for ZTID is given in Appendix B.

From the PCB form factor solutions, ICB [21] is selected for the ZTID. Table 7 shows the PCB form factor solutions. The IBP solution [22] provides the network solution for MC1321x with printed F-antennae. The ICB [21] and UCB [26] solution provide the network solution for MC1321x with chip antenna. The ICB and UCB provide the smallest form factor solution possible since they use chip antennae [27]. But ICB is a two-layer design compared to UCB that has four layers, so ICB is easier to implement for a PCB design. The ICB design offers good range and output power compared to UCB as given in the comparison of Table 7.

The schematics for the design solutions for ICB [21] and SRB [23] show that ICB design provides a solution for the network and SRB design provides a solution for the accelerometer, USB controller interface and also for LEDs and switches.

For the network design, ICB [21], UCB [26] and IBP [22] schematic solutions were compared using antenna design guidelines [27]. It was found that chip antennae offer the lowest possible PCB area among different antenna configurations. So it was decided to use chip antennae on a two-layer PCB for the ZTID.

ZTID schematic design for the accelerometer, USB controller and LED-switches was based on Freescale SRB [23] so that no changes were required on the stack level. It makes ZTIDs compatible with all network stacks offered by Freescale for MC1321x.

For the ZTID casing, an existing casing solution is selected based on the dimensions of the selected ICB design [21]. The width for the ICB is 21.6 mm. So a plastic casing was selected with an inner diameter of 22 mm. That was the closest possible. The selected casing is from Camelion [32] which is ultra-shock resistant and gave an excellent plastic casing solution for the ZTID.

## **2.2. Software Design**

On software level, the implementation of tangible interface poses the requirement to implement gesture recognition. Next, the prior research on gesture recognition is described in terms of the problems they solve, methods used for gesture recognition and their evaluation. For ZTID gesture recognition, all the methods were considered and those selected were the methods that could be easily implemented on the device hardware.

A real-time system for detecting and recognizing lower body activities was presented in [33]. For data collection, a single tri-axis accelerometer is used. Non-parametric method principle component analysis is used and through a number of experiments it is shown that FFT and wavelet features do not offer good results. For classification, the adaptive boosting algorithm (AdaBoost) built on decision stumps is used. An accuracy of 95% is achieved on a set of continuous data and activity is recognized every half a second using a PC running Matlab. It is found that recognition is highly dependent on the position where the accelerometer is attached. An ankle

accelerometer cannot give sufficient information for finding whether a person is sitting or standing because the confusion matrix does not have necessary information.

The classification and evaluation of human posture and activities using a hybrid classifier decision tree (DT) and neural networks (NN) was presented in [34]. For data collection, an accelerometer is used. For filtering, an elliptic IIR low pass for removing jittering noise and an elliptic IIR high pass for removing the gravity component are implemented. A hybrid classifier gave an overall accuracy of 98.99%.

An accelerometer-based traffic police gesture recognition system was proposed in [35]. For filtering, a 3rd-order Butterworth low pass filter is implemented. For classification, template generation and fuzzy rule based classifier is used. The classifier evaluation is not given but the results show that 8 out of the 9 gestures achieved a 100% recognition rate.

A neuro fuzzy classifier was used in [36]. Although the classifier evaluation is not given, the final result of classifying 25 gestures by extracting 19 features and using a neuro fuzzy system was 100%. The gestures which are able to control an appliance with maximum accuracy are also found.

A hybrid classifier using AdaBoost, a support vector machine (SVM), and RlogReg was presented in [37]. The performance of Ada-Boost, SVM and log-Reg with cases of subject-dependent, subject-independent and subject-adaptive is evaluated. Non-parametric methods, i.e., time-based, frequency-based and first-order derivative of acceleration data for feature extraction are explored. It is found that there is an improvement of 3% accuracy when both features are combined, i.e., non-parametric combined with the first-order derivative feature. The accuracy was 89.8% for standard, 81.9% for first-order and 92.8% for both.

The presence of fatigue in a subject wearing an accelerometer-based device with Blue-tooth was evaluated by [38]. Feature point extraction (FPE) is used to find the order of autoregressive-coefficients (AR). For the classification, ANN is used and the overall classification accuracy was 70%. The network was successful in identifying the presence of fatigue.

Recognition of daily human activities using a single accelerometer was done by [39] through the use of linear discriminant analysis (LDA) for activity recognition. For classification, a multi-layer back propagation neural network (Levenberg-Marquardt back propagation for training) is used. Four activities were recognized, of

which three were dynamic and the fourth one was static with little mismatch with the dynamic activity.

Dynamic time warping (DTW) is used by [40] in which a recognition algorithm for interaction based on gesture was presented. DTW is based on dynamic programming to match two time series with temporal dynamics. An algorithm “uWave” was presented that used a single three-axis accelerometer. The evaluation was done using a large gesture library with over 4000 samples for eight gesture patterns collected from eight users over one month. The uWave algorithm achieved 98.6% accuracy.

A tangible cube was designed and implemented by [41] in which the purpose was to control the user environment. By using an accelerometer, the concept of a cube-tangible interface was presented to function as a wireless remote control. A finite state machine model was made to show how hands gestures can be used to function as a remote controller. The device evaluation was not mentioned.

A comparative study was done on the classification techniques of BDM, LSM, k-NN, DTW, SVM, and ANN by [42]. Non-parametric methods (PCA-SFFS) are used. It is found that the classification technique BDM has the highest classification rate and classification time for ANN is the smallest, followed by LSM, BDM, SVM, and DTW methods.

Real time human activities were recognized by [43]. AR-coefficients, signal magnitude area, and tilt angle features are used for activity recognition and linear discriminant analysis is used for state recognition. An ANN classifier is used for which the overall recognition accuracy rate was 84.8%. The recognition results for sitting and standing were 76.6% and 72.4% respectively.

A neuro fuzzy classifier was used for human activity recognition by [44]. For feature reduction, feature subset selection (FSS) and LDA are used. For classification, the average recognition accuracy was 83.41+-5.93%. It is found that the LDA method outperforms the FSS method.

A hierarchical scheme is used by [45] for human activity classification by first finding the state and then activity that belongs to that state. The accelerometer data was filtered using a moving average filter. For classification, ANN is used with AR-coefficients, signal magnitude area, and tilt angle for activity recognition and linear discriminant analysis is used for state recognition. It is shown that a hierarchical

scheme is superior to single-level recognition as a large number of activities result in complex decision boundaries in the feature space which are difficult for a single classifier to solve.

Human activities that are independent of accelerometer sensor attachment to a specific part of a body were recognized by [46]. For feature reduction linear discriminant analysis is used. It is found from experimentation that spectral entropy affects the upper and lower body positions, and therefore signal magnitude area and AR coefficients are recalculated after upper and lower body identification. For classification ANNs are used with the average recognition accuracy being 94.4%.

The performance of a signal-based and model-based decision tree classifier was compared in [47]. It is found that the combine classifier of signal- and model-based features can improve feature recognition. For feature segmentation, a K-mean algorithm is used.

A novel algorithm for gesture classification using SOM and BMU is presented in [48]. A K-nearest neighborhood algorithm and distance measurement derived from the Kullback-Leibler-Divergence is used. The overall classification rate was 80%, while that with the support vector machine had a slightly worse classification result of 76%. The classification resulted after 90 training cycles.

Human activity recognition using a mobile phone with varying position and orientation was studied in [49]. For feature extraction mean, variance, correlation, FFT energy and frequency-domain entropy are used. For classification state vector machines are used with an accuracy of 93.2%. It is shown that compared with the generic SVM model, location-specific SVM training improves accuracy.

Hybrid classifiers DT, NB, KNN and SVM are used in [50]. A built-in cell phone accelerometer is used for data collection. For filtering and feature segmentation, a moving-average filter and K-mean methods are used. Classifiers for gesture recognition are designed and evaluated for cell phones with the use of non-parametric methods (but more time-based and less frequency-based) by defining vertical, horizontal and magnitude features. It is found that the decision tree achieves the best performance among four commonly used static classifiers, while vertical and horizontal features have better recognition accuracy than magnitude features. Moreover, it is highlighted that if an HMM model or similar Markov model is built

directly on a sequence of extracted features, the overall classification accuracy can be improved.

Real-time hand gesture recognition using HMM is evaluated in [51] by control of a cube game on a PC. Accelerometer and EMG sensors are used for data collection. A set of 18 gestures each trained with 10 repetitions are defined. The overall recognition accuracy was 91.7%.

A real-time hand gesture recognition system was made by [52]. A built in cell phone accelerometer is used for data collection and magnitude of acceleration is used to make the system axis-invariant. The data is filtered using an exponential moving average filter. For feature segmentation a custom definition is defined that is valid for hand gestures only. For classification, HMM and SVM are used. On average, the SVM algorithm had 96% accuracy and the HMM algorithm had 97.4% accuracy.

The wireless device SoapBox was designed and implemented by [53]. Gestures in air in the x and y planes are defined to control a DVD player by using an accelerometer. The optimal threshold for convergence of HMM is found to be 10-30 training cycles with a recognition accuracy of 98 percent.

Five Arabic numbers 0-4 are recognized in real time by [54] using HMM. For motion information, an inertial measurement unit (IMU) is used with an auto-cut algorithm for segmentation. The recognition accuracy is found to be 93%.

A gesture based input device was designed and implemented by [55] and was named as “The Magic Wand” or TMW. For motion information, acceleration data is used from an IMU. Using HMMs, TMW was able to interact with an MP3 player through the use of button-based segmentation.

Accelerometer-based gesture control for a design environment was studied by [56]. Hidden Markov machines were utilized and eight gestures were defined for evaluation purposes. It is found that both ergodic and left-to-right models gives similar results and good results have been obtained in earlier studies by using an ergodic model with five states for a set of gestures. A K-mean algorithm is used for dimensional reduction with a code book size of eight. The recognition accuracy using HMM was 81.2% for 1 and 98.9% for 12 training vectors.

### 2.2.1. Discussion

The literature review shows that gesture recognition is done in a generic architecture of filtering, feature extraction and segmentation and classification. First, the data is collected from a sensor and then it is filtered. The filtering defines the pass band and the stop band for the sensor data. Then features are extracted to reduce the data dimensionality such that it can be processed for training. The feature extraction methods can be categorized in to parametric and non-parametric methods. The parametric methods derive their name from the fact that a distribution is associated with the data. For the non-parametric methods, no distribution is associated with the data. The goal of the classification problem is to train a model based on sensor data such that this model can be used later for control decisions; i.e., the problem of recognition. The steps and the methods that were used for gesture recognition in the literature are summarized in Table 8.

The goal of a good gesture recognition system is to recognize the defined gestures with high accuracy and little time. Table 8 shows that there are many choices for selecting a sensor, filtering technique, feature extraction and segmentation technique and classification technique. The classifiers shown in Table 8 can be implemented with different methods of feature extractors, filters and sensors.

For ZTID gesture recognition, the goal was to use a classifier that can be easily implemented on ZTID device hardware. Although both dynamic time warping (DTW) and hidden Markov models (HMMs) are well-studied non-linear pattern matching algorithms, the research trend moved from DTW to HMM in between 1988-1990 [57]. DTW has a quadratic time and space complexity that limits its use to only small time series data sets [58]. DTW will take too much time on a small microcontroller because it uses dynamic programming. HMMs give higher performance than DTW and human abilities in terms of identifying stress patterns of word utterances [59]. HMMs were used for real time by [51], [54] and for wireless applications by [53], [56]. The number of HMM states can be selected easily because the number of HMM states does not have a significant effect on the gesture recognition results [60], [61]. Moreover, once trained, HMMs can be implemented on a ZTID microcontroller using an efficient forward algorithm. But HMMs require more training data compared to DTW.

Table 8: Gesture recognition steps and alternate methods

| <b>Steps</b>                        | <b>Alternate methods</b>  |
|-------------------------------------|---|
| Sensor for data collection          | Camera<br>Single accelerometer<br>EMG sensor<br>Gyro  |
| Filtering                           | Moving average<br>Elliptic IIR<br>Butterworth<br>Kalman   |
| Feature extraction and segmentation | Autoregressive-coefficients (AR)<br>AR + feature point extraction (FPE)<br>AR + signal magnitude area (SMA) + tilt angle (TA)<br>Linear discriminant analysis (LDA)<br>Feature subset selection (FSS) and LDA<br>Principal component analysis (PCA)<br>K-nearest neighborhood (KNN)<br>K-mean algorithm |
| Classification                      | Neuro fuzzy (NF)<br>Artificial neural network (ANN)<br>Hidden Markov models (HMMs)<br>Decision tree (DT)<br>State vector machine (SVM)<br>Dynamic time warping (DTW)<br>Adaptive boosting (AdaBoost)<br>Finite state machine (FSM)  |

For this thesis, HMMs are decided to be used because lot of data for surface gestures can be generated for which ZTID is specifically designed. Moreover, HMMs were used before for real-time and wireless applications. It was decided to use an accelerometer sensor, moving average filter and K-mean segmentation with HMMs because these methods were used together before and resulted in very high recognition accuracy for wireless applications [53], [56]. A moving average filter with HMM was used by [52] for which the recognition accuracy was 97.4% on average. The highest recognition accuracy of 98.8 % using K-mean segmentation with an HMM was obtained by [56] in which eight gestures were defined with 30 data sets per gesture. The evaluation was based on training and recognition using an HMM by varying the number of training vectors. It was decided to use the same method for evaluation as done by [56]. Eight surface gestures were defined for ZTIDs and the evaluation was done using a forward algorithm by varying number of training vectors

to find the optimum number of required training vectors to achieve recognition accuracy close to 100 percent.

### 2.3. Selected Solutions

The selected hardware and software solutions are given in Table 9. It was decided to design the ZTID hardware on the smallest possible form factor by integrating the selected ZigBee platform, accelerometer, USB controller and casing to achieve a surface-based tangible user interface that is hand-graspable, small, lightweight and low power. The hardware design for ZTID is given in the next chapter (Chapter 3). For ZTID software design, it was decided to define eight surface gestures and train and recognize HMMs for those surface gestures using Matlab. For recognition, a forward algorithm was decided to be used. The software design for ZTID is given in Chapter 4.

Table 9: Selected solutions

| Category        | High level technology requirement | Technology used                 | Selected solution            |
|-----------------|-----------------------------------|---------------------------------|------------------------------|
| <b>Hardware</b> | Networking                        | ZigBee                          | MC13213                      |
|                 | Round casing                      | TPE plastic casing              | Camelion                     |
|                 | Position awareness                | 3D accelerometer                | MMA7361L                     |
|                 | Power source                      | Alkaline batteries              | battery holder from Camelion |
|                 | Reprogram-ability                 | BDM interface                   | MC13213                      |
|                 | Switch                            | on-off                          | Switch from Camelion         |
|                 | Interface to computer             | USB controller                  | CP2102                       |
|                 | Input output interface            | Micro controller switches, LEDs | MC1321x, LEDs pad switches   |
| <b>Software</b> | Filtering                         | Moving average filter           | Matlab                       |
|                 | Feature segmentation              | K-mean algorithm                | Matlab                       |
|                 | Classification                    | HMM, Baum-Welch                 | Matlab                       |
|                 | Evaluation                        | Forward-algorithm               | Matlab                       |

## Chapter 3. Hardware Design

The hardware for the ZTID is implemented using two printed circuit (PC) boards: the RF-control-board and LED-switch-board. As Figure 11 shows, these two boards are encased in a specially-designed casing that also includes a battery holder and an on-off switch. The RF-control-board is the primary controller board of the ZTID shown in Figure 11. The LED-switch-board is the input/output interface for ZTID devices. Table 10 shows how the two proposed PCB designs meet the high level technology requirements with the selected casing.

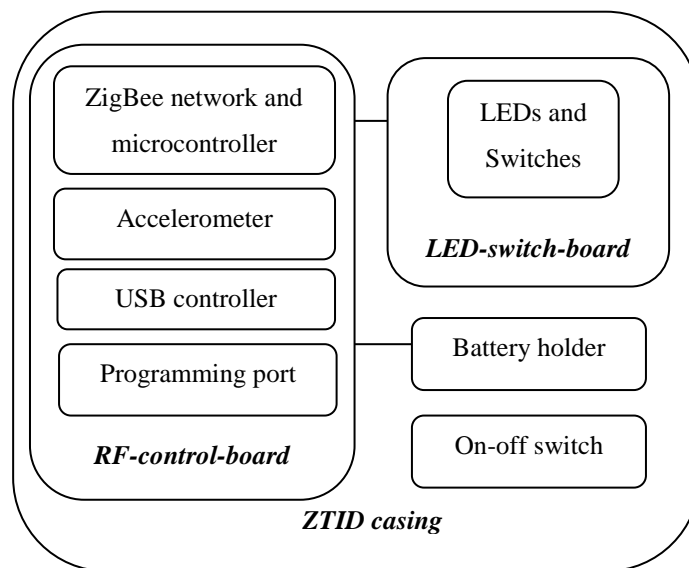


Figure 11: ZTID device hardware architecture

### 3.1. Schematic Design

To implement the high level requirements of Table 10, the schematics are designed for an RF-control-board and LED-switch-board. These schematics are designed in Eagle CAD.

The RF-control-board is the main control board of ZTID with the schematic diagram shown in Figure 12. For the network solution and the BDM interface, an ICB schematic solution is used [21]. For ZTID RF design, an MC13213 internal transmit receive switch is used with a chip antenna for a low cost design. This is done from the guide lines provided for antenna design [27], [62]. The interface for the USB controller CP2102 and the programming interface for the MC13213 both are designed

with two micro-USB ports with the symbols “X1” and “X2” shown in Figure 12. By doing this, a single USB cable can be used for both programming and for connecting the ZTIDs to any computer and also reduces the form factor of the final ZTID. The accelerometer design for the ZTID is based on accelerometer MMA7361L [63]. The USB controller design is based on SRB [23]. For the ZTID power, two power options are provided in the design. One source is from Camelion [32] using batteries “BAT1” and “BAT2” shown in Figure 12. The other source is when the ZTID is connected to any USB port by enabling USB controller CP2102’s internal voltage regulator [29] in the hardware configuration. A jumper is provided in the design shown by “SWITCH” in Figure 12. This jumper enables or disables the power supply option from CP2102. The “I/O port” in Figure 12 is designed for mounting the data cable on the RF-control-board for connecting it to the LED-switch-board. For the required 11 connections shown in Figure 12 for the “I/O port”, a data cable is specially designed for connecting the RF-control-board to the LED-switch-board. This data cable is composed of two ribbon cables each with six connections and an IDC connector. This data cable is soldered on the RF-control-board and is detachable from the LED-switch-board.

Table 10: Requirements for ZTID and the proposed hardware solutions

| <b>High level technology requirement</b> | <b>Technology used</b>          | <b>Selected solution</b>       | <b>PCB design solution</b>         |
|--|---------------------------------|--------------------------------|------------------------------------|
| Networking                               | ZigBee                          | MC13213                        | RF-control-board                   |
| Round casing                             | TPE plastic casing              | Casing from Camelion           |                                    |
| Position awareness                       | Accelerometer                   | MMA7361L                       | RF-control-board                   |
| Power source                             | Alkaline batteries              | battery holder from Camelion   |                                    |
| Programmability                          | BDM interface                   | MC13213                        | RF-control-board                   |
| Switch                                   | on-off                          | Switch from Camelion           |                                    |
| Interface to PC for training             | USB controller                  | CP2102                         | RF-control-board                   |
| Input output interface                   | Micro controller switches, LEDs | MC1321x, SMD LEDs pad switches | LED-switch-board+ RF-control-board |

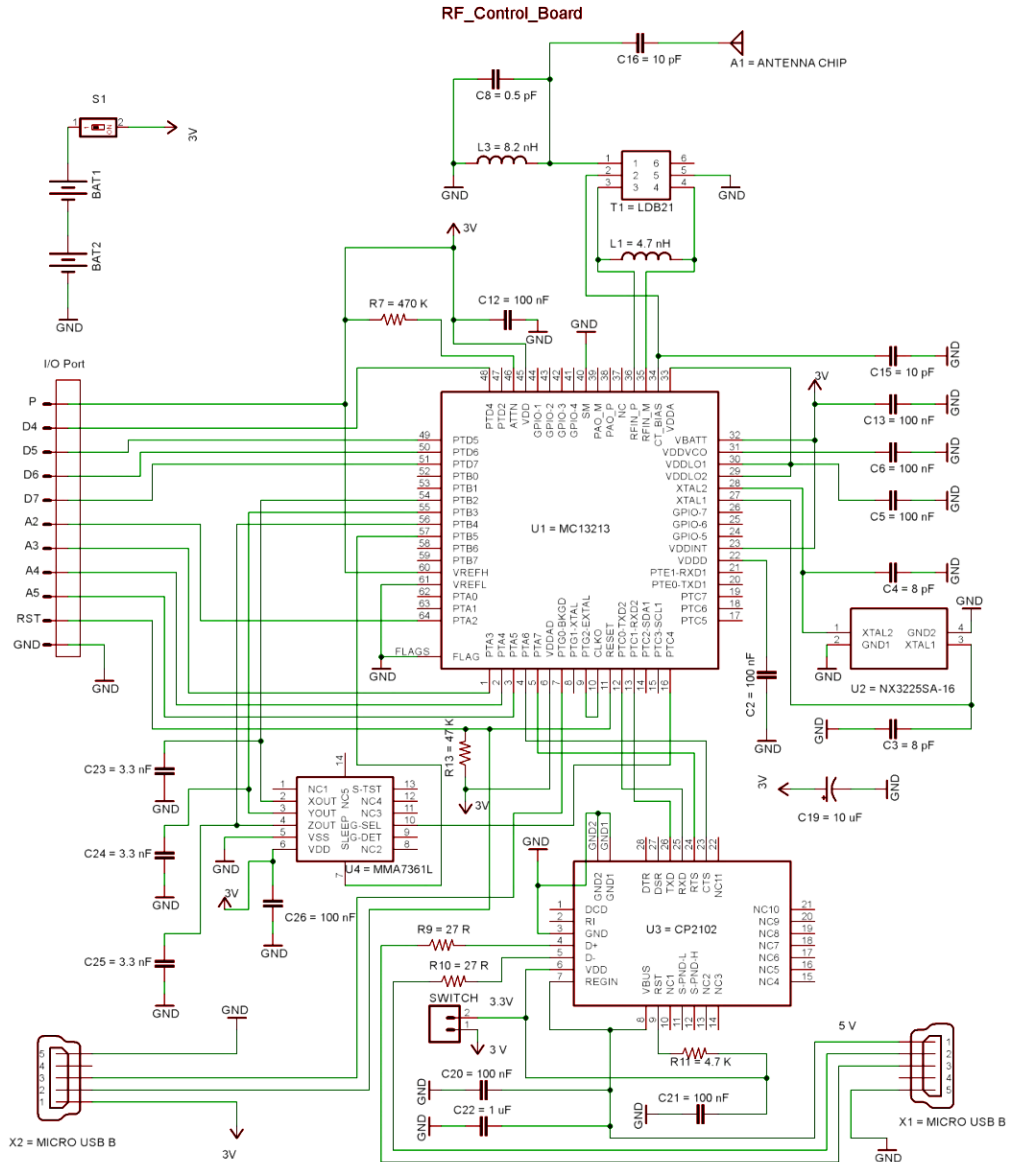


Figure 12: Schematic of RF-control-board

The LED-switch-board is the button based interface for the ZTID. The LED-switch-board schematic is shown in Figure 13. It consists of four surface mount LEDs, four pad switches, and a reset pad switch. The schematic solution is based on an SRB board. The LEDs and switches use the same pin configuration as for the SRB board [23]. Thus network demonstration software for SMAC [64] is compatible with ZTID using the LED-switch-board and RF-control-board. The LED-switch-board has two connectors, J1 and J2, for connecting to data cables coming from RF-control-board.

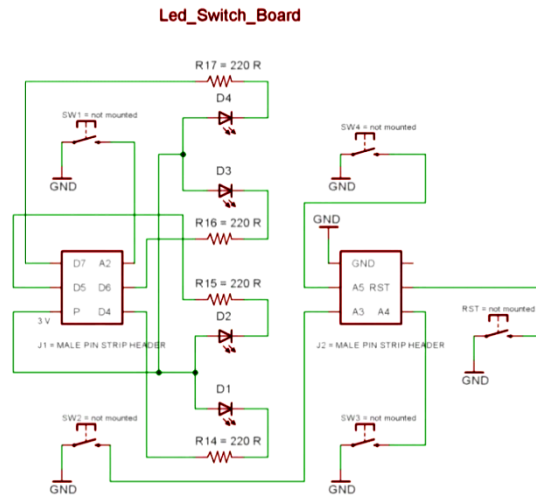


Figure 13: Schematic of LED-switch-board

### 3.2. PCB Design

The PCB is designed for the two ZTID boards using Eagle Cad software. Both the boards have two layers. The design files for each board are described in Table 11.

Table 11: PCB design files description

| Eagle file name | Eagle file description   |
|-----------------|--------------------------|
| CMP             | Top copper layer         |
| SOL             | Bottom copper layer      |
| CRC             | Solder paste top layer   |
| PLC             | Silkscreen top layer     |
| STC             | Solder stop top layer    |
| STS             | Solder stop bottom layer |
| DRL             | Excellon drill file      |

For the RF-control-board, three QFN chip footprints are designed with three layers: CMP, CRC and STC as given in Table 11. The three QFN chips are:

- MC13213: ZigBee network controller
- MMA7361L: Accelerometer
- CP2102: USB controller

The three layers, CMP, CRC and STC are designed by following the guidelines provided by the chip manufacturer. The design files for the RF-control-board are shown in Figure 14. For the MC13213, the layers are designed by following the specific guidelines provided for this platform. Freescale recommends designing

these layers whenever any change in the given reference design is done. So the footprint for the MC13213 is designed using [65]. Similar guidelines are followed to design a chip footprint for accelerometer MMA7361L using [66]. The chip footprint for the USB controller is designed using the guidelines provided by the manufacturer in [29].

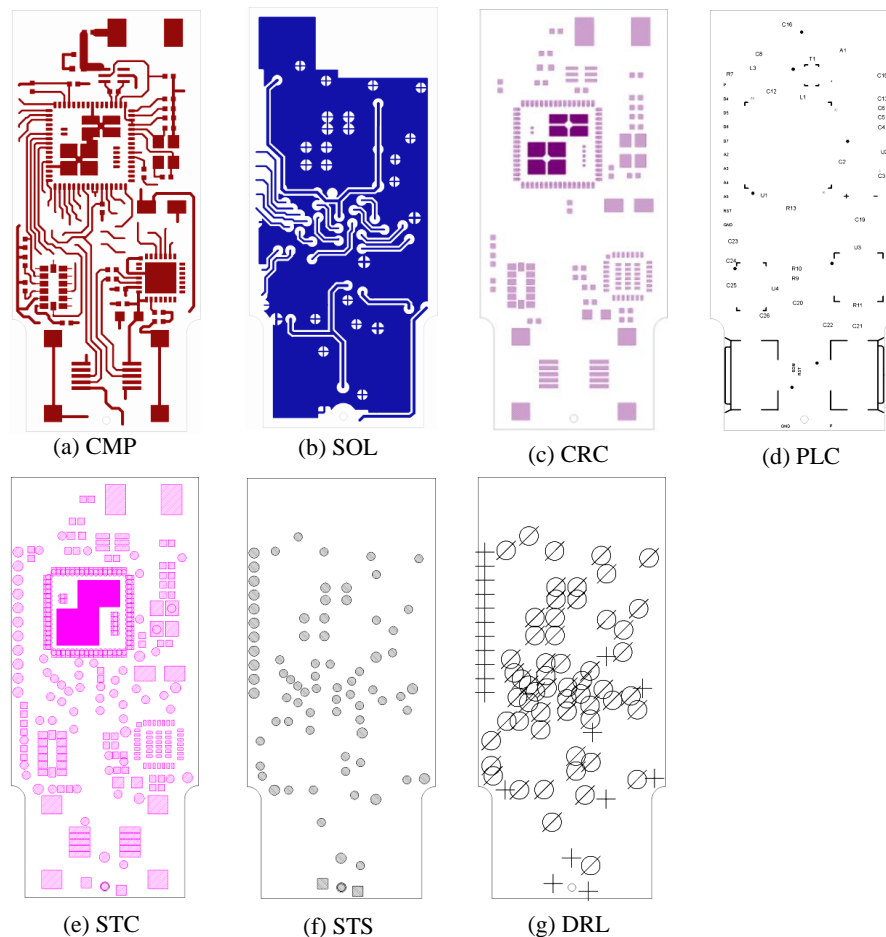


Figure 14: RF-control-board PCB design files (a) CMP (b) SOL (c) CRC (d) PLC (e) STC (f) STS (g) DRL

The PCB is designed on the smallest possible area. The space restrictions are also from the selected casing. The final dimensions of the RF-control-board design are 45x20 mm.

The shape of the LED-switch-board is circular with a diameter of 26 mm from the casing. The thickness of this board is kept at 3 mm so that the connectors J1 and J2 mounted on the bottom layer SOL cannot extrude towards the top layer CMP

where the LEDs and switches are mounted. The connectors J2 and J2 can be seen in the PLC layer. The design files for the LED-switch-board are shown in Figure 15.

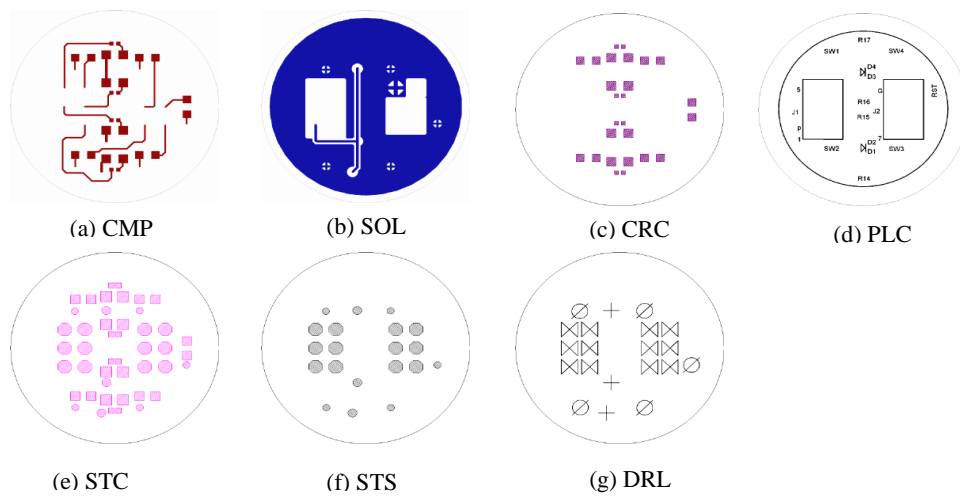


Figure 15: LED-switch-board PCB design files (a) CMP (b) SOL (c) CRC (d) PLC (e) STC (f) STS (g) DRL

### 3.3. PCB Manufacturing and Assembly

The required components and the board dimensions for the two boards are shown in Figure 16. The component layer shows the SMD part requirement. The part list of the ZTID is given in Table 12.

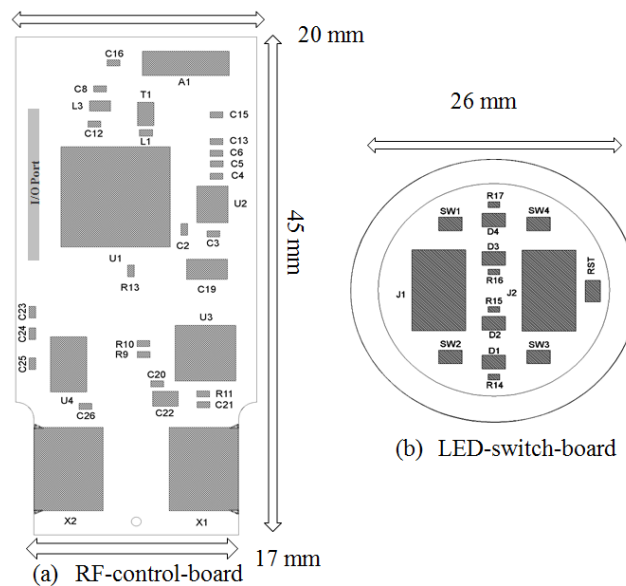


Figure 16: SMD part requirements and board dimensions (a) RF-control-board (b) LED-switch-board

Table 12: Part list for ZTID

| Index                               | Qty | Part                         | Description                              | Unit price (\$) | Price per Qty |
|-------------------------------------|-----|------------------------------|--|-----------------|---------------|
| 1                                   | 2   | C3,C4                        | CAP CER 8PF                              | 0.021           | 0.042         |
| 2                                   | 8   | C2,C5,C6,C12,C13,C20,C21,C26 | CAP CER 0.1UF                            | 0.015           | 0.12          |
| 3                                   | 1   | C8                           | CAP CER 0.5PF                            | 0.017           | 0.017         |
| 4                                   | 2   | C15,C16                      | CAP CER 10PF                             | 0.032           | 0.064         |
| 5                                   | 1   | C19                          | CAP TANTALUM                             | 0.155           | 0.155         |
| 6                                   | 1   | R7                           | RES 470K OHM                             | 0.026           | 0.026         |
| 7                                   | 1   | L1                           | INDUCTOR 4.7NH                           | 0.059           | 0.059         |
| 8                                   | 1   | L3                           | INDUCTOR 8.2NH                           | 0.122           | 0.122         |
| 9                                   | 1   | T1                           | TRANSFORMER BALUN 50 OHM                 | 0.287           | 0.287         |
| 10                                  | 1   | A1                           | ANTENNA CHIP                             | 1.058           | 1.058         |
| 11                                  | 1   | U1                           | Freescale RF Transceiver                 | 4.72            | 4.72          |
| 12                                  | 1   | U2                           | CRYSTAL 16 MHZ                           | 0.685           | 0.685         |
| 13                                  | 1   | R13                          | RES 47K                                  | 0.015           | 0.015         |
| 14                                  | 1   | C22                          | CAP CER 1.0UF                            | 0.131           | 0.131         |
| 15                                  | 2   | R9,R10                       | RES 27 OHM                               | 0.013           | 0.026         |
| 16                                  | 1   | R11                          | RES 4.7K OHM                             | 0.018           | 0.018         |
| 17                                  | 1   | U3                           | IC USB-TO-UART BRIDGE                    | 3.03            | 3.03          |
| 18                                  | 2   | X1, X2                       | CONN MICRO USB B                         | 1.259           | 2.518         |
| 19                                  | 1   | U4                           | 1.5 XYZ, LGA 14                          | 2.11            | 2.11          |
| 20                                  | 3   | C23,C24,C25                  | CAP CER 3300PF                           | 0.05            | 0.15          |
| <b>RF-control-board Total:</b>      |     |                              |  | <b>13.823</b>   | <b>15.353</b> |
| 1                                   | 4   | R14,R15,R16,R17              | RES 220 OHM                              | 0.026           | 0.104         |
| 2                                   | 4   | D1,D2,D3,D4                  | LED CHIPLD RED                           | 0.073           | 0.292         |
| 3                                   | 2   | J1, J2                       | 2X3 MALE PIN STRIP HEADER                | 0.39            | 0.78          |
| <b>LED-switch-board Total:</b>      |     |                              |  | <b>0.489</b>    | <b>1.176</b>  |
| 2                                   | 2   | CL1                          | 6 Conductor 28 AWG Ribbon Cable          | 0.25            | 0.5           |
| 3                                   | 2   | J4                           | 28 AWG Ribbon cable IDC female socket    | 0.5             | 1             |
| <b>Cables and connectors Total:</b> |     |                              |  | <b>0.75</b>     | <b>1.5</b>    |
| 1                                   | 1   | CAS-1                        | Camelion TRAV Lite Pocket LED Flashlight | <b>4.81</b>     | <b>4.81</b>   |
| <b>Total</b>                        |     |                              |  | <b>22.893</b>   |               |

The PCB manufacturing requirements for the two boards are given in Table 13. This table shows the requirements for the number of layers, PCB material, PCB layer-to-layer thickness, deposited copper thickness, type of plating, silk screen color, solder mask type, solder mask color, solder mask thickness and final PCB testing requirements.

Table 13: PCB manufacturing requirements for ZTID

|                                |   |
|--------------------------------|---|
| <b>RF-control-board</b>        |   |
| No of PCB required             | 50  |
| PCB dimensions (X,Y) mm        | (20,45)   |
| Part assembly                  | 10  |
| FR4 Core thickness             | 760 um  |
| Final PCB thickness            | 0.83 mm+/-10% (Cu/Cu excluding solder mask)             |
| <b>LED-switch-board</b>        |   |
| No of PCB required             | 50  |
| PCB dimensions (diameter) mm   | 26  |
| Part assembly                  | Not required  |
| FR4 Core thickness             | 2.7801 mm   |
| Final PCB thickness            | 3 mm ( including solder mask)                           |
| <b>PCB requirements</b>        |   |
| Base laminate material         | FR4   |
| No of PCB layers               | 2 Layer   |
| Finished copper foil thickness | 30-35 um  |
| Plating                        | All pad plating must be hot air leveling (HAL)          |
| Solder mask type               | Liquid Film Electra EMP 110 or Equivalent               |
| Solder mask thickness          | 10-30 um  |
| Solder mask color              | Green   |
| Silk screen color              | White   |
| Silk screen requirement        | Silk screen must not extend in to any plated thru holes |
| Electrical PCB testing         | All PCBs must be 100 % tested for opens and shorts      |

The PCB design was outsourced for manufacturing and assembly from a contract PCB manufacturer in China. The pricing for PCB manufacturing for an RF-control-board and the LED-switch-board is given in Table 14.

Table 14: Manufacturing and assembly pricing

| Type                              | Quantity | Lead time | Unit price          | Total price |
|-----------------------------------|----------|-----------|---------------------|-------------|
| PCB RF-control-board              | 50 pcs   | 5 days    | \$4.653<br>USD/pcs  | \$232.65    |
| PCB LED-switch-board              | 50 pcs   | 5 days    | \$4.0326<br>USD/pcs | \$201.63    |
| Stencil For RF-control-board      | 1 pcs    | 5 days    | \$248.16<br>USD/pcs | \$248.16    |
| PCB assembly for RF-control-board | 10 pcs   | 7 days    | \$20.246<br>USD/pcs | \$202.46    |
| PCB assembly freight              |          |           |                     | \$56.16     |
|                                   |          |           | Total:              | \$941.06    |

50 RF-control-boards are manufactured in China out of which 10 boards are assembled with parts. The top side of an RF-control-board with parts assembled is shown in Figure 17 (a) and the bottom side is shown in Figure 17 (b).

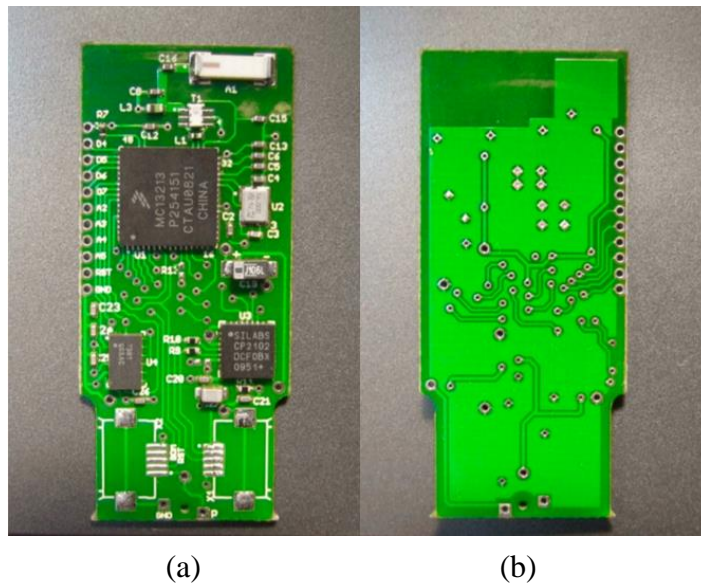
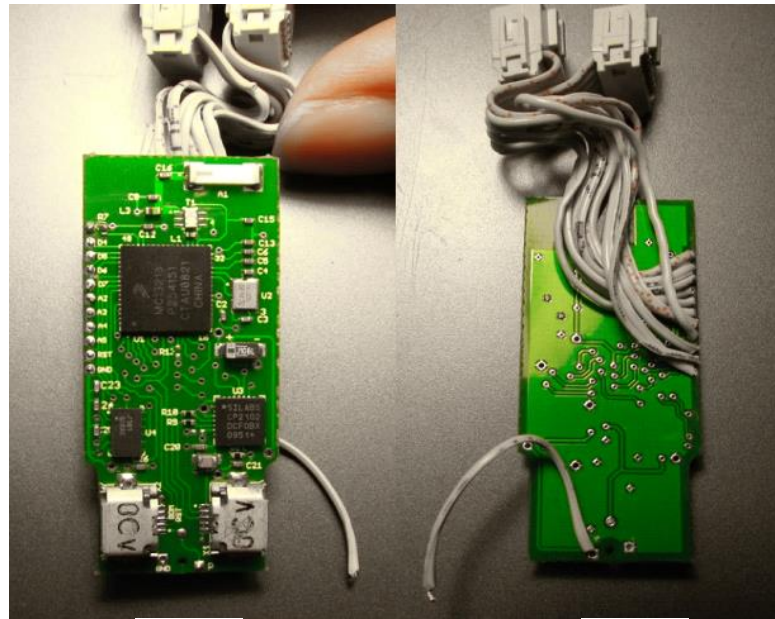


Figure 17: RF-control-board (a) top side (b) bottom side

The final assembly involves mounting the two micro-USB connectors as shown in Figure 18 (a). It also involves mounting the data cable and power cable as shown in Figure 18 (b).

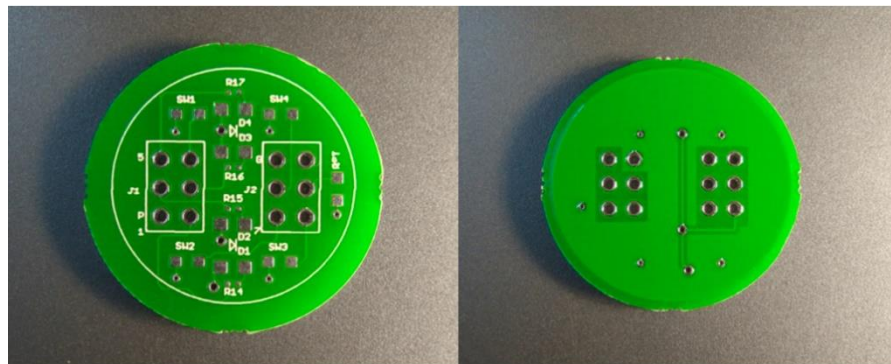


(a)

(b)

Figure 18: Fully assembled RF-control-board (a) top side (b) bottom side

50 LED-switch-boards are manufactured in China. The top side of the LED-switch-board is shown in Figure 19 (a) and the bottom side is shown in Figure 19 (b).



(a)

(b)

Figure 19: LED-switch-board (a) top side (b) bottom side

The final assembly involves mounting the surface-mount LEDs and resistors on the LED-switch-board as shown in Figure 20 (a). It also involves mounting the connectors J1 and J2 as shown in Figure 20 (b).

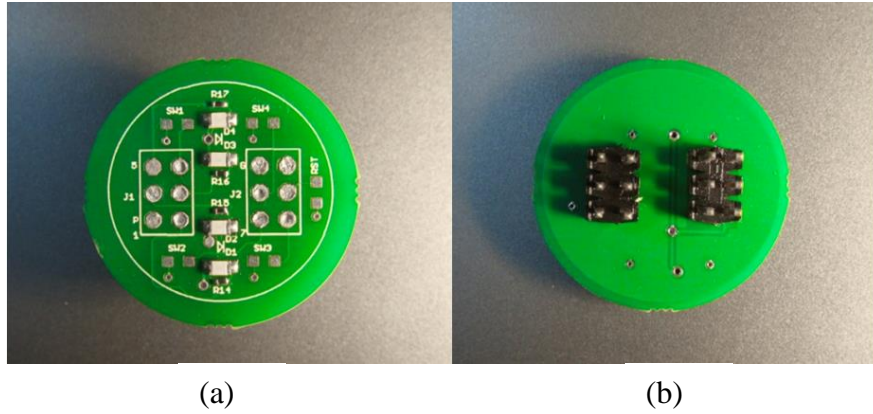


Figure 20: Fully assembled LED-switch-board (a) top side (b) bottom side

### 3.4. Final Device Assembly

The ZTID RF-control-board and LED-switch-board connected together and powered with batteries are shown in Figure 21. The RF-control-board is specially designed to have the smallest possible form factor (20x45 mm). This can be observed relative to the user hand holding the ZTID boards in Figure 21. The two ZTID boards were then finally encased in the selected casing [32]. The selected casing was modified to encase the RF-control-board and LED-switch-board. The casing modification that was done for the ZTID is described in Appendix C.

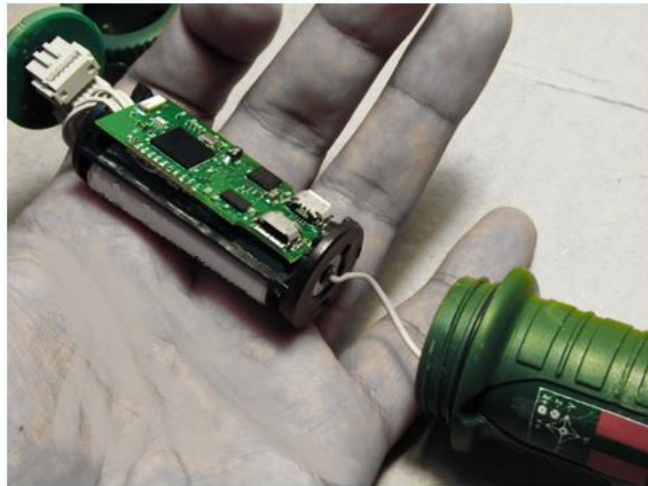


Figure 21: Final assembly of ZTID

## Chapter 4. Software Design

The primary purpose of the embedded software on the ZTID is to receive the 3D accelerometer data from the onboard accelerometer in real-time and to determine and transmit the particular gesture (e.g., tapping) being done at that time. A review of related work on gesture recognition (Table 8) shows that such gesture recognition systems are typically based on motion sensing, filtering, feature extraction/segmentation, and classification. Based on previous work, Figure 22 shows the high level software requirements for ZTID gesture recognition. As Figure 22 shows, acceleration data is filtered first using a moving average filter, followed by K-mean segmentation. The final step consists of classifying the timed sequence of segmented data using HMMs into one of various types of behaviors including tapping, rolling and rubbing.

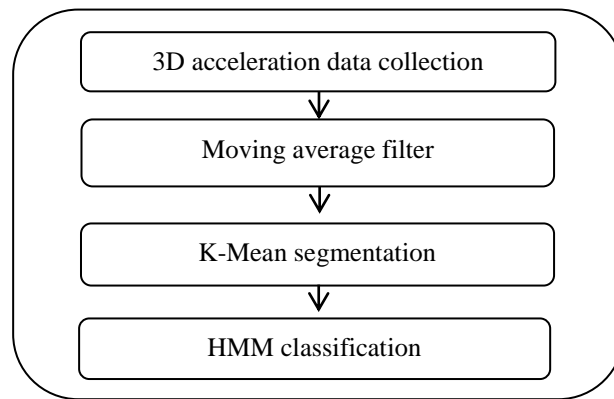


Figure 22: Steps for gesture recognition

The steps for ZTID gesture recognition are described in detail in the next section.

### 4.1. Data Collection

Eight surface gestures were initially defined for the ZTID that are described in Table 15. These surface gestures are shown in Appendix D. As Table 15 shows, the device was designed to distinguish between one type of rolling, two types of rubbing, and five types of tapping behaviors. Device stickers were designed to show the various axes of the accelerometer on the ZTID. The accelerometer axes relative to the device body are shown in Figure 23.

Table 15: ZTID surface gestures description

| Gesture name | Description                             |
|--------------|---|
| Roll-X       | Rolling device across x-axis            |
| Rub-4-X      | Rubbing device across x-axis four times |
| Rub-4-Y      | Rubbing device across y-axis four times |
| Tap-1        | Tapping device one time                 |
| Tap-2        | Tapping device two time                 |
| Tap-3        | Tapping device three time               |
| Tap-4        | Tapping device four time                |
| Tap-5        | Tapping device five time                |



Figure 23: ZTID stickers showing accelerometer axes

For training data collection, an experimental setup was created to wirelessly collect accelerometer data on a ZTID transmitter device on which the surface gestures were being performed. The data was wirelessly received by a receiver ZTID as shown in Figure 24.

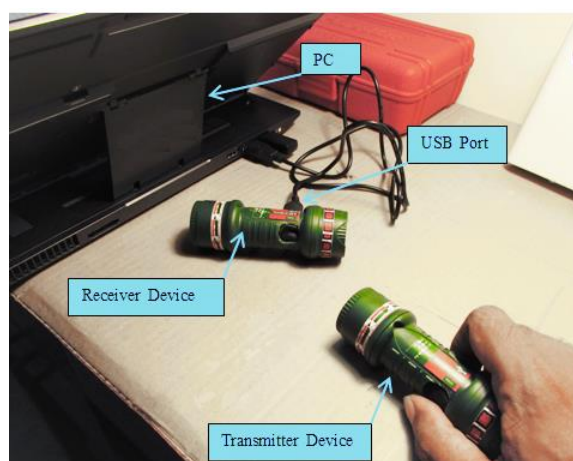


Figure 24: Experimental setup for data collection

The receiver device forwarded the accelerometer data to a personal computer (PC) using the built-in USB port. The receiver device established a ZigBee network and acted as the coordinator node while the transmitter device is an end point sending data to the coordinator node. A total of 240 data points were collected for eight surface gestures; 30 samples per gesture (e.g., Tap-5) were collected. The experimental details for data collection are given in detail in Appendix E. The wireless accelerometer data received by the PC was saved using a serial terminal program for offline training and recognition in the analysis software.

## 4.2. Filtering

A simple moving average filter was used for filtering. This type of filter gives the simple mean of the previous  $M$  data points in time [67] where  $M$  is the order of the filter. The filter output is given by Eq. (1).

$$y(n) = \frac{x(n) + x(n-1) + x(n-2) + \dots + x(n-M+1)}{M} \quad (1)$$

The frequency response of a simple moving average filter can be plotted by using Eq. (2).

$$H[f] = \frac{\sin(\pi f M)}{M \sin(\pi f)} \quad (2)$$

Filter orders of  $M$ : 5, 10 and 20 were considered for the filter design. The frequency response for these three filter orders are plotted in Figure 25. For the defined eight surface gestures the accelerometer data bandwidth is between 0-6 Hz. For filter design, the pass band was selected using a -3dB filter bandwidth. For the lowest filter order  $M = 5$ , the pass band (X: 9 Y: 0.708) is shown in red in Figure 25. The pass band for  $M = 5$  comes out to be 0-6 Hz with the data being sampled at 66.66 Hz. For higher filter orders  $M = 10$  and  $M = 20$  the pass bands comes out to be 0-4 Hz and 0-2 Hz respectively which will attenuate the surface gesture frequencies lying between 0-6 Hz. Consequently, filter order  $M = 5$  was selected to pass surface gesture frequencies between 0-6 Hz.

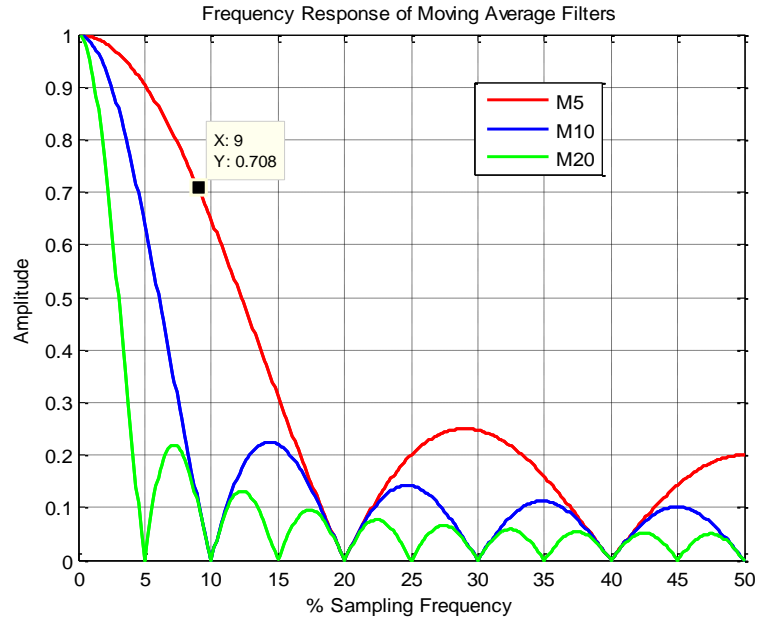


Figure 25: Selecting the order of moving average filter

Figure 26 shows a sample of filtered accelerometer data for the Roll-X gesture. For the eight gestures, the filtered data plots are given in Appendix F and the Matlab code for filtering is given in Appendix I.

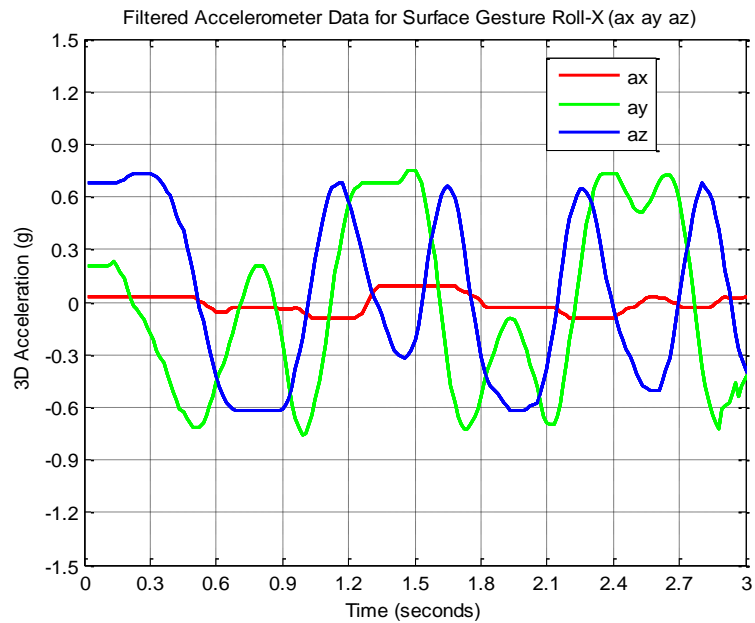


Figure 26: Sample of filtered accelerometer data for gesture Roll-X

### 4.3. Feature Segmentation

A simple K-mean algorithm was used for feature segmentation. The K-mean algorithm reduces the dimensionality of a gesture from a three dimensional space of acceleration to a one dimensional space called the codebook. The K-mean algorithm clusters the data into K 1D cluster centroids and associates each data point that is nearest to each cluster in a loop until convergence is achieved and no data point moves its associated cluster. In this way the dimensionality of the gesture is reduced without losing important information which is a part of the gesture.

For example, [56] used 2D gestures and a codebook size of  $K=8$ . Codebook size ( $K$ ) represents the number of discrete states a 3D acceleration vector can be collapsed into. A codebook size of 14 was suggested by [68] for gestures data scattered in 3D space. They also showed that a codebook size of 18 resulted in over-trained hidden Markov models (HMMs). Since the gestures for the ZTID were mostly scattered in 3D space, it was decided to use three values of codebook size  $K$  (8, 10 and 14) and to select the  $K$  with the best results. Since three  $K$  values were used, the K-mean algorithm created three code books for each gesture data as shown in Appendix F. Figure 27 shows an example of the resulting codebook sequence for  $K=8$  created from 3D accelerometer data collected on a ZTID. The codebook shows the 3D acceleration vector reduced to a one dimension sequence with 8 possible discrete states (1 to 8). The Matlab code implementing feature segmentation is given in Appendix I.

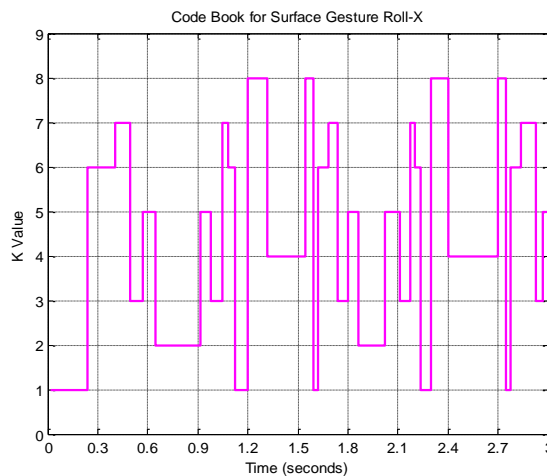


Figure 27: Sample codebook sequences of states for gesture Roll-X using  $K=8$

#### 4.4. Classification

Once the filtered acceleration data has been converted into a 1D timed sequence of codebook states, the classification problem for ZTID needs to decide which of the eight gestures is being currently executed. The classification problem was solved using hidden Markov models (HMMs). An HMM ( $\lambda$ ) is defined by the number of states  $N$  and three sets of probability distributions that are:

1. Initial state distribution  $\pi = \{\pi_i\}$  given by Eq. (3). Where states ( $Q$ ) are  $\{1, 2, \dots, N\}$ .

$$\pi_i = P\{q_i = i\}, 1 \leq i \leq N \quad (3)$$

2. State transition probability distribution  $A = \{a_{ij}\}$  given by Eq. (4). Where the state at time  $t$  is denoted as  $q_t$ .

$$a_{ij} = p\{q_{t+1} = j | q_t = i\}, 1 \leq i \leq N \quad (4)$$

3. Output probability distribution  $B = \{b_j(k)\}$  given by Eq. (5). Where  $v_k$  denotes the  $k^{\text{th}}$  observation symbol in the alphabet, and  $o_t$  is the current parameter vector [69]. More details about HMMs are described in Appendix G.

$$b_j(k) = p\{o_t = v_k | q_t = j\}, 1 \leq j \leq N, 1 \leq k \leq M \quad (5)$$

For each defined surface gesture, a single untrained HMM ( $\lambda$ ) needs to be trained. The trained HMM is denoted by  $(\lambda^*)$ . In the HMM training problem the observation sequence  $O = O_1, O_2, \dots, O_T$  is given as a 1D codebook and the goal is to adjust the model parameters  $\lambda = (A, B, \pi)$  to maximize  $P(O|\lambda)$ . When  $P(O|\lambda)$  is maximized, the model parameters  $A, B$  and  $\pi$  are obtained [70]. This probability is the total likelihood of the observations and can be expressed mathematically as given in Eq. (6).

$$L_{\text{tot}} = p\{O|\lambda^*\} \quad (6)$$

There is no known way to solve this problem analytically [69]. The most common HMM training technique is Baum-Welch. The Baum-Welch algorithm maximizes the probability  $P(O|\lambda)$  such that the model parameters  $A, B$  and  $\pi$  are

obtained. In order to do so, following [56], the Baum-Welch algorithm [71] was used to train one HMM ( $\lambda^*$ ) for each gesture as shown in Figure 28.

It was shown by [56] that both ergodic and left-to-right HMM models gave similar results, and that good results have been obtained in earlier studies by using an ergodic model with five states for a set of gestures. Moreover, the number of states does not have a significant effect on the gesture recognition results in similar applications [60], [61]. Consequently, five state ergodic HMMs were used [72] for ZTID gesture recognition. The training for the ZTID surface gestures was done offline in Matlab. An example of a trained HMM for the surface gesture Roll-X is given in Appendix G.

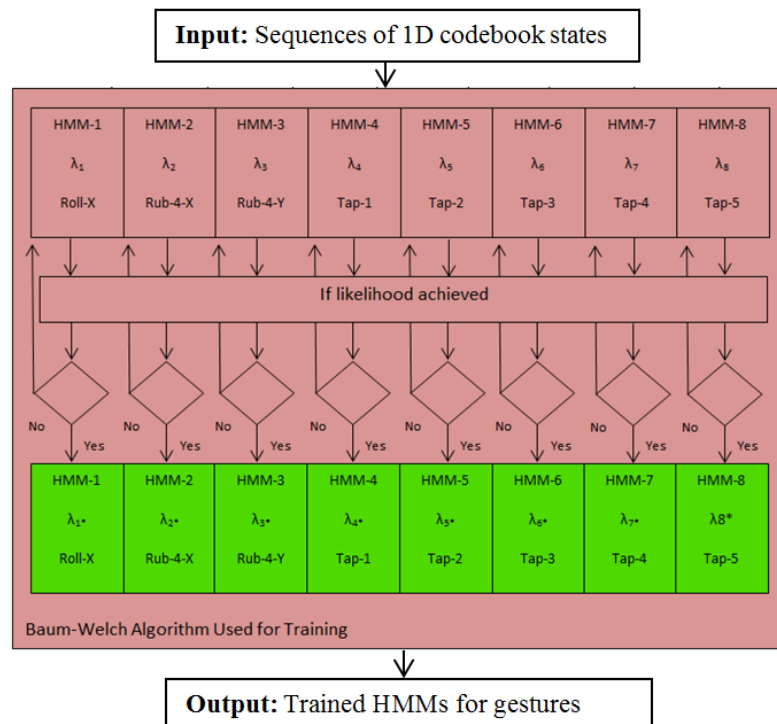


Figure 28: Using Baum-Welch algorithm to train one HMM per gesture

For embedded C code implementation the memory requirement to store eight trained HMMs ( $\lambda^*$ ) is given in Table 16. Each HMM model  $\lambda^* = (A, B, \pi)$  requires 3 matrices that are defined as 32 bit variables. With the implementation choices (5-state models with a 14 vector codebook) 400 bytes of memory are required to store each model. For storing eight surface gestures, a total of 3200 bytes are needed.

Table 16: Memory requirement for HMM storage

| Variables for a single HMM ( $\lambda^*$ ) | Variables size for a single HMM.<br>N is number of states; K is codebook size | Variables size for a single HMM with N = 5; K = 14 | Variable size for a single HMM using 32 bit variable $\times$ 4 (bytes) | Variable size for eight HMMs using 32 bit variable $\times$ 8 (bytes) |
|--|---|--|---|---|
| $\pi$                                      | $N \times 1$  | 5  | 20  | 160   |
| A  | $N \times N$  | 25   | 100   | 800   |
| B  | $N \times K$  | 70   | 280   | 2240  |
| Total                                      | $N + N^2 + N \times K$  | 100  | 400   | 3200  |

For solving the problem of ZTID gesture recognition, a forward algorithm was used. In the forward algorithm the observation sequence  $O = O_1, O_2, \dots, O_T$  (as 1D codebook) and model  $\lambda^* = (A, B, \pi)$  are given and the goal is to compute the probability  $P(O|\lambda^*)$  [70]. The direct calculation has exponential complexity  $N^T$  (N states, T observations).

Forward algorithms make use of an auxiliary variable,  $\alpha_t(i)$  called the forward variable which reduces probability calculation complexity to linear in T i.e.  $N^2T$ . The forward algorithm recursion consists of three steps [71] that are:

1. Initialization of the forward variable from the trained model  $\lambda^*$  and the code book sequence  $O$  given in Eq. (7).

$$\alpha_1(j) = a_0 b_j(o_1), 1 \leq j \leq N \quad (7)$$

2. Recursion to find the forward variable from the trained model  $\lambda^*$  and the code book sequence  $O$  given in Eq. (8).

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t); 1 \leq j \leq N, 1 \leq t \leq T \quad (8)$$

3. Termination to find the probability that the given observation sequence  $O$  is generated from the trained model  $\lambda^*$  given in Eq. (9).

$$p\{O|\lambda^*\} = \alpha_T(q_F) = \sum_{i=1}^N \alpha_T(i) a_{iF} \quad (9)$$

Once each of the HMMs  $\lambda^*$  has been trained for each gesture, the classifier uses the HMM forward-algorithm [72] to calculate the best match for an arbitrary

sequence of codebook states as shown in Figure 29. The software implementation of the forward algorithm in Matlab is given in Appendix I. For software implementation in C, the memory requirement for the embedded C code is given in Table 17.

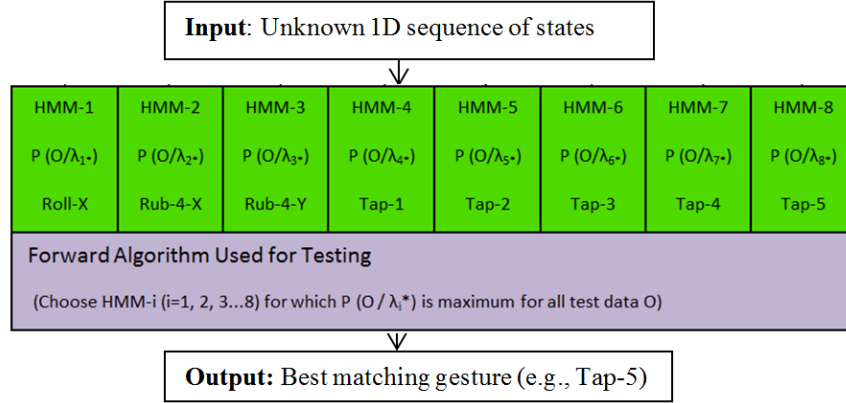


Figure 29: Using forward algorithm to classify a sequence of 1D codebook states in to the best matching gesture

Table 17: Memory requirement for embedded coding

| Code type   | Size (bytes) | Comments               |
|-------------|--------------|------------------------|
| HMM storage | 3200         | From Table 16          |
| Network     | 18088        | Binary file size (S19) |
| K-mean      | 1222         |                        |
| Forward     | 1090         |                        |
| Total       | 23600 (23KB) |                        |

The eight trained HMMs for the ZTID surface gesture require 3200 bytes of memory. The binary file size (S19) for the network software [73] is 18088 bytes. The K-mean and forward algorithm codes [74] are given in Appendix J. The binary file size (S19) for these two codes is 2312 bytes. Thus the total memory requirement for the ZTID embedded software using the trained surface gestures is 23 KB. The ZTID HCS08 microcontroller supports 64 KB of Flash memory which shows that the trained HMMs, K-mean and forward codes can be implemented on ZTID device memory. Table 18 and Table 19 show the time taken by the ZTID CPU for the K-mean and forward algorithm respectively on set of 30 data points. The K-mean algorithm takes 1.653 ms to generate a code-book on 30 data points and the forward algorithm takes 0.09375 ms to recognize the unknown surface gesture.

Table 18: K-Mean algorithm execution time

|  |                                    |              |           |         |  |   |
|--|------------------------------------|--------------|-----------|---------|--|---|
| <p>K-mean algorithm execution time (1.653 ms)<br/>                 Where code book size is K; number of data points are <math>N_p</math>; CPU instruction cycle time is <math>T_s</math> (in <math>\mu s</math>)<br/>                 and no of iterations for K-mean algorithm are <math>N_i</math><br/>                 For <math>K = 14</math>; <math>N_p = 30</math>; <math>T_s = 0.125</math> and <math>N_i = 15</math></p> |                                    |              |           |         |  |   |
| Steps in K-mean algorithm  | Type of operations used for K-mean |              |           |         | Sum of all operations<br>$S =$<br>additions+<br>subtractions+<br>divisions+<br>squares | K-mean algorithm CPU time in ( $\mu s$ )<br>$= S \times T_s \times N_i$<br>$= 1653$ |
|  | Additions                          | Subtractions | Divisions | Squares |  |   |
|  | Total number                       |              |           |         |  |   |
| Distance to centroids  | $K - 1$                            | $K$          |           | $K$     | $3 \times K - 1$   | 77  |
| Assign centroids   |                                    |              |           |         | $N_p \times K$   | 788   |
| New centroids  | $K \times (N_p - 1)$               |              | $K$       |         | $N_p \times K$   | 788   |

Table 19: Forward algorithm execution time

|   |
|---|
| <p>Forward Algorithm execution time (0.09375 ms)<br/>                 Where N is the number of states; T is the number of observations and <math>T_s</math> is CPU instruction cycle time (in <math>\mu s</math>)<br/>                 For <math>N = 5</math>, <math>T = 30</math> and <math>T_s = 0.125</math></p> |
| <p>Forward algorithm CPU time in (<math>\mu s</math>) is <math>N^2 \times T \times T_s = 93.75</math></p>   |

To summarize, the ZTID filters the 3D accelerometer data which is collapsed into 1D sequences using the K-mean algorithm which is then fed into multiple HMMs where each HMM corresponds to a gesture (e.g., Tap-3) and the HMM with the best fit is selected as the gesture being performed at that particular point in time.

## Chapter 5. Evaluation

This chapter is about ZTID evaluation. In this chapter the hardware designs described in Chapter 3 are evaluated. In addition, this chapter also presents the evaluation of HMM design presented in Chapter 4 by using classifier performance evaluation methods.

### 5.1. Hardware Evaluation

After the ZTID devices were assembled, each device was programmed with the SDK software through the USB interface as shown in Figure 30. A device pair was first programmed with a wireless range program [64] to check the wireless link quality and range.



Figure 30: Programming the ZTID

It was found that the device wireless link quality was satisfactory and that the devices could be used in star or mesh networking as desired. The range test can be seen on a YouTube video [75]. The video shows the radio frequency (RF) link quality between two ZTIDs as a function of distance between the two devices. If all the LEDs on both the devices are blinking, it means that that RF link quality is at its maximum. If no LED is blinking on both the devices, it means there is no RF link between the two ZTIDs. After the successful range testing for a ZTID pair, the accelerometer operation [73] was loaded in to another ZTID pair as seen on another YouTube video [76]. This video shows the real-time accelerometer data from a transmitter ZTID being received wirelessly by a receiver ZTID. The receiver ZTID then sends this received accelerometer data to a PC in real-time. This real-time data is seen on GUI in

the video. These hardware tests show that the ZTID supports all the envisioned hardware and networking operations.

## 5.2. Software Evaluation

To evaluate the gesture-based software, codebook sets for various values of  $K$  were divided into training and test sets using random sub-sampling in Matlab. Random sub-sampling or Monte Carlo cross-validation is based on randomly splitting the data into training and testing subsets, whereby the size of the subsets is defined by the user. The random partitioning of the data can be repeated arbitrarily often. In contrast to a full cross-validation procedure, random sub-sampling has been shown to be asymptotically consistent resulting in more pessimistic predictions of the test data compared with cross-validation. The predictions based on this method, therefore, provided a more realistic estimation of accuracy and performance [77].

The trained HMMs, one for each gesture, were tested by using the HMM forward-algorithm [72] using the random sub-sampling approach. Figure 31 shows how accuracy increases with the selection of larger sub-sets of training vectors. For example, Figure 31 shows that randomly selecting 12 test vectors from each gesture for training results in an accuracy of 99.7% for a codebook with  $K=14$ . As Table 20 shows, the codebook with  $K=14$  gave the highest recognition accuracy of 99.7% and the recognition accuracy is improved by increasing the codebook size.

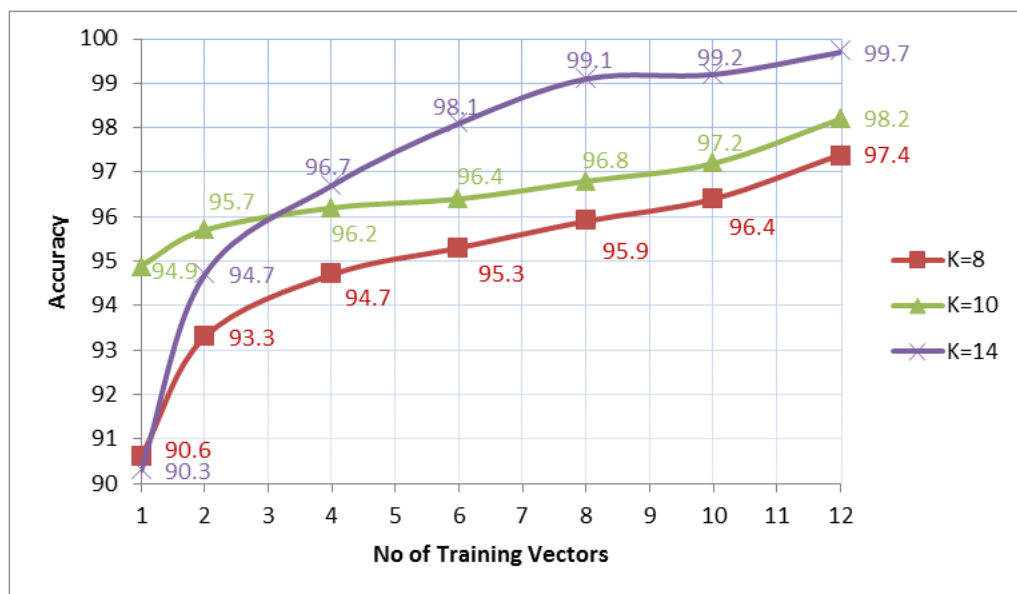


Figure 31: Accuracy verses the size of training sub-set for various code books

Table 20: Recognition accuracy and codebook size

| Codebook size (K) | Recognition accuracy (%) |
|-------------------|--------------------------|
| 8                 | 97.4                     |
| 10                | 98.2                     |
| 14                | 99.7                     |

The trained HMM classifiers for each codebook size K, 8, 10 and 14, are evaluated using receiver operating characteristics (ROC). The ROC shows the trained classifier true positive rate (TPR) verses the false positive rate (FPR). The TPR and FPR are defined as:

$$TPR = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (10)$$

$$FPR = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} \quad (11)$$

For ZTID surface gesture recognition, eight surface gestures (Roll-X, Rub-4-X, Rub-4-Y, Tap-1, Tap-2, Tap-3, Tap-4, and Tap-5) are defined. Figure 32, Figure 33 and Figure 34 show the ROC for these surface gestures using codebook sizes of K=8, 10 and 14 respectively. The ROC in Figure 32 is plotted for the HMM classifier that has given a recognition accuracy of 97.4 % using K=8. Figure 33 and Figure 34 show the improvement in recognition accuracy by increasing the code book size. The best HMM classifier is found from ROC for K=14 as shown in Figure 34. The TPR vs FPR of eight surface gestures for this best classifier resemble a step function that has achieved the highest recognition accuracy of 99.7%. The best HMM classifier (of Figure 34) is robust in recognizing surface gestures and thus provides an ideal solution for ZTID software because it is close to an ideal classifier. The Matlab code for ROC is given in Appendix I.

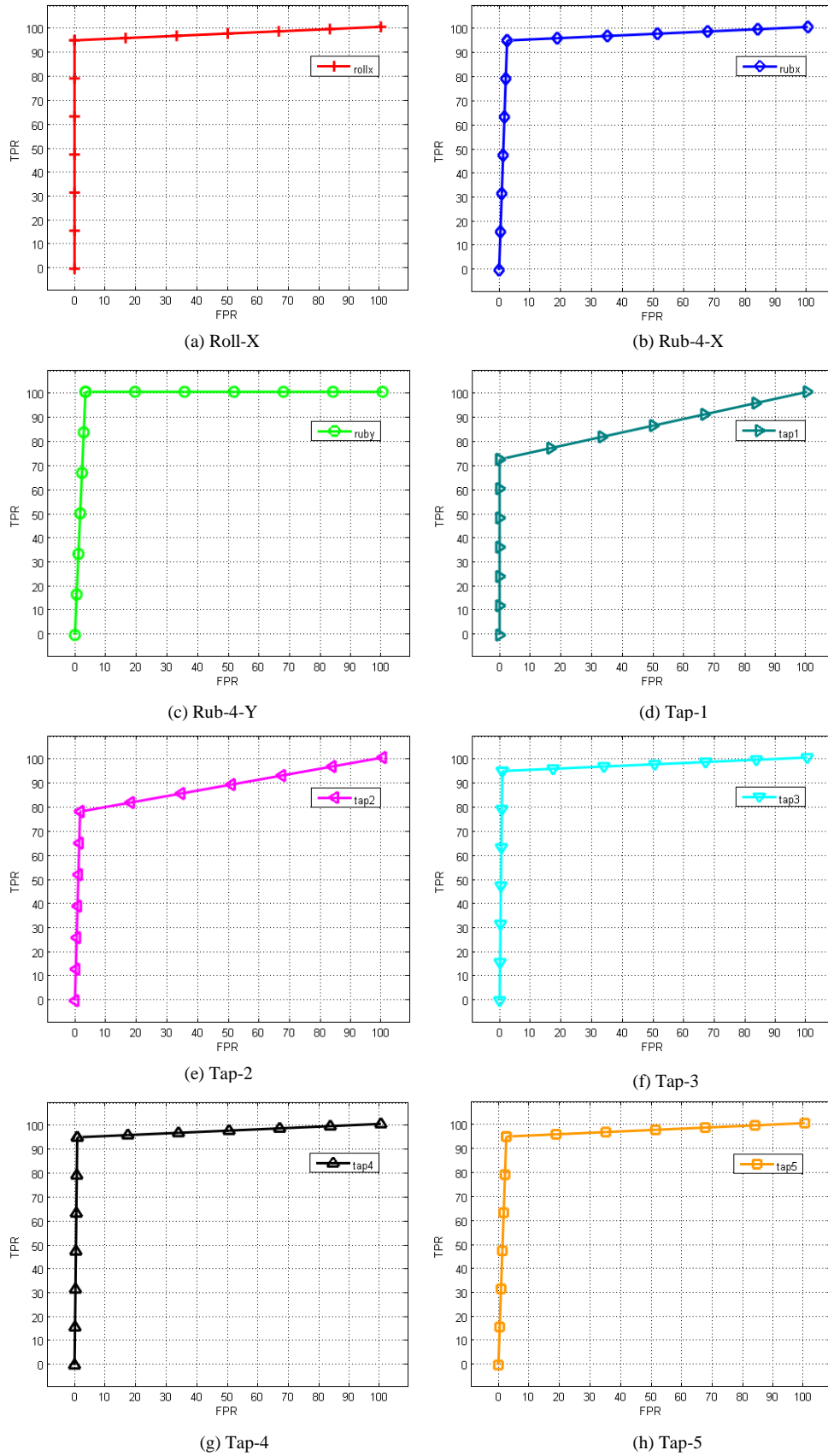
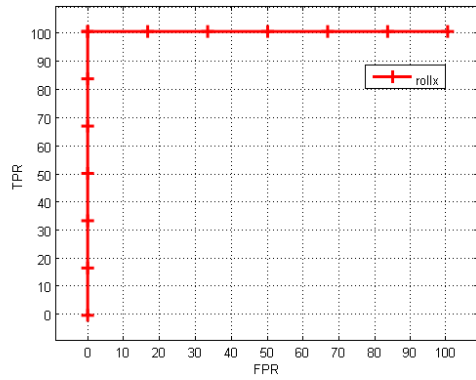
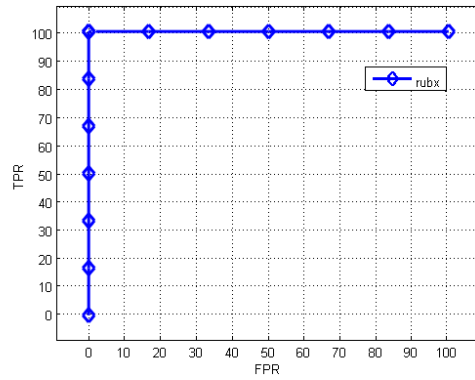


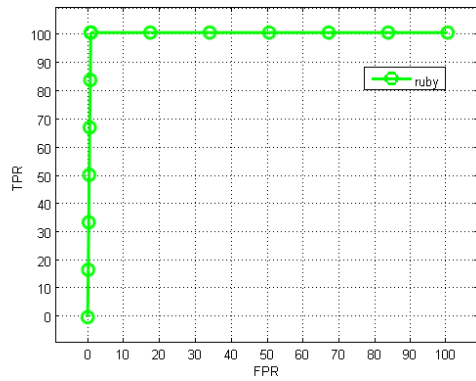
Figure 32: ROC for eight surface gestures using codebook size  $K=8$  (a) Roll-X (b) Rub-4-X (c) Rub-4-Y (d) Tap-1 (e) Tap-2 (f) Tap-3 (g) Tap-4 (h) Tap-5



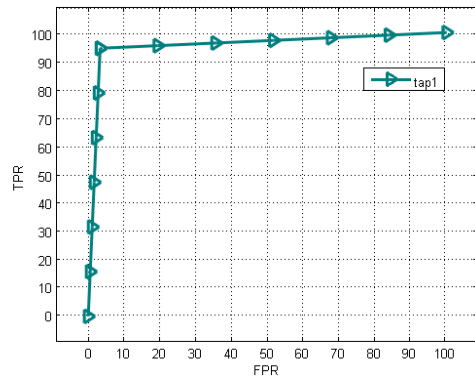
(a) Roll-X



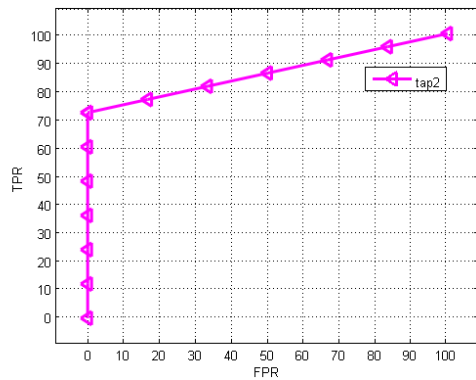
(b) Rub-4-X



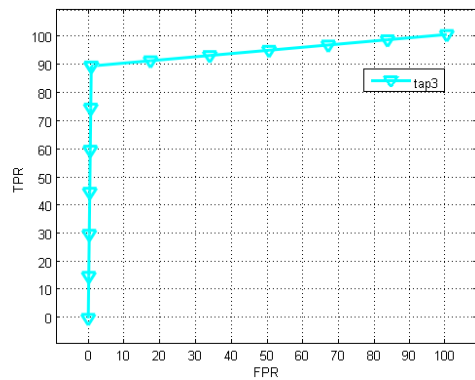
(c) Rub-4-Y



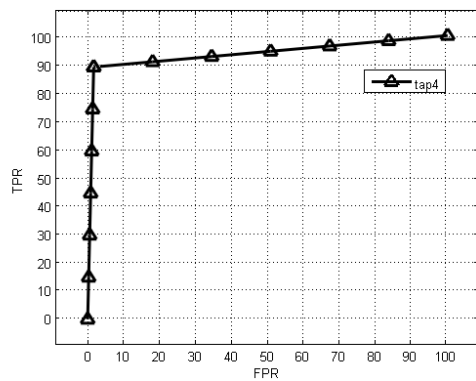
(d) Tap-1



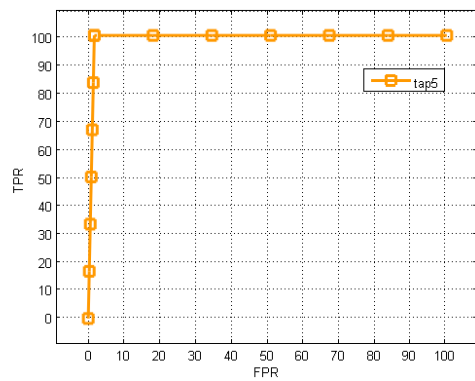
(e) Tap-2



(f) Tap-3



(g) Tap-4



(h) Tap-5

Figure 33: ROC for eight surface gestures using codebook size  $K=10$  (a) Roll-X (b) Rub-4-X (c) Rub-4-Y (d) Tap-1 (e) Tap-2 (f) Tap-3 (g) Tap-4 (h) Tap-5

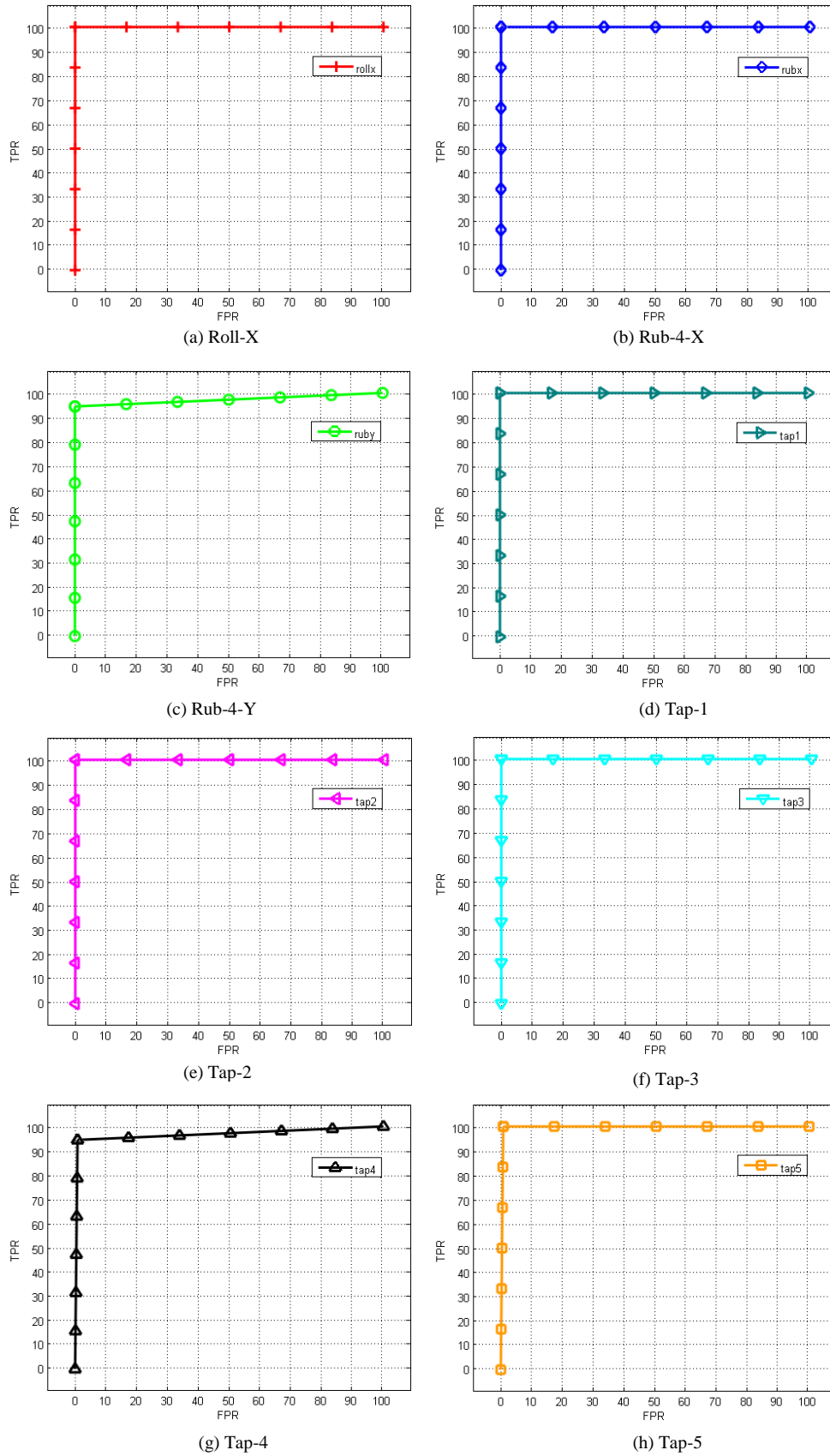


Figure 34: ROC for eight surface gestures using codebook size  $K=14$  (a) Roll-X (b) Rub-4-X (c) Rub-4-Y (d) Tap-1 (e) Tap-2 (f) Tap-3 (g) Tap-4 (h) Tap-5

## Chapter 6. Applications

ZTID features a tangible input interface that is specially designed to be surface-gesture-based for developing applications that may need sensing, control, monitoring and ZigBee wireless network capabilities. The ZTID is capable of tangible input by sensing from an onboard accelerometer, can control devices (home appliances, robots, gaming applications etc. connected in a ZigBee network) by recognizing unknown surface gestures using surface gesture models that are trained (offline in Matlab) and implemented on ZTID hardware (as trained HMMs, K-mean and forward codes), and can monitor (using SMD LEDs) and control actuators (for example a DC motor) from a specially-designed I/O interface that is integrated with the device's 8-bit microcontroller. ZTIDs can also create and configure ZigBee networks from an on-board ZigBee controller by using open source ZigBee software, can talk to external devices using the device's built-in USB and can be easily reprogrammed for developing new applications for network design and surface gesture recognition from a USB port. The applications for ZTID cannot be limited because the device is capable of network design using ZigBee by using open source network software that enables ZTID to be used as a ZigBee evaluation device for creating ZigBee applications. The main features of ZTID are shown in Figure 35 and described in Table 21. Figure 35 (a) shows the device being hand-graspable with hardware interface featuring a USB port, programming port and a power switch. Figure 35 (b) shows the device's I/O interface with four surface-mount LEDs, four pad switches and a device reset switch giving a total of 8 programmable I/Os.

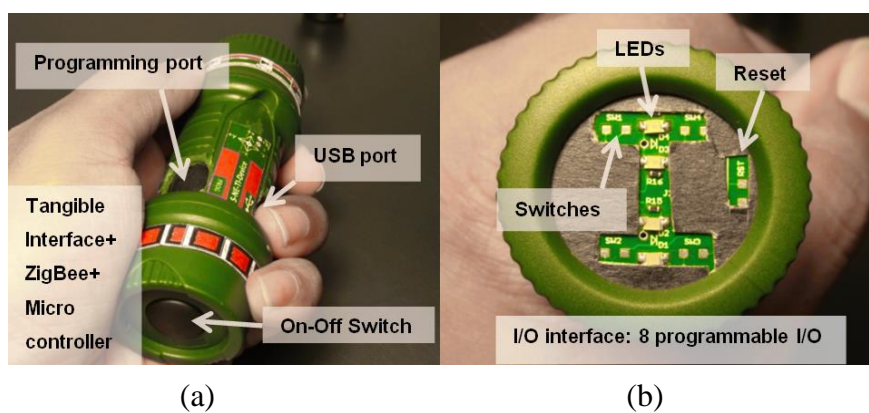


Figure 35: ZTID features (a) Hardware ports (b) I/O interfaces

Table 21 shows that ZTID can be used as pocket-sized ZigBee device for low power applications with input voltages as low as 2V and a sleep mode current of only 3uA. The ZTID ZigBee controller supports programmable RF output power and offers 16 selectable communication channels. The accelerometer sensitivity is also software programmable for two possible values: 1.5 or 6 g.

Table 21: Detailed features of ZTID

| <b>Main features</b>                 | <b>Details</b>  |
|--------------------------------------|---|
| Tangible interface features          | Round casing with capacitive micro machined accelerometer for surface input gestures<br>Selectable sensitivity: 1.5g / 6 g<br>High sensitivity 800 mV/g     |
| ZigBee and micro-controller features | 2.4 GHz full functional ZigBee device<br>16 selectable channels<br>Programmable output power<br>8 bit micro-controller, 60 KB Flash, 4KB RAM                |
| Communication interfaces             | USB interface<br>Programming interface<br>Number of communication ports: 1  |
| Button based interface/control       | Four LED's<br>Four switches<br>Device reset switch<br>Total programmable inputs/outputs: 8  |
| Casing                               | Pocket sized, light weight, round, ultra-shock resistant casing   |
| Power/ temperature range             | Input voltage range: 2V-3.4V<br>Low current consumption; sleep mode 3 uA<br>Two power options; 2 AAA batteries, USB power<br>Temperature range: -40 to 85 C |

ZTID integrates the ZigBee network controller, micro controller, USB controller, I/O interface and accelerometer, all specially designed for surface gesture recognition with a casing that is round, light-weight, hand graspable and pocket-sized.

ZTID surface gestures of type roll, rub and tap can be used to control home appliances in a smart home by integrating them with the existing home automation system presented by [12]. They provided solutions to connect all home appliances (with and without IR receivers) using ZigBee protocol in a home network system. Their hardware consists of an integrated remote controller (IRC), ZigBee power

adapter (ZPA) and ZigBee to IR converter (ZB2Ir). The IRC with infrared (IR) receiver, ZigBee module, microprocessor and GUI serves as a central coordinator to initiate and control the network from the GUI by registering home appliances tied to ZPA and ZB2Ir. IRC is also used to learn IR codes from home appliances' remote controls. The ZPA has a ZigBee module and solid state relays (SSR) for controlling home appliances that are without an IR receiver. The ZB2Ir has ZigBee to IR converter that converts the ZigBee control code from IRC to IR code for controlling the home appliances with the IR receiver. The work done by [12] shows that any home appliance (with and without an IR receiver) can be controlled using a ZigBee network by using a central coordinator (IRC) and end devices (ZPA and ZB2Ir). The ZTID can be integrated in to the system designed by [12] as a gesture based remote controller that will join the ZigBee network initiated by IRC for controlling home appliances using ZPA. The trained HMMs can be implemented on the IRC and surface gestures will be performed on the ZTID. The IRC will recognize the surface gestures and will control the home appliances (without the IR receiver) using ZPA. To control home appliances (with an IR receiver) using ZB2Ir, a mapping will be required for the home appliance remote control function to the ZTID's new defined surface gestures. Once that is done, the ZTID can be used as a surface-gesture-based remote controller to control both non-IR and IR based home appliances. The ZTID surface gesture will provide a more natural way to control home appliances by turning lights on/off, dimming lights, rolling curtains up and down, etc.

Moreover, ZTID surface gestures can also be used to train HMMs for controlling the direction and speed of a mobile robot. To implement this type of application, a single HMM needs to be trained for each motor command like "move forward or backward," "turn left or right" or "increase or decrease the robot speed." As an example, to implement the robot forward command, a new surface gesture Rub-Y-F can be defined in which the ZTID is rubbed in the +y direction as shown by the device axis in Figure 36. Similarly to the "implement robot backward" command, a new surface gesture, Rub-Y-B can be defined in the -y direction. To increase or decrease the speed of the robot, two surface gestures can be defined as Roll-F and Roll-B in which the device is rolled in the forward and backward directions, respectively. These two surface gestures, when used in a continuous forward or backward role, can serve to increase or decrease pulse width modulation for speed

control. Similarly, two new surface gestures can be used as Turn-L and Turn-R in which the device is turned left or right on a surface. These two surface gestures in a continuous left or right turn can function for increasing or decreasing pulse width modulation for direction control. In this way, the direction and speed of a mobile robot can be controlled by defining new surface gestures. However it will be difficult to control a robot for a coordinated action that requires movement on a path or line in space. More surface gestures need to be defined for handling such cases.



Figure 36: ZTID stickers showing accelerometer axes

To summarize, we can conclude that the applications for ZTIDs are not limited because these device are capable of network design using ZigBee and 8-bit microcontrollers. Since the form factor of the ZTID is small, these devices can be used as portable ZigBee evaluation platforms. The ZTID can be used for students and researchers for implementing engineering design projects that require many devices connected in a wireless personal area network with low data rate, low power and a small micro controller.

ZTID can be used to develop remote control applications that are based on surface gestures like roll, rub and tap or their derivatives (roll half time, one time, two times, horizontal roll, vertical roll, roll on an angled surface, rub in a particular direction, rub one time, two times, tap one time, two times, four times, etc.) for which the tangible interface of these devices is specially designed.

## Chapter 7. Conclusion

This thesis introduces a new class of tangible user interfaces that is based on surface behaviors like roll, tap and rub. The new class of surface behaviors originates from the design and implementation of “ZigBee-based Tangible Input Devices” or ZTIDs. A ZTID is a ZigBee-based device that offers a tangible interface from surface gestures. The devices are designed such that the surface gestures can be both computationally and perceptually coupled to the underlying surface gesture model. To accomplish this, hardware and software was designed for the ZTID. The hardware design includes the device design for wireless networks, a motion sensor, USB controller, device casing, power options, and a microcontroller-based interface and programmability. The software design includes the training of HMM classifiers using the Baum-Welch algorithm for the defined surface gesture type roll, rub and tap.

The ZTIDs were tested and evaluated for both hardware and software. The purpose of hardware testing was to check the functionality of each designed hardware component. The hardware testing shows that each hardware component performed well for device network and range, motion sensor and USB controller functionality and device re-programmability. The software evaluation was done to find the recognition accuracy of the trained classifier. The software evaluation was based on a forward algorithm that was used with the random sub-sampling approach. It was found from software evaluation based on trained HMM classifiers that the codebook size value of 14 (among 8, 10 and 14) has given the best classifier among the trained HMM classifiers. The best HMM classifier has achieved a recognition accuracy of 99.7 percent in recognizing unknown surface gestures.

However, since ZTID training and recognition for HMMs was done using Matlab, it is recommended to implement the recognition software in the device micro controller. For real time recognition, a single surface gesture can be recognized in approximately 1.75 (ms) using the device’s 8 bit micro controller. To store eight trained surface gestures, network software and recognition software (K-mean and forward codes) it will take 23KB of the ZTID’s micro controller memory. Since the ZTID microcontroller supports 64KB of memory, it can be concluded that ZTID recognition software can be implemented for real time surface gesture recognition. Since ZTID has a limited device memory (64KB), there is a limit on the maximum

number of surface gestures that can be stored on the device's 8 bit micro controller. For the case in which 20 KB of memory is taken by network and recognition software and 3 KB for storing 8 surface gestures, 41KB of device memory is left which can be used to store approximately 100 more surface gestures (as a 32 bit variable) or 200 more surface gestures (as a 16 bit variable). In case the network software uses more ZTID memory, the maximum number of surface gestures can be reduced to 50 or less depending on the network software size. In that particular case, to solve the memory limitation problem, it is recommended to use ZTIDs as end devices connected in a network (preferably a ZigBee network). Any ZigBee coordinator that has sufficient memory can be used to recognize the surface behaviors that are received from the ZTID in a ZigBee network.

Moreover since the surface gestures were executed on a horizontal surface, it is recommended to define surface gestures roll, tap and rub for surfaces that are vertical or rough. This will give rise to more surface gestures that can be used to control the user environment in more numbers of ways. For example, the gesture roll on a horizontal surface is different from a roll on a vertical surface because the accelerometer sensor data will be different for the two cases. The number of times a device is rolled will also give a different set of data. So in this way new gesture configurations can be defined. But HMM training will be required for every new gesture type. A limitation of HMMs is that they require a lot of training data. It was found from ZTID software evaluation that 12 training vectors is an optimum number to train an HMM model for a gesture. For ZTID, it becomes very easy to get sensor data for training as the devices are specially designed for this purpose. So it is preferred to use HMMs for ZTID. However, it is recommended to implement classification methods on the ZTID that will require fewer training vectors than HMMs. It is also recommended to explore fast DTW methods for ZTID since this method has linear time and space complexity.

To summarize, ZTID design has introduced a new class of tangible user interfaces based on surface behaviors like roll, rub and tap. The ZTID can be used for wireless control of any device or application (e.g., mobile robots, switches, lights, motors, games, software, etc.) where a natural surface gesture input can be used to control the user environment in a ZigBee network.

## References

- [1] C. Wang, T. Jiang, and Q. Zhang, *ZigBee network protocols and applications*. Boston, MA, USA: Auerbach Publications, 2014, pp. 232-233.
- [2] H. K. Jung, J. T. Kim, T. Sahama, and C. H. Yang, *Future information communication technology and applications*. Netherlands: Springer, 2013, pp. 390-391.
- [3] F. Bellido-Outeirino, J. Flores-Arias, F. Domingo-Perez, A. Gil-de-Castro, and A. Moreno-Munoz, "Building lighting automation through the integration of DALI with wireless sensor networks," *IEEE Trans. Consum. Electron.*, vol. 58, no. 1, pp. 47-52, 2012.
- [4] B. Jinsung, H. Insung, L. Byoungjoo, and P. Sehyun, "Intelligent household LED lighting system considering energy efficiency and user satisfaction," *IEEE Trans. Consum. Electron.*, vol. 59, no. 1, pp. 70-76, 2013.
- [5] H. Dae-Man and L.Jae-Hyun, "Design and implementation of smart home energy management systems based on ZigBee," *IEEE Trans. Consum. Electron.*, vol. 56, no. 3, pp. 1417-1425, 2010.
- [6] H. Dae-Man and L.Jae-Hyun, "Smart home energy management system using IEEE 802.15.4 and ZigBee," *IEEE Trans. Consumer Electron.*, vol. 56, no. 3, pp. 1403-1410, 2010.
- [7] K. Gill, Y. Shuang-Hua, Y. Fang, and L. Xin, "A ZigBee-based home automation system," *IEEE Trans. Consum. Electron.*, vol. 55, no. 2, pp. 422-430, 2009.
- [8] I. Zualkernan, A. Al-Ali, M. Jabbar, I. Zabalawi, and A. Wasfy, "InfoPods: ZigBee-based remote information monitoring devices for smart-homes," *IEEE Trans. Consum. Electron.*, vol. 55, no. 3, pp. 1221-1226, 2009.
- [9] S. Guangming, D. Fei, Z. Weijuan, and S. Aiguo, "A wireless power outlet system for smart homes," *IEEE Trans. Consum. Electron.*, vol. 54, no. 4, pp. 1688-1691, 2008.
- [10] H. Jinsoo, C. Chang-Sic, and L. Ilwoo, "More efficient home energy management system based on ZigBee communication and infrared remote controls," *IEEE Trans. Consum. Electron.*, vol. 57, no. 1, pp. 85-89, 2011.
- [11] B. Mrazovac, M. Bjelica, D. Kukolj, B. Todorovic, and D. Samardzija, "A human detection method for residential smart energy systems based on ZigBee RSSI changes," *IEEE Trans. Consum. Electron.*, vol. 58, no. 3, pp. 819-824, 2012.
- [12] H. Il-kyu, L. Dae-sung, and B. Jin-wook, "Home network configuring scheme for all electric appliances using ZigBee-based integrated remote controller," *IEEE Trans. Consum. Electron.*, vol. 55, no. 3, pp. 1300-1307, 2009.
- [13] Z. Zhang, "Microsoft Kinect sensor and its effect," *IEEE Multimedia Mag.*, vol. 19, no. 2, pp. 4-10, Feb 2012.
- [14] J. Lee, "Hacking the Nintendo Wii remote," *IEEE Pervasive Comput.*, vol. 7, no. 3, pp. 39-45, July 2008.
- [15] J. Andrews and N. Baker, "Xbox 360 system architecture," *IEEE Micro*, vol. 26, no. 2, pp. 25-37, March 2006.

- [16] B. Ullmer and H. Ishii, "Emerging frameworks for tangible user interfaces," *IBM Syst. J.*, vol. 39, no. 3-4, pp. 915-931, 2000.
- [17] Freescale Semiconductor, "ZigBee/IEEE 802.15.4 standards and architecture overview," Tutorial., 2005. [online]. Available: [http://www.freescale.com/files/training\\_pdf/28081\\_ZIGBEE\\_OVERVIEW\\_WBT.pdf?lang\\_cd=en](http://www.freescale.com/files/training_pdf/28081_ZIGBEE_OVERVIEW_WBT.pdf?lang_cd=en), Accessed on: Mar. 10, 2015.
- [18] R. Morais, M. Fernandes, S. Matos, C. Serodio, P. Ferreira, and M. Reis, "A ZigBee multi-powered wireless acquisition device for remote sensing applications in precision viticulture," *Comput. Electron. in Agriculture*, vol. 62, no. 2, pp. 94-106, 2008.
- [19] Atmel Corporation, "ATmega128RFA1: 8-bit AVR microcontroller with low power 2.4GHz transceiver for ZigBee and IEEE 802.15.4," Datasheet., 2014. [online]. Available: [http://www.atmel.com/Images/Atmel-8266-MCU\\_Wireless-ATmega128RFA1\\_Datasheet.pdf](http://www.atmel.com/Images/Atmel-8266-MCU_Wireless-ATmega128RFA1_Datasheet.pdf), Accessed on: Mar. 16, 2015.
- [20] Digi, "XBee and XBee-PRO ZB", Datasheet., 2015. [online]. Available: [http://www.digi.com/pdf/ds\\_xbeezbmodules.pdf](http://www.digi.com/pdf/ds_xbeezbmodules.pdf), Accessed on: Mar. 16, 2015.
- [21] Freescale Semiconductor, "1321x-ICB development board reference manual," *Freescale Semiconductor Document Number: 1321xICBBRM, Rev. 0.0*, Jan. 2008.
- [22] Freescale Semiconductor, "1321x-IPB development board reference manual," *Freescale Semiconductor Document Number: 1321xIPBRM, Rev. 0.0*, Jan. 2008.
- [23] Freescale Semiconductor, "MC1321x evaluation kit (EVK) reference manual," *Freescale Semiconductor Document Number: MC1321xEVKRM, Rev. 1.1*, Sept. 2006.
- [24] Freescale Semiconductor, "BeeKit wireless connectivity toolkit user's guide," *Freescale Semiconductor Document Number: BKWCTKUG, Rev. 2.1*, July 2011.
- [25] M. Macniven, "Wireless application development with SynkroRF," Freescale Semiconductor, Tutorial., 2010. [online]. Available: [http://www.freescale.com/files/training/doc/dwf/AMF\\_ENT\\_T1014.pdf](http://www.freescale.com/files/training/doc/dwf/AMF_ENT_T1014.pdf), Accessed on: Mar. 20, 2015.
- [26] Freescale Semiconductor, "1321x-Universal Serial Bus reference manual," *Freescale Semiconductor Document Number: 1321xUCBRM, Rev. 0.0*, Jan. 2008.
- [27] Freescale Semiconductor, "Compact integrated antennas designs and applications for the MC1319x, MC1320x, and MC1321x," *Freescale Semiconductor Document Number: AN2731, Rev. 1.4*, July 2006.
- [28] Freescale Semiconductor, "MC13211/212/213 ZigBee-compliant platform-2.4 GHz low power transceiver for the IEEE 802.15.4 standard plus microcontroller reference manual," *Document Number: MC1321xRM Rev. 1.6*, May 2010.
- [29] Silicon Labs, "CP2102/9 single-chip USB to UART bridge," Datasheet., 2013. [online]. Available:

- <https://www.silabs.com/Support%20Documents/TechnicalDocs/CP2102-9.pdf>, Accessed on: Mar. 20, 2015.
- [30] Freescale Semiconductor, “ $\pm 1.5g - 6g$  three axis low-g micromachined accelerometer,” *Freescale Semiconductor Document Number: MMA7260Q Rev. 1.0*, June 2005.
  - [31] Freescale Semiconductor, “MC13211/212/213 ZigBee-compliant platform-2.4 GHz low power transceiver for the IEEE 802.15.4 standard plus microcontroller,” *Freescale Semiconductor Document Number: MC1321x, Rev.1.2*, May 2007.
  - [32] Camelion, “Product catalogue,” Datasheet., 2015. [online]. Available: [http://www.camelion.com/fileadmin/camelion/kataloge/2015-Camelion\\_en-web.pdf](http://www.camelion.com/fileadmin/camelion/kataloge/2015-Camelion_en-web.pdf), Accessed on: Mar. 20, 2015.
  - [33] N. Krishnan, D. Colbry, C. Juillard, and S. Panchanathan, “Real time human activity recognition using tri-axial accelerometers,” *Sensors Signals and Inform. Process. Workshop*, 2008.
  - [34] G. Krassnig, D. Tantinger, C. Hofmann, T. Wittenberg, and M. Struck, “User-friendly system for recognition of activities with an accelerometer,” in *Proc. of the 4<sup>th</sup> Int. Conf. on Pervasive Comput. Technologies for Healthcare*, 2010, pp. 1-8.
  - [35] T. Yuan and B. Wang, “Accelerometer-based Chinese traffic police gesture recognition system,” *Chinese J. of Electron.*, vol. 19, no. 2, pp. 270-274, 2010.
  - [36] N. Helmi and M. Helmi, “Applying a neuro-fuzzy classifier for gesture-based control using a single wrist-mounted accelerometer,” in *Proc. IEEE Int. Symp. on Comput. Intell. in Robot. and Autom.*, 2009, pp. 216-221.
  - [37] N. Krishnan and S. Panchanathan, “Analysis of low resolution accelerometer data for continuous human activity recognition,” in *Proc. IEEE Int. Conf. on Acoust., Speech, Signal Process.*, 2008, pp. 3337-3340.
  - [38] S. Orlandi and L. Bocchi, “Discrimination of fatigue in walking patterns,” in *Proc. Int. Federation for Med. Biol. Eng.*, 2009, pp. 1275-1278.
  - [39] H. Rubaiyeat, T. Kim, and M. Hasan, “Real-time recognition of daily human activities using a single tri-axial accelerometer,” in *Proc of the 5<sup>th</sup> Int. Conf. on Embedded and Multimedia Comput.*, 2010, pp. 1-5.
  - [40] J. Liu, Z. Wang, L. Zhong, J. Wickramasuriya, and V. Vasudevan, “uWave: Accelerometer-based personalized gesture recognition and its applications,” in *Proc. of the IEEE Int. Conf. on Pervasive Comput. Commun.*, 2009, pp. 1-9.
  - [41] A. Ferscha, S. Vogl, B. Emsenhuber, and B. Wally, “Physical shortcuts for media remote controls,” in *Proc. of the 2nd int. conf. on Intell. technologies for interactive entertainment*, 2008, pp. 9:1-9:8.
  - [42] K. Altun and B. Barshan, “Human activity recognition using inertial/magnetic sensor units,” in *Proc. of the 1st Int. Conf. on Human Behavior Understanding*, 2010, pp. 38-51.
  - [43] M. Lee, A. Khan, J. Kim, Y. Cho, and T. Kim, “A single tri-axial accelerometer-based real-time personal life log system capable of activity classification and exercise information generation,” in *Proc. of the Annu. Int. Conf. of the IEEE Eng. in Med. and Biol. Soc.*, 2010, pp. 1390-1393.

- [44] J. Yang, Y. Chen, G. Lee, S. Liou, and J. Wang, "Activity recognition using one triaxial accelerometer: A neuro-fuzzy classifier with feature reduction," in *Proc. of the 6<sup>th</sup> Int. Conf. on Entertainment Comput.*, 2007, pp. 395-400.
- [45] A. Khan, Y. Lee, S. Lee, and T. Kim, "Accelerometer's position independent physical activity recognition system for long-term activity monitoring in the elderly," *Medical and Biological Eng. and Comput.*, vol. 48, no. 12, pp. 1271-1279, 2010.
- [46] U. Kawamoto, T. Kurata, N. Sakata, T. Okuma, and H. Kuzuoka, "Position/orientation-aware physical tags using photo sensors and accelerometers for a tangible tabletop interface," in *Proc. IEEE Int. Conf. on Distrib. Human-Mach. Syst.*, 2008, pp. 397-404.
- [47] A. Reiss, G. Hendebay, G. Bleser, and D. Stricker, "Activity recognition using biomechanical model based pose estimation," in *Proc. of the 5<sup>th</sup> Eur. Conf. on Smart Sens. and Context*, 2010, pp. 42-55.
- [48] V. Baier, L. Mosenlechner, and M. Kranz, "Gesture classification with hierarchically structured recurrent self-organizing maps," in *Proc. of the 4<sup>th</sup> Int. Conf. on Networked Sens. Syst.*, 2007, pp. 81-84.
- [49] L. Sun, D. Zhang, B. Li, B. Guo, and S. Li, "Activity recognition on an accelerometer embedded mobile phone with varying positions and orientations," in *Proc. of the 7<sup>th</sup> Int. Conf. on Ubiquitous Intell. and Comput.*, 2010, pp. 548-562.
- [50] J. Yang, "Toward physical activity diary: motion recognition using simple acceleration features with mobile phones," in *Proc. of the 1st Int. Workshop on Interactive Multimedia for Consum. Electron.*, 2009, pp. 1-10.
- [51] X. Zhang, X. Chen, W. Wang, J. Yang, V. Lantz, and K. Wang, "Hand gesture recognition and virtual game control based on 3D accelerometer and EMG sensors," in *Proc. of the 14<sup>th</sup> Int. Conf. on Intell. User Interfaces*, 2009, pp. 401-405.
- [52] Z. Prekopcsak, "Accelerometer based real-time gesture recognition," CiteSeerX, 2008. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.148.5590&rep=rep1&type=pdf>, Accessed on: Mar. 26, 2015.
- [53] J. Mantyjarvi, J. Kela, P. Korpipaa, and S. Kallio, "Enabling fast and effortless customisation in accelerometer based gesture interaction," in *Proc. of the 3rd Int. Conf. on Mobile and Ubiquitous Multimedia*, 2004, pp. 25-31.
- [54] Z. Dong, U. Wejinya, S. Zhou, Q. Shan, and W. Li, "Real-time written-character recognition using MEMS motion sensors: calibration and experimental results," in *Proc. IEEE Int. Conf. on Robot. and Biomimetics*, 2008, pp. 687-691.
- [55] I. Hillgaar, "The magic wand," Master's thesis, Univ. of Dublin, Dublin, California, 2006.
- [56] J. Kela, P. Korpipaa, J. Mantyjarvi, S. Kallio, G. Savino, L. Jozzo, and D. Marca, "Accelerometer-based gesture control for a design environment," *Pers. and Ubiquitous Comput.*, vol. 10, no. 5, pp. 285-299, 2006.

- [57] C. Fang, "From dynamic time warping (DTW) to hidden Markov model (HMM)," Final project report for ECE742 stochastic decision, 2009. [Online]. Available: [http://www.cs.uc.edu/~fangcg/course/-FromDTWtoHMM\\_ChunshengFang.pdf](http://www.cs.uc.edu/~fangcg/course/-FromDTWtoHMM_ChunshengFang.pdf), Accessed on: Mar. 28, 2015.
- [58] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, vol. 11, no. 5, pp. 561-580, 2007.
- [59] N. Minematsu, Y. Fujisawa, and S. Nakagawa, "Performance comparison among HMM, DTW, and human abilities in terms of identifying stress patterns of word utterances," in *Proc. of the 6<sup>th</sup> Int. Conf. on Spoken Language Process.*, 2000, pp. 617-620.
- [60] F. Hofmann, P. Heyer, and G. Hommel, "Velocity profile based recognition of dynamic gestures with discrete hidden Markov models," in *Proc. of the Int. Gesture Workshop on Gesture and Sign Language in Human-Computer Interaction*, 1998, pp. 81-95.
- [61] V. Mantyla, "Discrete hidden Markov models with application to isolated user-dependent hand gesture recognition," *VTT Publications*, vol. 4, no. 4, p. 9, 2001.
- [62] Freescale Semiconductor, "Using the MC1321x internal transmit/receive switch in a low-cost, single-port, two-layer design," *Freescale Semiconductor Document Number: AN3248, Rev. 0.0*, Mar. 2006.
- [63] Freescale Semiconductor, " $\pm 1.5g, \pm 6g$  three axis low-g micromachined accelerometer," *Freescale Semiconductor Document Number: MMA7361L Rev. 0.0*, Apr. 2008.
- [64] Freescale Semiconductor, "Running SMAC based demonstration operations," *Freescale Semiconductor Document Number: AN3231, Rev. 1.1*, Oct. 2006.
- [65] Freescale Semiconductor, "PCB layout guidelines for the MC1321x," *Freescale Semiconductor Document Number: AN3149, Rev. 0.0*, Mar. 2006.
- [66] K. Tuck and C. Han, "Soldering and mounting guidelines for the LGA accelerometer sensor to a PC board," *Freescale Semiconductor Document Number: AN3484, Rev 2.0*, Dec. 2008.
- [67] S. Smith, *The scientist and engineer's guide to digital signal processing*. San Diego, CA, USA: California Technical Publishing, 1997, pp. 277-280.
- [68] T. Schlomer, B. Poppinga, N. Henze, and S. Boll, "Gesture recognition with a Wii controller," in *Proc. of the 2nd Int. Conf. on Tangible and Embedded Interaction*, 2008, pp. 11-14.
- [69] K. Doo, "A design framework for 3D spatial gesture interfaces," Ph.D. dissertation, Univ. of Washington, USA, 2008.
- [70] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. of the IEEE*, vol. 77, no. 2, pp. 257-286, Feb 1989.
- [71] D. Jurafsky and J. H. Martin, *Speech and language processing*. Pearson Education India, 2000, pp. 169-173.
- [72] S. Cho, "Introduction to hidden Markov model and Its application," CiteSeerX, 2005. [Online]. Available: [http://enpub.fulton.asu.edu/cseml/05fall/-s3\\_HMM\\_slides.pdf](http://enpub.fulton.asu.edu/cseml/05fall/-s3_HMM_slides.pdf), Accessed on: Mar. 28, 2015.

- [73] Freescale Semiconductor, “Accelerometer demonstration with the 13213-SRB (Sensor Reference Board),” *Freescale Semiconductor Document Number: AN3232, Rev. 1.1*, Mar. 2008.
- [74] J. KIM. (2013, Feb. 6). HMM forward algorithm - c source code. [Online]. Available: <http://feelmare.blogspot.ae/2013/02/hmm-forward-algorithm-source-code.html>, Accessed on: Mar. 28, 2015.
- [75] W. Khan. (2014, Nov. 20). S-NE-TI-Device-Wireless-Link-Quality. [YouTube video]. Available: <https://www.youtube.com/watch?v=UjKz4NMJ0YI>, Accessed on: Nov. 20, 2014.
- [76] W. Khan. (2014, Nov. 20). S-NE-TI-Device-Wireless-Accelerometer-Data. [YouTube video]. Available: <https://www.youtube.com/watch?v=5zaxst1heWo>, Accessed on: Nov. 20, 2014.
- [77] F. Dieterle, “Multianalyte quantifications by means of integration of artificial neural networks, genetic algorithms and chemometrics for time-resolved analytical data,” Ph.D. dissertation, Univ. of Tübingen, 2003.
- [78] Freescale Semiconductor, “A guide to CodeWarrior development studio for microcontrollers,” Tutorial., 2007. [online]. Available: [http://cache.freescale.com/files/soft\\_dev\\_tools/doc/fact\\_sheet/CWS-MCU-BG.pdf?fsrch=1](http://cache.freescale.com/files/soft_dev_tools/doc/fact_sheet/CWS-MCU-BG.pdf?fsrch=1), Accessed on: Mar. 28, 2015.
- [79] R. Rhoades, “CodeWarrior development studio for microcontrollers v10.0,” Tutorial., 2010. [online]. Available: [http://www.freescale.com/files/training\\_pdf/WBNR\\_FTF10\\_ENT\\_F0669.pdf?lang\\_cd=en](http://www.freescale.com/files/training_pdf/WBNR_FTF10_ENT_F0669.pdf?lang_cd=en), Accessed on: Mar. 28, 2015.

## Appendices

### Appendix A: MC1321x Block Diagram

The block diagram of MC1321x is given in Figure 37.

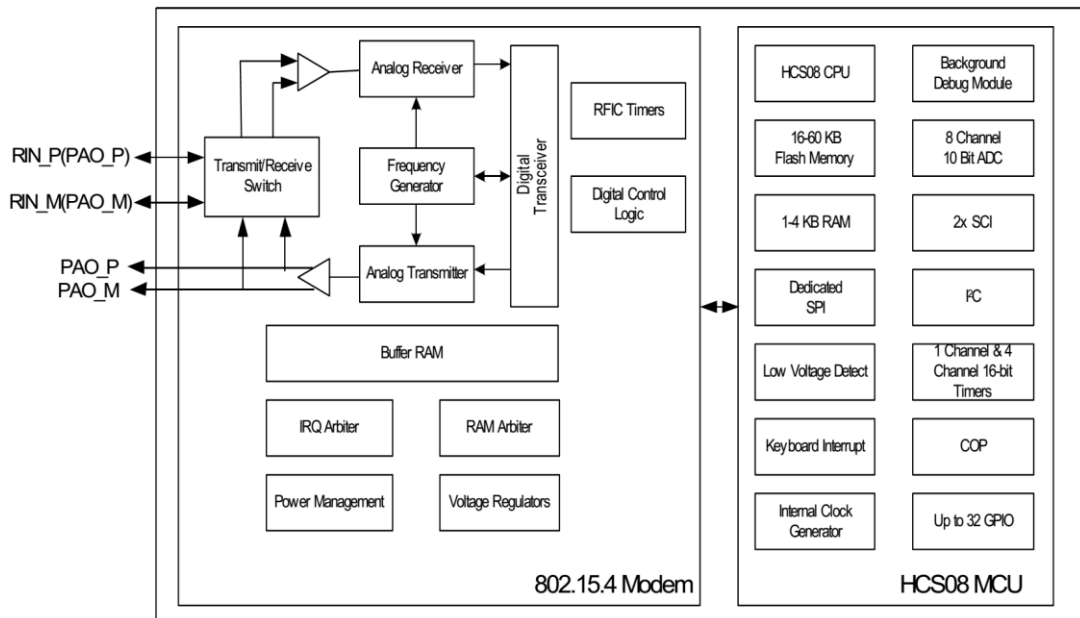


Figure 37: Block diagram of MC1321x [31]

MC1321x comes in three package options:

- MC13211 with 16KB of flash and 1KB of RAM
- MC13212 with 32K of flash and 2KB of RAM
- MC13213 with 60K of flash and 4KB of RAM

## Appendix B: Network Implementation for MC1321x

Freescale provides a GUI known as BeeKit. The BeeKit GUI is shown in Figure 38. The network for the selected hardware can be configured and implemented using the BeeKit GUI. As an example, suppose that the network hardware is selected to be MC1321x. After the network configuration, the network solution can be exported [24]. The exported network solution can then be imported in Freescale Code Warrior classic [78] if running Windows XP or Vista. The import procedure is given in [64].

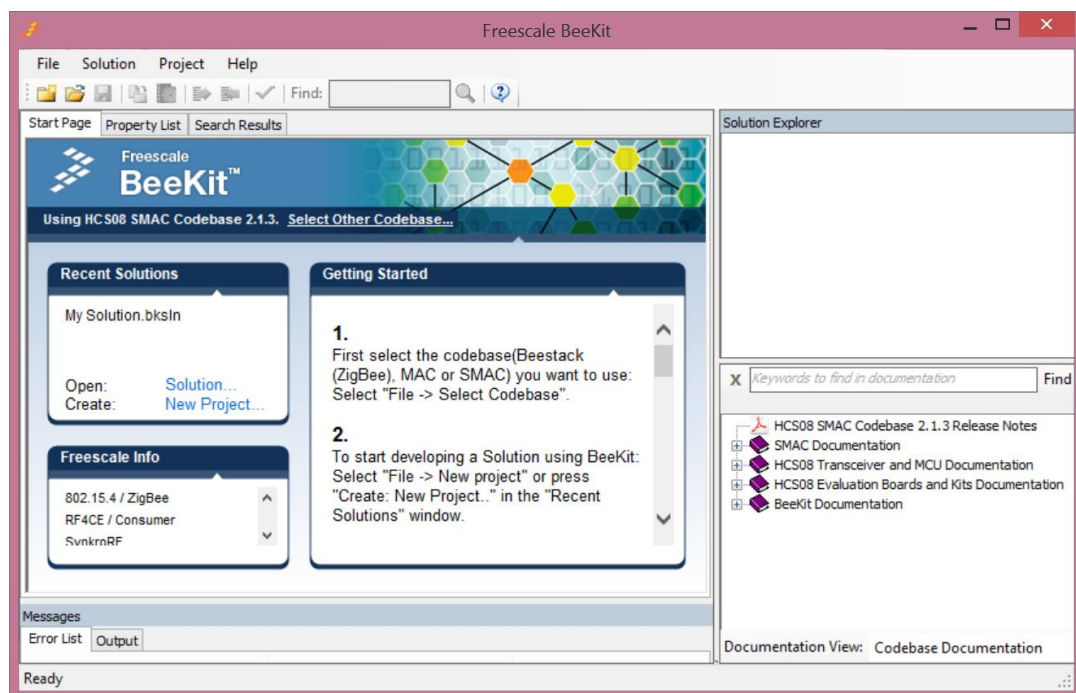


Figure 38: Network implementation using BeeKit GUI [24]

For Windows 7/8, Freescale Code Warrior 10.1 can be used with the import procedure given in [79]. Using Freescale Code Warrior classic, the import procedure is shown in Figure 39. The project file is imported as an .xml file and then saved as an .mcp file. Once the project file is created as .mcp, it can be programmed to the target board. The programming method is given in [64].

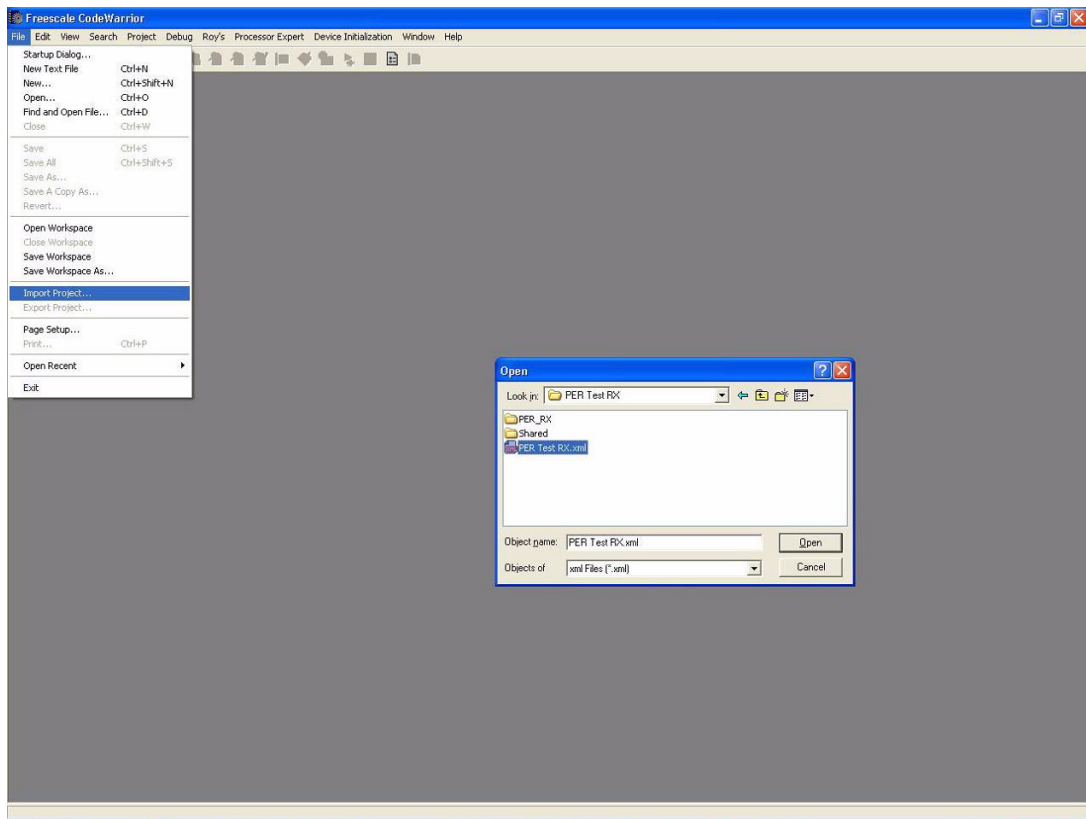


Figure 39: Importing BeeKit solution in Freescale Code Warrior classic [64]

## Appendix C: Casing Modification

The casing modification is subdivided into modifications that were done in the selected casing [32] for the RF-control-board and LED-switch-board. First the casing modifications done for LED-switch-board are described followed by the RF-control-board.

### LED-switch-board

For the LED-switch-board, first the modification was done in the casing and then board form factor was selected. The LED and the LED holder of the original casing were removed as shown in Figure 40 step 1 and 2. The final form factor used in the design of the LED-switch-board and the location of connectors was selected by making a dummy LED-switch-board as shown in Figure 40 step 3 and step 4.

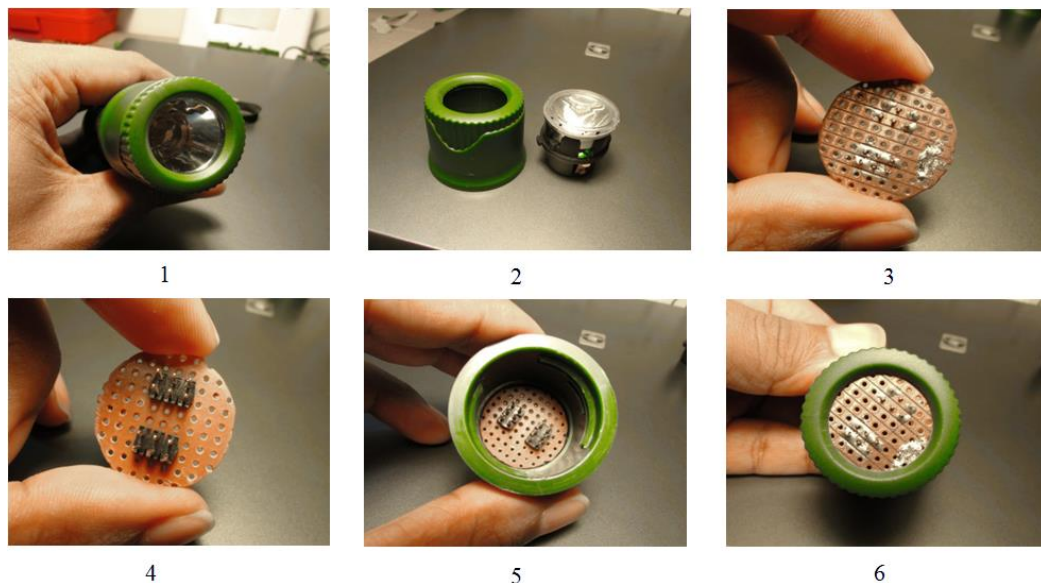


Figure 40: Casing modifications for the LED-switch-board

This dummy LED-switch-board was tested on the LED-holder as shown in Figure 40 step 5 and 6. It was found that the final shape for the dummy LED-switch-board should be round with a diameter of 26 mm.

### RF-control-board

For the RF-control-board, first the modifications were done in the casing and then the form factor was selected for RF-control-board.

First, two metal rings were removed along with two thin plates. The removal of a single metal ring is shown in Figure 41 step 1 and 2. These metal rings are used in combination with two thin plates to provide current for the torch LED of the selected casing [32]. These metal rings along with thin plates are removed so that they don't interfere with chip antenna performance. Step 3 shows the torch battery holder. It is a 3 AAA battery holder that gives 4.5 volts. But for ZTIDs, all the three main chips work on 3 volts. So it was decided to remove one cell and modify the casing. The modified casing is shown in step 4. This was done to get space for the RF-control-board. Also, a part from the side of the battery holder was modified to make space for the data cable. Finally, two holes were made with threads on the back side of this battery holder shown in step 5 so that the final assembly could be fixed from outside the casing using screws.

Then a dummy board was made to find the final shape and dimensions of the RF-control-board. This dummy board is shown being tested on the battery holder in step 6. It was found that in order for the RF-control-board to fit inside the battery holder, the dimensions should be exactly in (X, Y) mm (20, 45). A ribbon cable was connected from a dummy RF-control-board to dummy LED-switch-board for defining a new interface and to find the pin definitions for this interface. This connection is shown in step 7. Two micro-USB ports were mounted on the dummy RF-control-board and the whole assembly was put inside the round casing as shown in step 8. This was done because the micro-USB ports would not go inside the round casing if they were mounted at the corners. So it was found that the dimensions for the RF-control-board must be 17mm from the bottom side where two micro-USB ports are mounted. So by this method the final form factor of the RF-control-board was adopted.

Finally, two holes were made in the original casing shown in step 9. This was done to make space for two micro-USB ports as shown in step 10. Then two screws were mounted to fix the battery holder as shown in step 11. Finally, device stickers were designed for the accelerometer, USB port and the programming port connections as shown in step 12.



Figure 41: Casing modifications for RF-control-board

## Appendix D: Surface Gestures

### Gesture Roll-X

The gesture Roll-X means device rolling about its X-axis. The sequence for this gesture is shown in Figure 42.

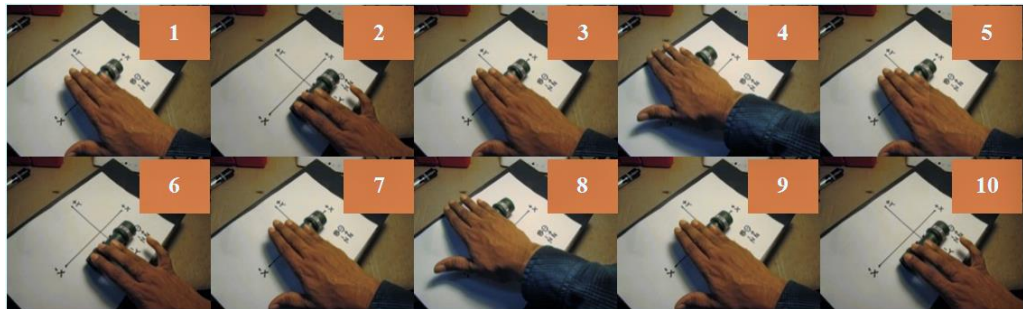


Figure 42: Gesture Roll-X

### Gesture Rub-4-X

The gesture Rub-4-X means device rubbing in X-axis direction four times. This gesture sequence is shown in Figure 43.

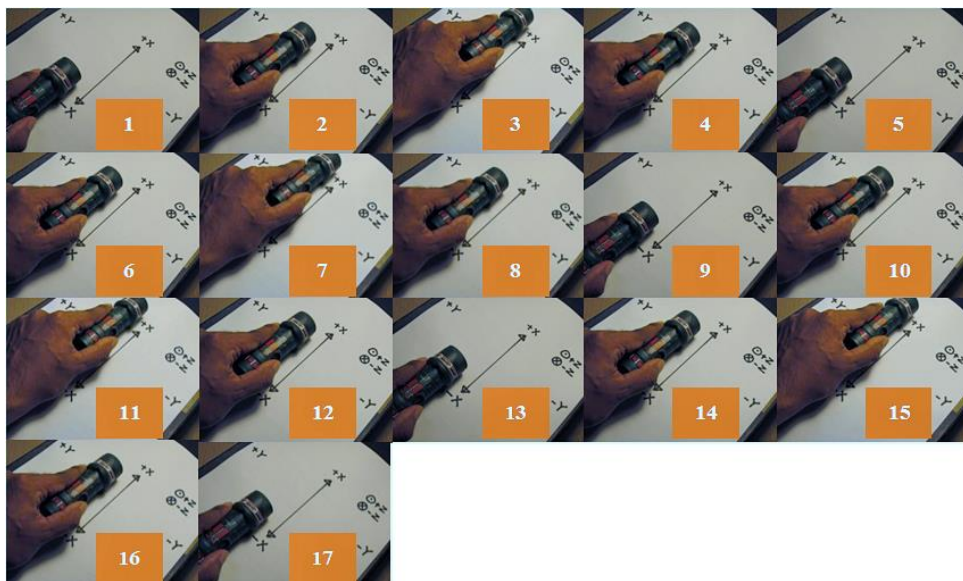


Figure 43: Gesture Rub-4-X

### Gesture Rub-4-Y

The gesture Rub-4-Y means device rubbing in Y-axis direction four times. This gesture is shown in Figure 44.

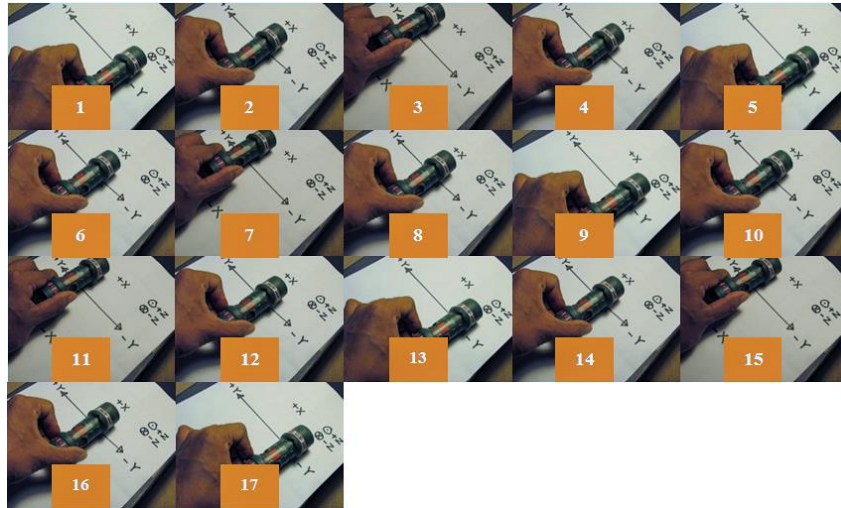


Figure 44: Gesture Rub-4-Y

### Gesture Tap-1

The gesture Tap-1 means device is tapped on a surface one time. This gesture is shown in Figure 45.



Figure 45: Gesture Tap-1

### Gesture Tap-2

The gesture Tap-2 means device is tapped on a surface two times. This gesture is shown in Figure 46.



Figure 46: Gesture Tap-2

### Gesture Tap-3

The gesture Tap-3 means device is tapped on a surface three times. This gesture is shown in Figure 47.

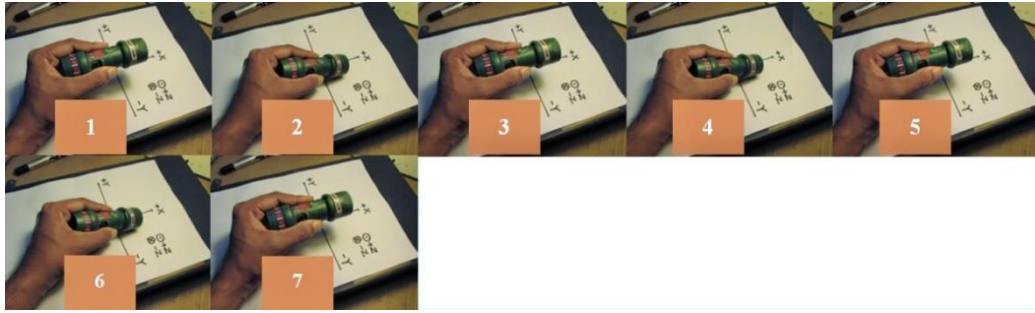


Figure 47: Gesture Tap-3

#### Gesture Tap-4

The gesture Tap-4 means device is tapped on a surface four times. This gesture is shown in Figure 48.

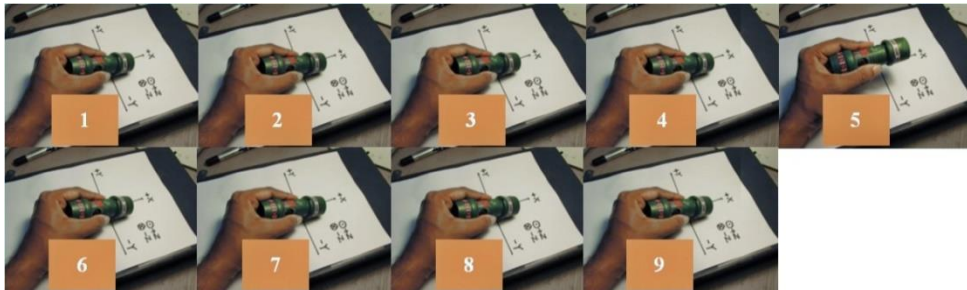


Figure 48: Gesture Tap-4

#### Gesture Tap-5

The gesture Tap-5 means device is tapped on a surface five times. This gesture is shown in Figure 49.

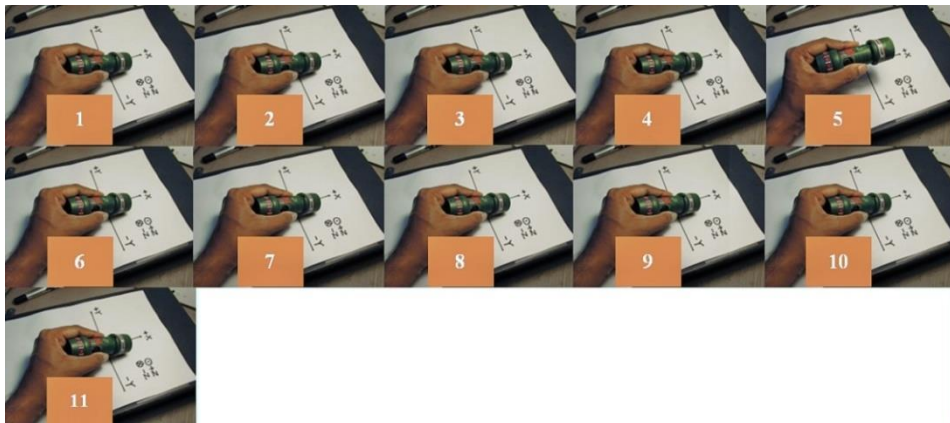


Figure 49: Gesture Tap-5

## Appendix E: Experimental Setup for Data Collection

For data collection using HyperTerminal, two ZTIDs were programmed with the Freescale accelerometer project [73] that hardcodes an option for both the devices to work either as transmitters for accelerometer data or to become a receiver. The device on which switch 1 is pressed becomes a transmitter device for accelerometer data and the other becomes an accelerometer data receiver automatically. The receiver device is hardcoded to send the received acceleration to one of its serial ports at a fixed baud rate only when it receives a capital letter “V” from the serial port. For all ZTIDs, this serial port is connected to USB controller CP2102 in the design to make a virtual com port in the PC for receiving accelerometer data.

Initially when the devices are turned on, they are both in low power mode and in wait state for switch press. Both devices were first turned on using the devices on/off switch. Initially, both were in low power mode. Then one of the devices was made to transmit its accelerometer data by pressing switch 1. The other device then became the receiver. The receiver device sent the received acceleration to the PC through its USB port that acted as a virtual com port whenever it received a capital letter “V” from the PC. The device’s hardware setup for data collection is shown in Figure 50.

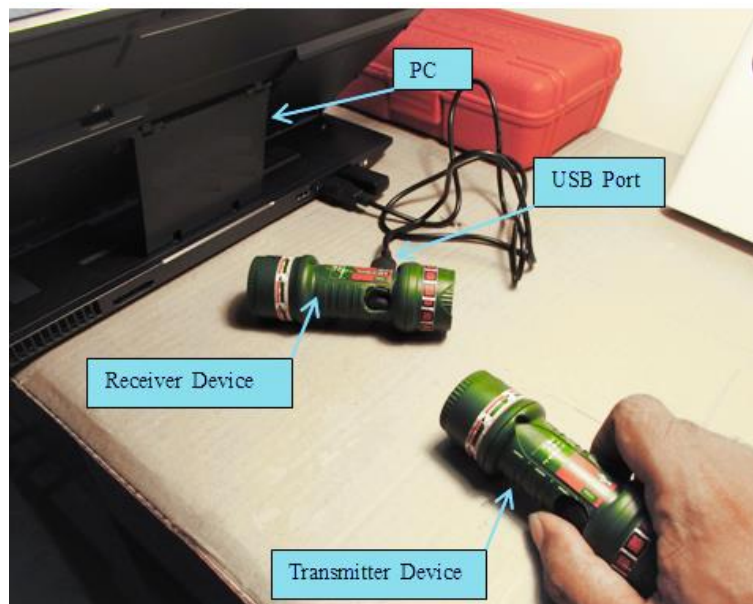


Figure 50: Device setup for data collection using HyperTerminal

Before the data is collected, the accelerometer on transmitter device of Figure 50 must be calibrated. The calibration is done on device accelerometer MMA7361L by the MC13213 micro-controller. The calibration function comes with the Freescale accelerometer project [73] that was programmed in the devices.

To start the calibration, the transmitter device of Figure 50 must be first placed and oriented with the device sticker coming on the top side as shown in Figure 50. Then switch 4 on the transmitter device must be pressed to start the calibration. Once the device is calibrated, all the 4 LEDs on the transmitter device will blink one time. This will confirm that the transmitter device accelerometer is now calibrated with reference to the device position. So it is important to keep the device straight and properly oriented during the calibration procedure. Once the transmitter device is calibrated, both the transmitter device and the receiver device must be reset. This is done by pressing the reset switch provided on both the devices. The calibrated transmitter device accelerometer data can then be viewed using Freescale Triax Software to confirm that the device is calibrated properly.

The wireless accelerometer data is collected in the PC from the receiver device using windows HyperTerminal. When the receiver device is connected to the PC, the USB controller CP2102 asks for its driver installation. Once the driver is installed, a virtual COM port appears and can be seen in PC hardware device manager. The hardware settings using HyperTerminal are given in Table 22.

Table 22: HyperTerminal setup for ZTID

|                 |       |
|-----------------|-------|
| Bits per second | 38400 |
| Data bits       | 8     |
| Parity          | None  |
| Stop bits       | 1     |
| Flow control    | None  |

To receive the data in the HyperTerminal window, the user must enter a capital letter “V”. The data will come in the HyperTerminal window as long as the letter “V” is pressed. So the amount of time for which the letter “V” is pressed is used to control gesture length. The data is sent by the receiver device of Figure 50 to the PC in the format “ $a_x a_y a_z$ ” as shown in Figure 51. The accelerometer data “ $a_x a_y a_z$ ”

is in ASCII format. The data is copied from the HyperTerminal window to a text file. The text file is then saved with the name of the gesture that is performed.

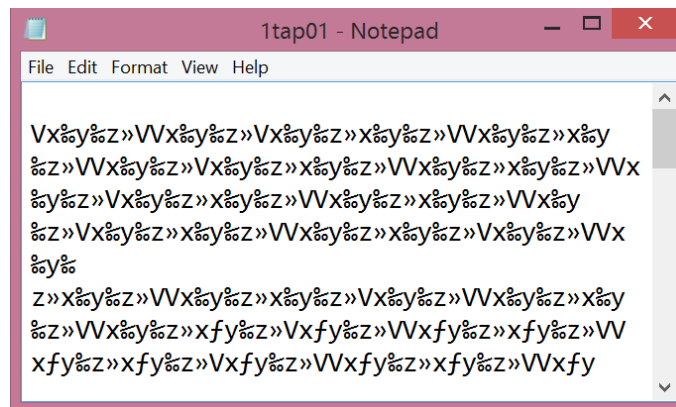


Figure 51: Data in ASCII for a gesture

Since the actual data is always followed by the letters ‘x’ ‘y’ and ‘z’, these letters are helpful for writing a Matlab code to get the actual data in ASCII which can then be converted to decimal.

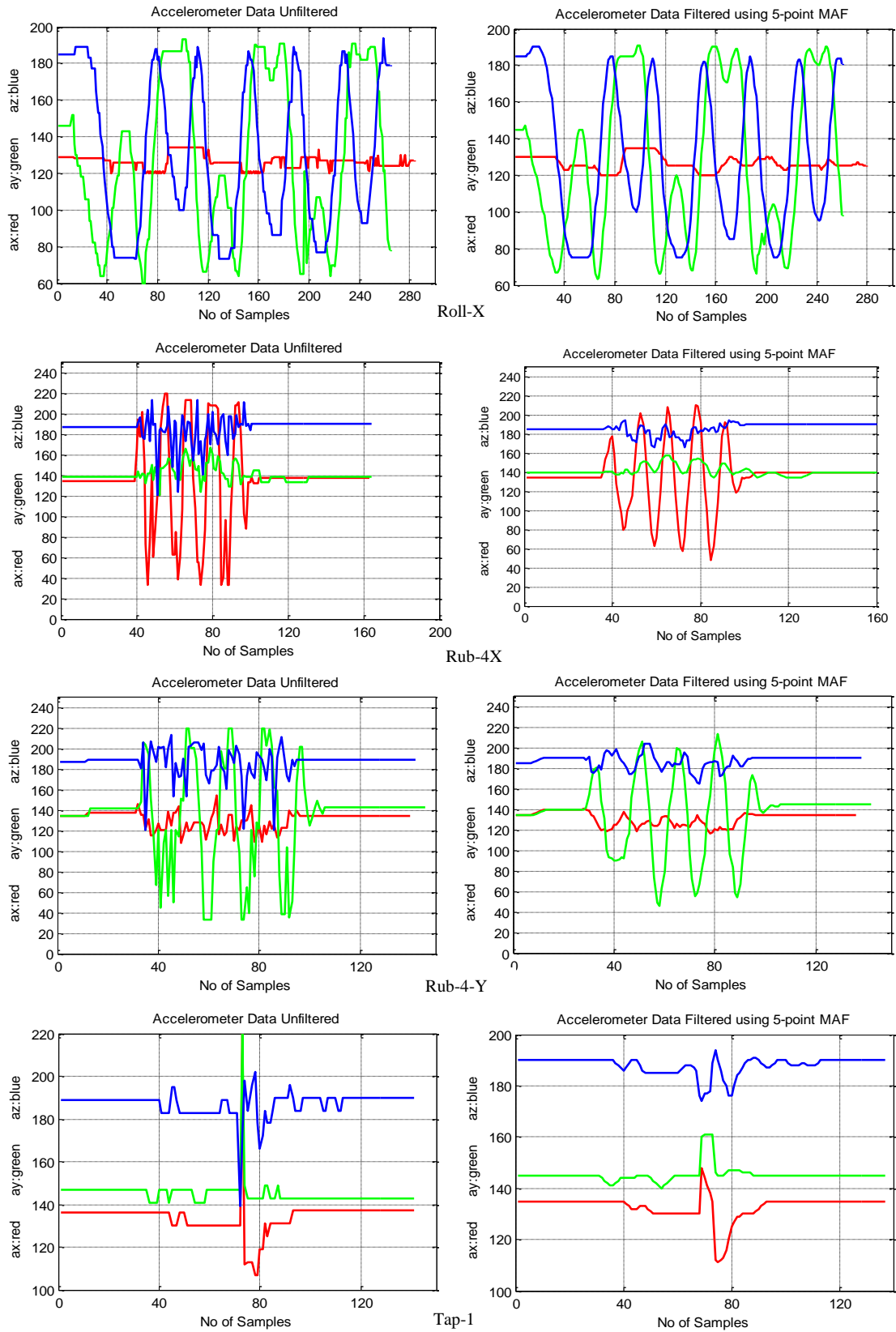
Following the same approach, a Matlab script was written that reads a text file saved from HyperTerminal for a gesture. This data is then converted to decimal and an acceleration vector “p” is created which is the unfiltered acceleration as shown in Figure 52.

| p = |     |     |
|-----|-----|-----|
| 137 | 137 | 187 |
| 137 | 137 | 187 |
| 137 | 137 | 187 |
| 137 | 137 | 187 |
| 137 | 137 | 187 |
| 137 | 137 | 187 |
| 137 | 137 | 187 |
| 137 | 137 | 187 |
| 137 | 137 | 187 |

Figure 52: Un-filtered acceleration vector

Each of the eight gestures Roll-X, Rub-4-X, Rub-4-Y, Tap-1, Tap-2, Tap-3, Tap-4 and Tap-5 were performed 30 times. The time duration for each gesture was different. For each gesture, 30 data points were taken from the HyperTerminal. So in total, 240 data points were taken for eight gestures.

## Appendix F: Filtering and Feature Segmentation Plots



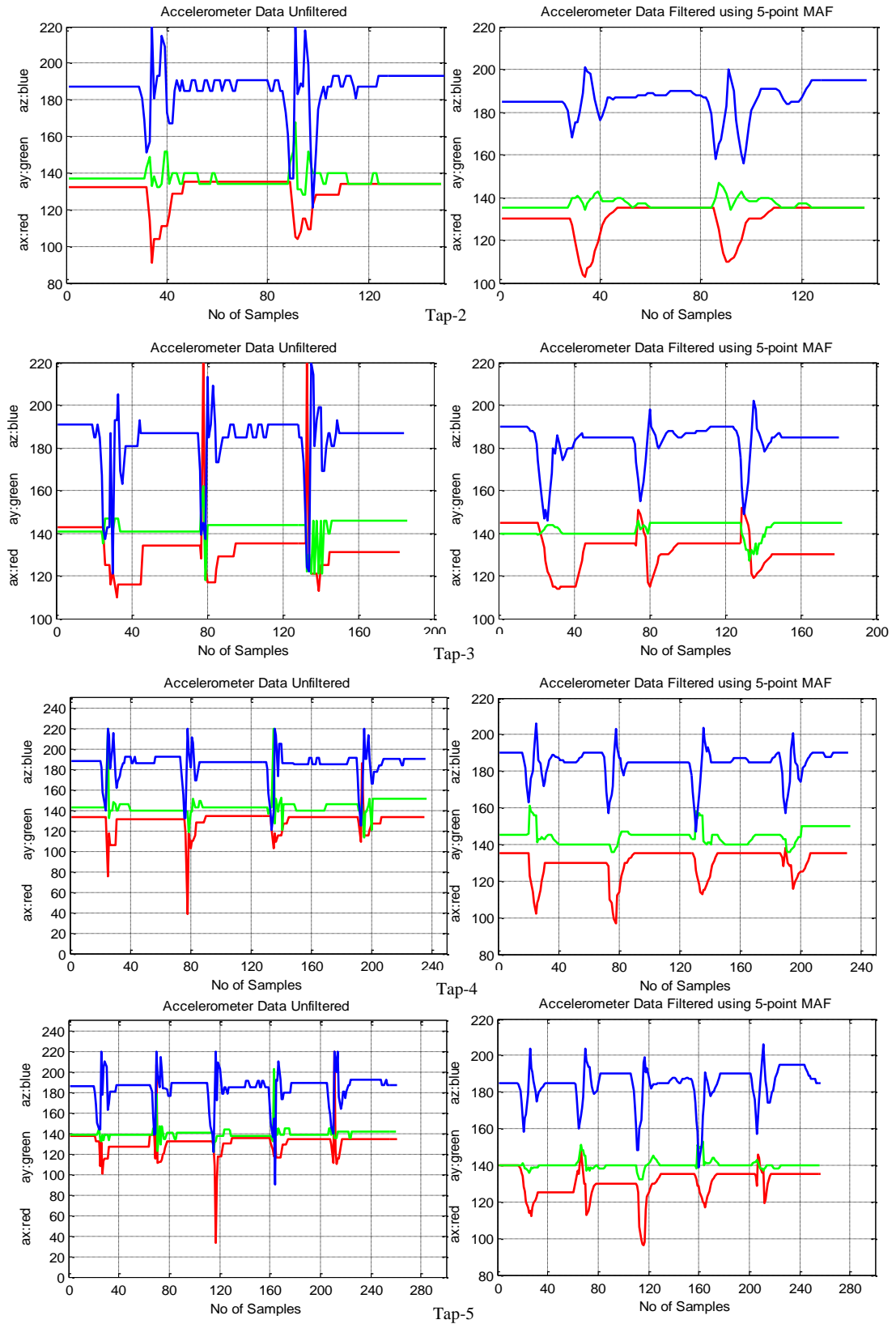


Figure 53: Un-filtered and filtered acceleration for eight gestures

For the filtered acceleration of gesture Roll-X of Figure 53, the cluster centroid using K: 8, 10 and 14 are shown in Figure 54 and the corresponding code books are shown in Figure 55.

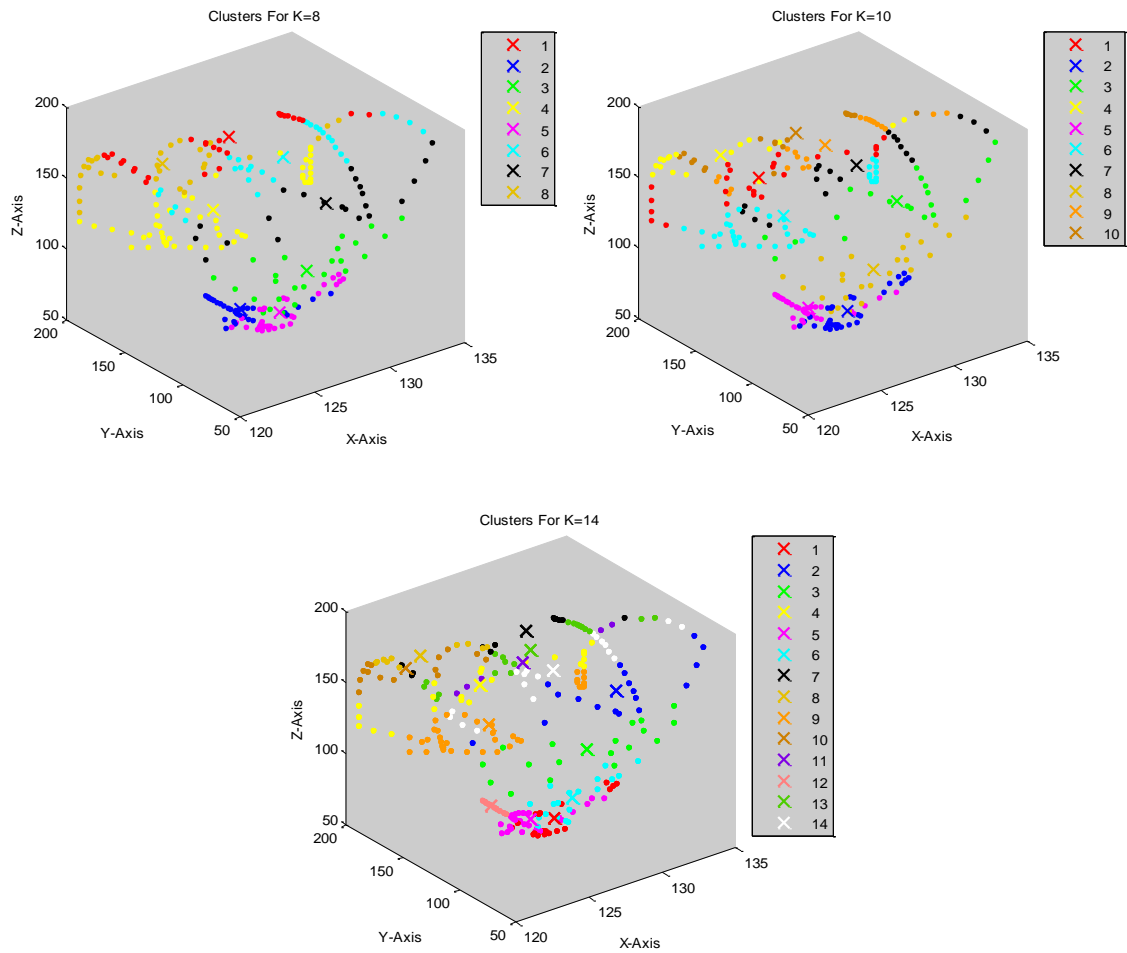


Figure 54: Clusters for gesture Roll-X using K=8, 10 and 14

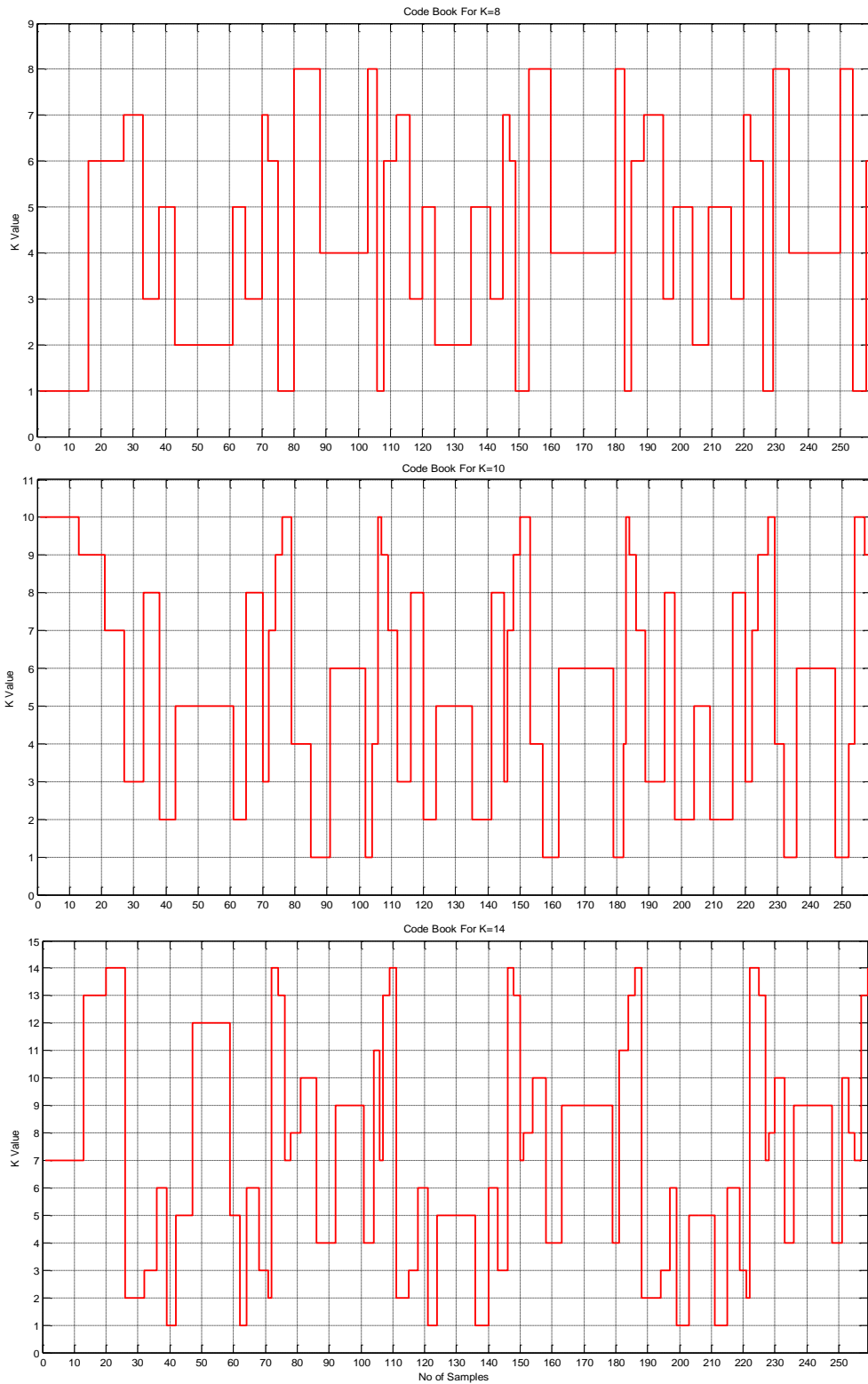


Figure 55: Codebooks for gesture Roll-X using K=8, 10 and 14

## Appendix G: Hidden Markov Modeling

This appendix describes the Markov process, hidden Markov models, HMM structure and the two problems associated with HMM training and recognition [69].

### Markov Processes

The Markov process is summarized by the Markov property: for any sequence of time domain events, the conditional probability density of a current event given all the past and present events depends only on the most recent  $j$  events. This is represented as follows:

$$P(X_{t+1} = sj_{t+1} | X_t = sj_t, X_{t-1} = sj_{t-1}, \dots, X_0 = sj_0) = P(X_{t+1} = sj_{t+1} | X_t = sj_t, \dots, X_{t-j+1} = sj_{t-j+1})$$

At each time step the state is changed with a given transition probability  $a_{ij}$ . Figure 56 shows an example of a Markov process where each state is indicated by its circle. The graph visualizes a finite state machine that consists of states, transitions and actions. Each directed line is a transition from one state ( $i$ ) to another state ( $j$ ), whose probability is indicated by  $a_{ij}$  alongside the line.

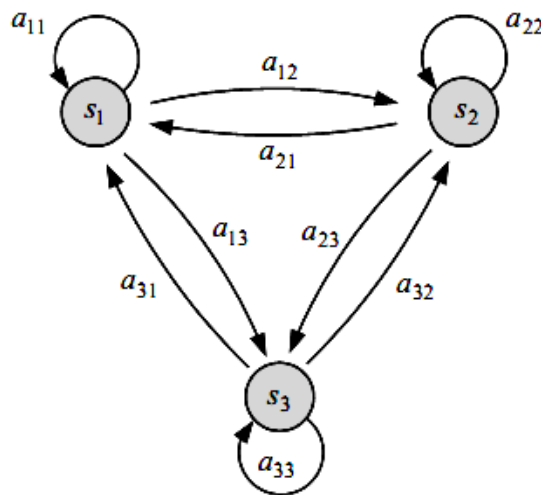


Figure 56: Example of a Markov process [69]

The transitions are normally represented by a simple probabilistic model while the states generally involve more complex stochastic models. In an observable Markov model, the states correspond to random processes whose outcomes are directly observable.

## Hidden Markov Model

In a hidden Markov model (HMM), the output for each state will be hidden (not observable) and can only be observed through another set of observable stochastic processes.

State transition is a stochastic process and a sequence of hidden states is inferred from the observed data. Each hidden state of the model is associated with a set of output probability distributions, which are characterized by either discrete probability distributions or continuous probability distribution functions.

As shown in Figure 57, HMMs are described in terms of two random processes. The first process is a discrete-time Markov chain, meaning that the system is in one of a finite number of states ( $s_1, \dots, s_N$ ) at each time  $t = 1, 2, \dots$ . The process starts in one of these states and moves successively from one state to another. The state change is determined by a set of transition probabilities ( $a_{ij}$ ) associated with the current state. The second random process generates an output symbol  $o \in \{o_1, \dots, o_k\}$  at every time step, subject to a probability distribution  $b_j$ , which depends on the current state.

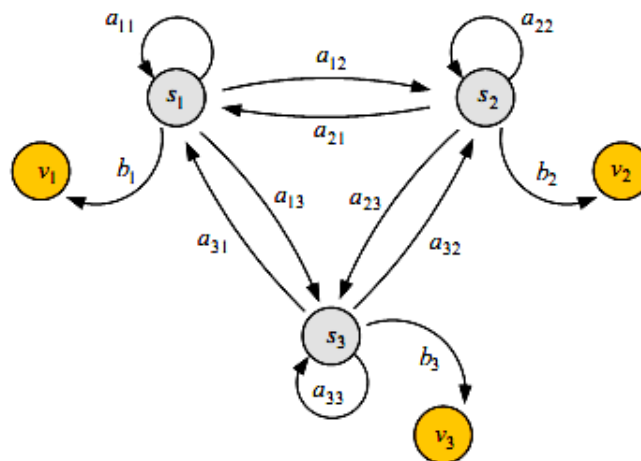


Figure 57: Example of hidden Markov modeling [69]

A hidden Markov model consists of two random processes: the first process is a Markov chain with a state transition and the second process provides observable output symbols depending on these hidden states.

We can describe the HMM formally with a set of notations. An HMM ( $\lambda$ ) is defined by the number of states  $N$  and three sets of probability distributions. The states ( $Q$ ) are simply  $\{1, 2, \dots, N\}$  and the state at time  $t$  is denoted as  $q_t$ . There are three different probability distributions. First, an HMM includes the initial state distribution,  $\pi = \{\pi_i\}$ , where  $\pi_i = P\{q_i = i\}$ ,  $1 \leq i \leq N$ . The second probability distribution is a set of state transition probabilities,  $A = \{a_{ij}\}$ , where  $a_{ij} = p\{q_{t+1} = j | q_t = i\}$ ,  $1 \leq i \leq N$ . The current state is denoted by  $q_t$ . The third set of distributions is an output probability in each of the states,  $B = \{b_j(k)\}$ ,  $b_j(k) = p\{o_t = v_k | q_t = j\}$ ,  $1 \leq j \leq N$ ,  $1 \leq k \leq M$ , where  $v_k$  denotes the  $k^{\text{th}}$  observation symbol in the alphabet, and  $o_t$  the current parameter vector. If the observations are continuous then we use a continuous probability density function, instead of a set of discrete probabilities. In this case we specify the parameters of the probability density function. Usually the probability density is approximated by a weighted sum of  $M$  Gaussian distributions  $N$ ,  $b_j(o_t) = \sum_{m=1}^M c_{jm} N(\mu_{jm}, U_{jm}, o_t)$  where  $c_{jm}$  is the weighting coefficients,  $\mu_{jm}$  are mean vectors, and  $U_{jm}$  are covariance matrices. Figure 58 shows a one dimensional Gaussian mixture probability distribution function consisting of two single Gaussians (red and blue).

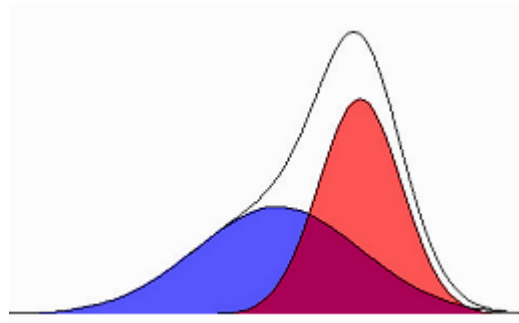


Figure 58: Example of two Gaussian mixtures [69]

A Gaussian mixture probability distribution function is constructed from two Gaussian probability distributions (red and blue). Based on the notations described above, a complete notation of an HMM can be defined as follows: an HMM with discrete probability distributions,  $\lambda = (\pi, A, B)$ , and an HMM with continuous densities,  $\lambda = (\pi, A, c_{jm}, \mu_{jm}, U_{jm})$ . Specification of an HMM involves the choice of the number of states,  $N$ , the number of discrete symbols  $L$ , and specification of three

probability densities with matrix form  $A$ ,  $B$ , and  $\pi$ . A set of initial states  $Q_i$  and final states  $Q_f$  can also be defined. Thus, transitions must start from one of  $Q_i$  and end at one of  $Q_f$ .

A left-right HMM has the property that limits state transitions only to consecutive states with no backward transition. Left-right HMM models have been widely used for speech recognition because of the relatively short duration of phonemes and an underlying structure of language based on phonetic ordering.

In Ergodic HMM the states are fully connected, i.e. transitions may occur from any state. In this case, any state is reached from any other state in a finite number of steps.

The two problems associated with HMM for HMM training and evaluations are:

- Model training problem
- Model evaluation problem

The model training and evaluation problem is defined by [69] as:

#### Model Training Problem

This problem is for estimating the model parameters and is solved using the Baum-Welch re-estimation algorithm.

In this problem, the observation sequence  $O = O_1, O_2, \dots, O_T$  is given and the goal is to adjust the model parameters  $\lambda = (A, B, \pi)$ , to maximize  $P(O|\lambda)$  [70]. So the learning problem is to adjust the HMM parameters to determine an HMM model for each gesture. The multiple observation sequences  $K$  (training data) can be used to train the model [69]. When  $P(O|\lambda) = \prod_{k=1}^K P(O^{(k)}|\lambda)$  is maximized, the model parameters  $A, B$  and  $\pi$  are obtained.

Maximum likelihood (ML) maximizes the probability of a given sequence of observations  $O$ , belonging to a given gesture class  $g$ , and gives the trained model HMM  $\lambda_*$  for the class  $g$ . This probability is the total likelihood of the observations and can be expressed mathematically as

$$L_{tot} = p\{O|\lambda_*\}.$$

There is no known way to solve this problem analytically. The most common HMM training technique is an iterative method that finds a local maximum of  $L_{tot}$  such as the Baum-Welch or Expectation Maximization (EM) algorithm.

#### Model Evaluation Problem

Given the observation sequence,  $O = O_1, O_2, \dots, O_T$ , and the model  $\lambda = (A, B, \pi)$ , the model evaluation problem is to compute the probability  $P(O|\lambda)$ , that this observed sequence is produced by the model. In other words, given a set of models and a sequence of observations, we want to know how to choose the model which best matches the observations for the purpose of recognition [70].

We calculate the probability  $P(O|\lambda)$  using simple probabilistic arguments. But this calculation involves a number of operations in the order of  $N^T$  for  $N$  states and  $T$  observations. This will be very large, even if the length is of considerably low complexity,  $T$  is moderate. An efficient algorithm for this problem is the forward algorithm that has a considerably low complexity and makes use of an auxiliary variable,  $\alpha_t(i)$  called the forward variable.

The forward variable is defined as the probability of the partial observation sequence  $O = O_1, O_2, \dots, O_T$ , when it terminates at state  $i$ . mathematically,

$\alpha_t(i) = p\{o_1, o_2, \dots, o_t, q_t = i|\lambda\}$ . Then it is easy to see that the following recursive relationship holds.

$$\alpha_{t+1}(j) = b_j(o_{t+1}) \sum_{i=1}^N \alpha_t(i) a_{ij}, 1 \leq j \leq N, 1 \leq t \leq T - 1$$

Where,  $\alpha_t(i), 1 \leq j \leq N$ . Using the recursion we can calculate  $\alpha_1(j) = \pi_j b_j(o_1), 1 \leq j \leq N$  and then the required probability is given by,

$$p\{O|\lambda\} = \sum_{i=1}^N \alpha_T(i)$$

The complexity of this method, known as the forward algorithm, is proportional to  $N^2T$ , which is linear to  $T$  whereas the direct calculation mentioned earlier, had an exponential complexity.

## Appendix H: Trained HMMs

The trained HMM for each surface gesture consists of the transition probability matrix A and the observation probability matrix B.

For K=8, the A and B matrix for gesture Roll-X are given in Table 23 and Table 24.

Table 23: State transition matrix for K=8 for gesture Roll-X

| A | 1        | 2        | 3        | 4        | 5        |
|---|----------|----------|----------|----------|----------|
| 1 | 0.02792  | 0.100414 | 0.86777  | 0.000294 | 0.003601 |
| 2 | 0.002146 | 0.784965 | 0.108841 | 2.07E-09 | 0.104048 |
| 3 | 0.001188 | 0.087634 | 0.787256 | 0.064098 | 0.059824 |
| 4 | 0.000157 | 5.76E-08 | 0.050082 | 0.93349  | 0.016271 |
| 5 | 0.036944 | 0.074284 | 0.028584 | 0.020441 | 0.839747 |

Table 24: Observations probability matrix using K=8 for gesture Roll-X

| B | 1        | 2        | 3        | 4        | 5        | 6        | 7        | 8        |
|---|----------|----------|----------|----------|----------|----------|----------|----------|
| 1 | 0.106206 | 1.00E-05 | 0.056794 | 0.03348  | 1.00E-05 | 0.021768 | 0.001747 | 0.78     |
| 2 | 0.411208 | 1.00E-05 | 1.00E-05 | 0.001183 | 1.00E-05 | 0.000343 | 0.586852 | 0.000405 |
| 3 | 5.36E-05 | 1.00E-05 | 0.46263  | 1.00E-05 | 1.00E-05 | 1.00E-05 | 1.00E-05 | 0.53731  |
| 4 | 1.00E-05 | 0.521479 | 1.40E-05 | 1.00E-05 | 0.478507 | 1.00E-05 | 1.00E-05 | 1.00E-05 |
| 5 | 0.011296 | 1.00E-05 | 1.00E-05 | 0.628134 | 1.00E-05 | 0.356762 | 2.43E-05 | 0.003784 |

For K=10, the A and B matrix are for gesture Roll-X given in Table 25 and Table 26.

Table 25: State transition matrix for K=10 for gesture Roll-X

| A | 1        | 2        | 3        | 4        | 5        |
|---|----------|----------|----------|----------|----------|
| 1 | 0.786334 | 0.004019 | 0.105302 | 0.091749 | 0.012597 |
| 2 | 0.001078 | 0.83109  | 0.042441 | 0.002529 | 0.122862 |
| 3 | 0.081933 | 0.040604 | 0.854881 | 0.003651 | 0.018931 |
| 4 | 0.34933  | 0.013949 | 0.027627 | 0.292585 | 0.31651  |
| 5 | 0.0024   | 0.136498 | 0.001831 | 0.075532 | 0.783739 |

Table 26: Observations probability matrix using K=10 for gesture Roll-X

| B | 1        | 2        | 3        | 4        | 5        | 6        | 7        | 8        |
|---|----------|----------|----------|----------|----------|----------|----------|----------|
| 1 | 0.000599 | 1.00E-05 | 1.00E-05 | 0.32293  | 2.18E-05 | 0.521587 | 1.00E-05 | 0.001981 |
| 2 | 1.00E-05 | 0.540255 | 0.299188 | 1.00E-05 | 1.00E-05 | 1.00E-05 | 0.000735 | 0.14433  |
| 3 | 0.373432 | 1.00E-05 | 1.00E-05 | 1.06E-05 | 0.626314 | 0.000231 | 1.00E-05 | 1.00E-05 |
| 4 | 1.22E-05 | 1.16E-05 | 1.69E-05 | 0.009456 | 1.00E-05 | 0.144337 | 0.068551 | 0.081863 |
| 5 | 1.00E-05 | 0.008782 | 0.096768 | 1.00E-05 | 1.00E-05 | 1.07E-05 | 0.256488 | 0.249937 |
|   | 9        | 10       |          |          |          |          |          |          |
| 1 | 0.007961 | 0.144921 |          |          |          |          |          |          |
| 2 | 0.00331  | 0.012182 |          |          |          |          |          |          |
| 3 | 1.00E-05 | 1.00E-05 |          |          |          |          |          |          |
| 4 | 0.04634  | 0.649412 |          |          |          |          |          |          |
| 5 | 0.324169 | 0.063845 |          |          |          |          |          |          |

For K=14, the A and B matrix for gesture Roll-X are given in Table 27 and Table 28.

Table 27: State transition matrix for K=14 for gesture Roll-X

| A | 1          | 2        | 3        | 4       | 5        |
|---|------------|----------|----------|---------|----------|
| 1 | 0.71547671 | 0.089157 | 0.00616  | 0.14372 | 0.04549  |
| 2 | 0.02170044 | 0.926311 | 0.0518   | 0.00019 | 1.23E-28 |
| 3 | 0.0002203  | 0.038921 | 0.89143  | 0.06943 | 1.11E-11 |
| 4 | 0.19828687 | 0.000384 | 0.09273  | 0.53057 | 0.17803  |
| 5 | 0.04932186 | 0.002756 | 7.67E-09 | 0.04637 | 0.90155  |

Table 28: Observations probability matrix using K=14 for gesture Roll-X

| B | 1          | 2         | 3         | 4        | 5        | 6        | 7        | 8         |
|---|------------|-----------|-----------|----------|----------|----------|----------|-----------|
| 1 | 1.00E-05   | 1.00E-05  | 0.01943   | 1.00E-05 | 0.00328  | 0.00011  | 0.02315  | 1.00E-05  |
| 2 | 1.00E-05   | 1.00E-05  | 1.00E-05  | 0.27573  | 1.00E-05 | 1.00E-05 | 0.2433   | 0.1366548 |
| 3 | 1.00E-05   | 0.385303  | 0.00011   | 1.00E-05 | 1.00E-05 | 1.00E-05 | 1.00E-05 | 1.00E-05  |
| 4 | 1.00E-05   | 1.72E-05  | 0.62925   | 1.00E-05 | 0.00156  | 4.92E-05 | 1.00E-05 | 1.00E-05  |
| 5 | 0.32467067 | 1.00E-05  | 0.00044   | 1.00E-05 | 0.31078  | 0.36411  | 1.00E-05 | 1.00E-05  |
|   | 9          | 10        | 11        | 12       | 13       | 14       |          |           |
| 1 | 0.4239643  | 1.00E-05  | 0.0001036 | 0.52422  | 0.005733 | 1.00E-05 |          |           |
| 2 | 1.00E-05   | 0.1910749 | 0.1518947 | 1.00E-05 | 0.00135  | 1.00E-05 |          |           |
| 3 | 1.00E-05   | 1.00E-05  | 0.0102014 | 1.00E-05 | 0.333058 | 0.27132  |          |           |
| 4 | 0.0664021  | 1.00E-05  | 1.00E-05  | 0.302724 | 1.00E-05 | 1.00E-05 |          |           |
| 5 | 1.00E-05   | 1.00E-05  | 1.00E-05  | 1.00E-05 | 1.00E-05 | 1.00E-05 |          |           |

## Appendix I: Matlab Codes

Data collection, filtering and feature segmentation

```
% importing data as 'A'
A=importdata('rollx01.txt');
% convert 'A' to character array 'a' that has rows and columns
a = char(A);
% converting 'a' to an array of single row
b=reshape(a,1,numel(a));
% getting values of ax, ay and az from text file as characters
for i=1:length(b)
    if (b(i)=='x')
        ax(i)=b(i+1);
    end
    if (b(i)=='y')
        ay(i)=b(i+1);
    end
    if (b(i)=='z')
        az(i)=b(i+1);
    end
end
% converting ax ay az from character to decimal values
for j=1:length(ax)
    axx(j)=unicode2native(ax(j));
end
for j=1:length(ay)
    ayy(j)=unicode2native(ay(j));
end
for j=1:length(az)
    azz(j)=unicode2native(az(j));
end
% get rid of zeros due to conversion from character to decimal
axx(axx==0)=[];
ayy(ayy==0)=[];
azz(azz==0)=[];
lengthacc=[length(axx) length(ayy) length(azz)]
lengthupto=min(lengthacc-5)
% creating a vector of unfiltered acceleration
for i=1:(lengthupto)
    p(i,1)=axx(i);
    p(i,2)=ayy(i);
    p(i,3)=azz(i);
end
% moving average filter implementation
for m = 5:length(axx)
    axxfilt(m) = .2*axx(m) + .2*axx(m-1)+.2*axx(m-2)+.2*axx(m-3)+.2*axx(m-4);
end
for m = 5:length(ayy)
    ayyfilt(m) = .2*ayy(m) + .2*ayy(m-1)+.2*ayy(m-2)+.2*ayy(m-3)+.2*ayy(m-4);
end
for m = 5:length(azz)
    azzfilt(m) = .2*azz(m) + .2*azz(m-1)+.2*azz(m-2)+.2*azz(m-3)+.2*azz(m-4);
end
% removing 0's due to MAF implementation
axxfilt(axxfilt==0)=[]; ayyfilt(ayyfilt==0)=[]; azzfilt(azzfilt==0)=[];
% creating a vector of filtered acceleration
for i=1:(lengthupto)
    n(i,1)=axxfilt(i);
```

```

n(i,2)=ayyfilt(i);
n(i,3)=azzfilt(i);
end
% K-Mean clustering using K-Mean function
k=14;
m=double(n);
[idx,centroid,counter]=MyKMeans(m,k,1);
IDX=idx';
% plotting unfiltered accelerations
figure(1)
plot(axx,'r','LineWidth',1.3)
hold on
plot(ayy,'g','LineWidth',1.3)
hold on
plot(azz,'b','LineWidth',1.3)
grid on;
Title('Accelerometer Data Unfiltered')
xlabel('No of Samples')
ylabel('ax:red ay:green az:blue')
set(gca,'YTick',0:20:260);
set(gca,'XTick',0:40:length(ax));
% plotting filtered accelerations using 5-point MAF
figure(2)
plot(axxfilt,'r','LineWidth',1.3)
hold on
plot(ayyfilt,'g','LineWidth',1.3)
hold on
plot(azzfilt,'b','LineWidth',1.3)
grid on;
Title('Accelerometer Data Filtered using 5-point MAF')
xlabel('No of Samples')
ylabel('ax:red ay:green az:blue')
set(gca,'YTick',0:20:260);
set(gca,'XTick',0:40:length(ax));
% plotting codebook using K=14
figure(3)
STAIRS(IDX,'r','LineWidth',1.5)
grid on;
Title('Code Book For K=14')
xlabel('No of Samples')
ylabel('K Value')
axis([0 length(IDX) 0 15]);
set(gca,'YTick',0:1:15);
set(gca,'XTick',0:10:length(IDX));
% plotting k-clusters (code book) and corresponding data points
figure(4)
% red
h1=plot3(m(idx==1,1),m(idx==1,2),m(idx==1,3),'.','Color',[1 0 0],'MarkerSize',12)
hold on
% blue
h2=plot3(m(idx==2,1),m(idx==2,2),m(idx==2,3),'.','Color',[0 0 1],'MarkerSize',12)
hold on
% green
h3=plot3(m(idx==3,1),m(idx==3,2),m(idx==3,3),'.','Color',[0 1 0],'MarkerSize',12)
hold on
% yellow
h4=plot3(m(idx==4,1),m(idx==4,2),m(idx==4,3),'.','Color',[1 1 0],'MarkerSize',12)
hold on
% magenta

```

```

h5=plot3(m(idx==5,1),m(idx==5,2),m(idx==5,3),'.','Color',[1 0 1],'MarkerSize',12)
hold on
% cyan
h6=plot3(m(idx==6,1),m(idx==6,2),m(idx==6,3),'.','Color',[0 1 1],'MarkerSize',12)
hold on
% black
h7=plot3(m(idx==7,1),m(idx==7,2),m(idx==7,3),'.','Color',[0 0 0],'MarkerSize',12)
hold on
% gold
h8=plot3(m(idx==8,1),m(idx==8,2),m(idx==8,3),'.','Color',[0.9 0.75 0],'MarkerSize',12)
hold on
% orange
h9=plot3(m(idx==9,1),m(idx==9,2),m(idx==9,3),'.','Color',[1 0.6 0],'MarkerSize',12)
hold on
% brown
h10=plot3(m(idx==10,1),m(idx==10,2),m(idx==10,3),'.','Color',[0.8 0.5 0],'MarkerSize',12)
hold on
% purple
h11=plot3(m(idx==11,1),m(idx==11,2),m(idx==11,3),'.','Color',[0.5 0 0.9],'MarkerSize',12)
hold on
% peach
h12=plot3(m(idx==12,1),m(idx==12,2),m(idx==12,3),'.','Color',[1 0.5 0.5],'MarkerSize',12)
hold on
% avocado
h13=plot3(m(idx==13,1),m(idx==13,2),m(idx==13,3),'.','Color',[0.3 0.8 0],'MarkerSize',12)
hold on
% white
h14=plot3(m(idx==14,1),m(idx==14,2),m(idx==14,3),'.','Color',[1 1 1],'MarkerSize',12)
hold on
% red
j1=plot3(centroid(1,1),centroid(1,2),centroid(1,3),'x','Color',[1 0 0],'MarkerSize',12,'LineWidth',2)
hold on
% blue
j2=plot3(centroid(2,1),centroid(2,2),centroid(2,3),'x','Color',[0 0 1],'MarkerSize',12,'LineWidth',2)
hold on
% green
j3=plot3(centroid(3,1),centroid(3,2),centroid(3,3),'x','Color',[0 1 0],'MarkerSize',12,'LineWidth',2)
hold on
% yellow
j4=plot3(centroid(4,1),centroid(4,2),centroid(4,3),'x','Color',[1 1 0],'MarkerSize',12,'LineWidth',2)
hold on
% magenta
j5=plot3(centroid(5,1),centroid(5,2),centroid(5,3),'x','Color',[1 0 1],'MarkerSize',12,'LineWidth',2)
hold on
% cyan
j6=plot3(centroid(6,1),centroid(6,2),centroid(6,3),'x','Color',[0 1 1],'MarkerSize',12,'LineWidth',2)
hold on
% black
j7=plot3(centroid(7,1),centroid(7,2),centroid(7,3),'x','Color',[0 0 0],'MarkerSize',12,'LineWidth',2)
hold on
% gold
j8=plot3(centroid(8,1),centroid(8,2),centroid(8,3),'x','Color',[0.9 0.75 0],'MarkerSize',12,'LineWidth',2)
hold on
% orange
j9=plot3(centroid(9,1),centroid(9,2),centroid(9,3),'x','Color',[1 0.6 0],'MarkerSize',12,'LineWidth',2)
hold on
% brown
j10=plot3(centroid(10,1),centroid(10,2),centroid(10,3),'x','Color',[0.8 0.5
0],'MarkerSize',12,'LineWidth',2)

```

```

hold on
% purple
j11=plot3(centroid(11,1),centroid(11,2),centroid(11,3),'x','Color',[0.5 0
0.9],'MarkerSize',12,'LineWidth',2)
hold on
% peach
j12=plot3(centroid(12,1),centroid(12,2),centroid(12,3),'x','Color',[1 0.5
0.5],'MarkerSize',12,'LineWidth',2)
hold on
% avocado
j13=plot3(centroid(13,1),centroid(13,2),centroid(13,3),'x','Color',[0.3 0.8
0],'MarkerSize',12,'LineWidth',2)
hold on
% white
j14=plot3(centroid(14,1),centroid(14,2),centroid(14,3),'x','Color',[1 1 1],'MarkerSize',12,'LineWidth',2)
hold on
legend([j1 j2 j3 j4 j5 j6 j7 j8 j9 j10 j11 j12 j13 j14],{'1','2','3','4','5','6','7','8','9','10','11','12','13','14'});
set(legend,'Color',[0.8 0.8 0.8]);
% light grey
set(gca,'Color',[0.8 0.8 0.8]);
hold on
set(gcf,'Color',[0.8 0.8 0.8]);
xlabel('X-Axis'); ylabel('Y-Axis'); zlabel('Z-Axis'); Title('Clusters For K=14'); legend('show');

Baum Welch training and evaluation using forward algorithm

% HMM for class 'rollx' and random initialization of parameters
hmmrollx.prior = [1 0 0 0 0];
hmmrollx.transmat = rand(5,5); % 5 by 5 transition matrix
hmmrollx.transmat = mk_stochastic(hmmrollx.transmat);
hmmrollx.obsmat = rand(5, 10); % # of states * # of observation
hmmrollx.obsmat = mk_stochastic(hmmrollx.obsmat);
% training of HMM model rollx (Baum-Welch algorithm)
[LLrollx, hmmrollx.prior, hmmrollx.transmat, hmmrollx.obsmat] = dhmm_em(rollx, hmmrollx.prior,
hmmrollx.transmat, hmmrollx.obsmat);
% smoothing of HMM observation parameter: set floor value 1.0e-5
hmmrollx.obsmat = max(hmmrollx.obsmat, 1.0e-5);
% HMM for class 'rubx' and random initialization of parameters
hmmrubx.prior = [1 0 0 0 0];
hmmrubx.transmat = rand(5,5); % 5 by 5 transition matrix
hmmrubx.transmat = mk_stochastic(hmmrubx.transmat);
hmmrubx.obsmat = rand(5, 10); % # of states * # of observation
hmmrubx.obsmat = mk_stochastic(hmmrubx.obsmat);
% training of HMM model rubx (Baum-Welch algorithm)
[LLrubx, hmmrubx.prior, hmmrubx.transmat, hmmrubx.obsmat] = dhmm_em(rubx, hmmrubx.prior,
hmmrubx.transmat, hmmrubx.obsmat);
% smoothing of HMM observation parameter: set floor value 1.0e-5
hmmrubx.obsmat = max(hmmrubx.obsmat, 1.0e-5);
% HMM for class 'ruby' and random initialization of parameters
hmmruby.prior = [1 0 0 0 0];
hmmruby.transmat = rand(5,5); % 5 by 5 transition matrix
hmmruby.transmat = mk_stochastic(hmmruby.transmat);
hmmruby.obsmat = rand(5, 10); % # of states * # of observation
hmmruby.obsmat = mk_stochastic(hmmruby.obsmat);
% training of HMM model ruby (Baum-Welch algorithm)
[LLruby, hmmruby.prior, hmmruby.transmat, hmmruby.obsmat] = dhmm_em(ruby, hmmruby.prior,
hmmruby.transmat, hmmruby.obsmat);
% smoothing of HMM observation parameter: set floor value 1.0e-5
hmmruby.obsmat = max(hmmruby.obsmat, 1.0e-5);
% HMM for class 'tap1' and random initialization of parameters

```

```

hmmtap1.prior = [1 0 0 0 0];
hmmtap1.transmat = rand(5,5); % 5 by 5 transition matrix
hmmtap1.transmat = mk_stochastic(hmmtap1.transmat);
hmmtap1.obsmat = rand(5, 10); % # of states * # of observation
hmmtap1.obsmat = mk_stochastic(hmmtap1.obsmat);
% training of HMM model tap1 (Baum-Welch algorithm)
[LLtap1, hmmtap1.prior, hmmtap1.transmat, hmmtap1.obsmat] = dhmm_em(tap1, hmmtap1.prior,
hmmtap1.transmat, hmmtap1.obsmat);
% smoothing of HMM observation parameter: set floor value 1.0e-5
hmmtap1.obsmat = max(hmmtap1.obsmat, 1.0e-5);
% HMM for class 'tap2' and random initialization of parameters
hmmtap2.prior = [1 0 0 0 0];
hmmtap2.transmat = rand(5,5); % 5 by 5 transition matrix
hmmtap2.transmat = mk_stochastic(hmmtap2.transmat);
hmmtap2.obsmat = rand(5, 10); % # of states * # of observation
hmmtap2.obsmat = mk_stochastic(hmmtap2.obsmat);
% training of HMM model tap2 (Baum-Welch algorithm)
[LLtap2, hmmtap2.prior, hmmtap2.transmat, hmmtap2.obsmat] = dhmm_em(tap2, hmmtap2.prior,
hmmtap2.transmat, hmmtap2.obsmat);
% smoothing of HMM observation parameter: set floor value 1.0e-5
hmmtap2.obsmat = max(hmmtap2.obsmat, 1.0e-5);
% HMM for class 'tap3' and random initialization of parameters
hmmtap3.prior = [1 0 0 0 0];
hmmtap3.transmat = rand(5,5); % 5 by 5 transition matrix
hmmtap3.transmat = mk_stochastic(hmmtap3.transmat);
hmmtap3.obsmat = rand(5, 10); % # of states * # of observation
hmmtap3.obsmat = mk_stochastic(hmmtap3.obsmat);
% training of HMM model tap3 (Baum-Welch algorithm)
[LLtap3, hmmtap3.prior, hmmtap3.transmat, hmmtap3.obsmat] = dhmm_em(tap3, hmmtap3.prior,
hmmtap3.transmat, hmmtap3.obsmat);
% smoothing of HMM observation parameter: set floor value 1.0e-5
hmmtap3.obsmat = max(hmmtap3.obsmat, 1.0e-5);
% HMM for class 'tap4' and random initialization of parameters
hmmtap4.prior = [1 0 0 0 0];
hmmtap4.transmat = rand(5,5); % 5 by 5 transition matrix
hmmtap4.transmat = mk_stochastic(hmmtap4.transmat);
hmmtap4.obsmat = rand(5, 10); % # of states * # of observation
hmmtap4.obsmat = mk_stochastic(hmmtap4.obsmat);
% training of HMM model tap4 (Baum-Welch algorithm)
[LLtap4, hmmtap4.prior, hmmtap4.transmat, hmmtap4.obsmat] = dhmm_em(tap4, hmmtap4.prior,
hmmtap4.transmat, hmmtap4.obsmat);
% smoothing of HMM observation parameter: set floor value 1.0e-5
hmmtap4.obsmat = max(hmmtap4.obsmat, 1.0e-5);
% HMM for class 'tap5' and random initialization of parameters
hmmtap5.prior = [1 0 0 0 0];
hmmtap5.transmat = rand(5,5); % 5 by 5 transition matrix
hmmtap5.transmat = mk_stochastic(hmmtap5.transmat);
hmmtap5.obsmat = rand(5, 14); % # of states * # of observation
hmmtap5.obsmat = mk_stochastic(hmmtap5.obsmat);
% training of HMM model tap5 (Baum-Welch algorithm)
[LLtap5, hmmtap5.prior, hmmtap5.transmat, hmmtap5.obsmat] = dhmm_em(tap5, hmmtap5.prior,
hmmtap5.transmat, hmmtap5.obsmat);
% smoothing of HMM observation parameter: set floor value 1.0e-5
hmmtap5.obsmat = max(hmmtap5.obsmat, 1.0e-5);
rollxcount=0; rubxcount=0; rubycount=0; tap1count=0; tap2count=0; tap3count=0; tap4count=0;
tap5count=0;
% evaluation of class 'rollx'
for dt =1:length(rollxt)
loglikerollx = dhmm_logprob(rollxt{dt}, hmmtap1.prior, hmmtap1.transmat, hmmtap1.obsmat);

```

```

loglikerubx = dhmm_logprob(rollxt{dt}, hmmrubx.prior, hmmrubx.transmat, hmmrubx.obsamat);
loglikeruby = dhmm_logprob(rollxt{dt}, hmmruby.prior, hmmruby.transmat, hmmruby.obsamat);
logliketap1 = dhmm_logprob(rollxt{dt}, hmmtap1.prior, hmmtap1.transmat, hmmtap1.obsamat);
logliketap2 = dhmm_logprob(rollxt{dt}, hmmtap2.prior, hmmtap2.transmat, hmmtap2.obsamat);
logliketap3 = dhmm_logprob(rollxt{dt}, hmmtap3.prior, hmmtap3.transmat, hmmtap3.obsamat);
logliketap4 = dhmm_logprob(rollxt{dt}, hmmtap4.prior, hmmtap4.transmat, hmmtap4.obsamat);
logliketap5 = dhmm_logprob(rollxt{dt}, hmmtap5.prior, hmmtap5.transmat, hmmtap5.obsamat);
maxloglile=max([loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5]);
    if (maxloglile==loglikerollx)
        rollxcount=rollxcount+1;
    end
    if (maxloglile==loglikerubx)
        rubxcount=rubxcount+1;
    end
    if (maxloglile==loglikeruby)
        rubycount=rubycount+1;
    end
    if (maxloglile==logliketap1)
        tap1count=tap1count+1;
    end
    if (maxloglile==logliketap2)
        tap2count=tap2count+1;
    end
    if (maxloglile==logliketap3)
        tap3count=tap3count+1;
    end
    if (maxloglile==logliketap4)
        tap4count=tap4count+1;
    end
    if (maxloglile==logliketap5)
        tap5count=tap5count+1;
    end
disp(sprintf(['class rollx: %02d-th data] model rollx: %.2f, model rubx: %.2f, model ruby: %.2f, model tap1: %.2f, model tap2: %.2f, model tap3: %.2f, model tap4: %.2f, model tap5: %.2f',dt,loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5]));
end
totalcount_rollx=rollxcount+rubxcount+rubycount+tap1count+tap2count+tap3count+tap4count+tap5count;
correct_rollx=rollxcount;
wrong_rollx=rubxcount+rubycount+tap1count+tap2count+tap3count+tap4count+tap5count;
percent_correct_rollx=(correct_rollx*100)/totalcount_rollx;
percent_wrong_rollx=(wrong_rollx*100)/totalcount_rollx;
disp(sprintf('Model rollx Classification: True: %02d False: %02d TPR: %.1f',correct_rollx,wrong_rollx,percent_correct_rollx));
rollxcount1=rollxcount; rubxcount1=rubxcount; rubycount1=rubycount; tap1count1=tap1count; tap2count1=tap2count; tap3count1=tap3count; tap4count1=tap4count; tap5count1=tap5count;
rollxcount=0; rubxcount=0; rubycount=0; tap1count=0; tap2count=0; tap3count=0; tap4count=0; tap5count=0;
% evaluation of class 'rubx'
for dt = 1:length(rubxt)
loglikerollx = dhmm_logprob(rubxt{dt}, hmmrollx.prior, hmmrollx.transmat, hmmrollx.obsamat);
loglikerubx = dhmm_logprob(rubxt{dt}, hmmrubx.prior, hmmrubx.transmat, hmmrubx.obsamat);
loglikeruby = dhmm_logprob(rubxt{dt}, hmmruby.prior, hmmruby.transmat, hmmruby.obsamat);
logliketap1 = dhmm_logprob(rubxt{dt}, hmmtap1.prior, hmmtap1.transmat, hmmtap1.obsamat);
logliketap2 = dhmm_logprob(rubxt{dt}, hmmtap2.prior, hmmtap2.transmat, hmmtap2.obsamat);
logliketap3 = dhmm_logprob(rubxt{dt}, hmmtap3.prior, hmmtap3.transmat, hmmtap3.obsamat);
logliketap4 = dhmm_logprob(rubxt{dt}, hmmtap4.prior, hmmtap4.transmat, hmmtap4.obsamat);

```

```

logliketap5 = dhmm_logprob(rubxt{dt}, hmmtap5.prior, hmmtap5.transmat, hmmtap5.obsmat);
maxloglile=max([loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5]);
    if (maxloglile==loglikerollx)
        rollxcount=rollxcount+1;
    end
    if (maxloglile==loglikerubx)
        rubxcount=rubxcount+1;
    end
    if (maxloglile==loglikeruby)
        rubycount=rubycount+1;
    end
    if (maxloglile==logliketap1)
        tap1count=tap1count+1;
    end
    if (maxloglile==logliketap2)
        tap2count=tap2count+1;
    end
    if (maxloglile==logliketap3)
        tap3count=tap3count+1;
    end
    if (maxloglile==logliketap4)
        tap4count=tap4count+1;
    end
    if (maxloglile==logliketap5)
        tap5count=tap5count+1;
    end
disp(sprintf(['class rubx: %02d-th data] model rollx: %.2f, model rubx: %.2f, model ruby: %.2f, model tap1: %.2f, model tap2: %.2f, model tap3: %.2f, model tap4: %.2f, model tap5: %.2f',dt,loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5]));
end
totalcount_rubx=rollxcount+rubxcount+rubycount+tap1count+tap2count+tap3count+tap4count+tap5count;
correct_rubx=rubxcount;
wrong_rubx=rollxcount+rubycount+tap1count+tap2count+tap3count+tap4count+tap5count;
percent_correct_rubx=(correct_rubx*100)/totalcount_rubx;
percent_wrong_rubx=(wrong_rubx*100)/totalcount_rubx;
disp(sprintf('Model rubx Classification: True: %02d False: %02d TPR: %.1f',correct_rubx,wrong_rubx,percent_correct_rubx ));
rollxcount2=rollxcount; rubxcount2=rubxcount; rubycount2=rubycount; tap1count2=tap1count; tap2count2=tap2count; tap3count2=tap3count; tap4count2=tap4count; tap5count2=tap5count;
rollxcount=0; rubxcount=0; rubycount=0; tap1count=0; tap2count=0; tap3count=0; tap4count=0; tap5count=0;
% evaluation of class 'ruby'
for dt =1:length(rubyt)
loglikerollx = dhmm_logprob(rubyt{dt}, hmmrollx.prior, hmmrollx.transmat, hmmrollx.obsmat);
loglikerubx = dhmm_logprob(rubyt{dt}, hmmrubx.prior, hmmrubx.transmat, hmmrubx.obsmat);
loglikeruby = dhmm_logprob(rubyt{dt}, hmmruby.prior, hmmruby.transmat, hmmruby.obsmat);
logliketap1 = dhmm_logprob(rubyt{dt}, hmmtap1.prior, hmmtap1.transmat, hmmtap1.obsmat);
logliketap2 = dhmm_logprob(rubyt{dt}, hmmtap2.prior, hmmtap2.transmat, hmmtap2.obsmat);
logliketap3 = dhmm_logprob(rubyt{dt}, hmmtap3.prior, hmmtap3.transmat, hmmtap3.obsmat);
logliketap4 = dhmm_logprob(rubyt{dt}, hmmtap4.prior, hmmtap4.transmat, hmmtap4.obsmat);
logliketap5 = dhmm_logprob(rubyt{dt}, hmmtap5.prior, hmmtap5.transmat, hmmtap5.obsmat);
maxloglile=max([loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5]);
    if (maxloglile==loglikerollx)
        rollxcount=rollxcount+1;
    end

```

```

if (maxloglile==loglikerubx)
rubxcount=rubxcount+1;
end
if (maxloglile==loglikeruby)
rubycount=rubycount+1;
end
if (maxloglile==logliketap1)
tap1count=tap1count+1;
end
if (maxloglile==logliketap2)
tap2count=tap2count+1;
end
if (maxloglile==logliketap3)
tap3count=tap3count+1;
end
if (maxloglile==logliketap4)
tap4count=tap4count+1;
end
if (maxloglile==logliketap5)
tap5count=tap5count+1;
end
disp(sprintf(['class ruby: %02d-th data] model rollx: %.2f, model rubx: %.2f, model ruby: %.2f, model
tap1: %.2f, model tap2: %.2f, model tap3: %.2f, model tap4: %.2f, model tap5:
%.2f',dt,loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5
));
end
totalcount_ruby=rollxcount+rubxcount+rubycount+tap1count+tap2count+tap3count+tap4count+tap5count;
correct_ruby=rubycount;
wrong_ruby=rollxcount+rubxcount+tap1count+tap2count+tap3count+tap4count+tap5count;
percent_correct_ruby=(correct_ruby*100)/totalcount_ruby;
percent_wrong_ruby=(wrong_ruby*100)/totalcount_ruby;
disp(sprintf('Model ruby Classification: True: %02d False: %02d TPR:
%.1f',correct_ruby,wrong_ruby,percent_correct_ruby ));
rollxcount3=rollxcount; rubxcount3=rubxcount; rubycount3=rubycount; tap1count3=tap1count;
tap2count3=tap2count; tap3count3=tap3count; tap4count3=tap4count; tap5count3=tap5count;
rollxcount=0; rubxcount=0; rubycount=0; tap1count=0; tap2count=0; tap3count=0; tap4count=0;
tap5count=0;
% evaluation of class 'tap1'
for dt =1:length(tap1t)
loglikerollx = dhmm_logprob(tap1t{dt}, hmmrollx.prior, hmmrollx.transmat, hmmrollx.obsmat);
loglikerubx = dhmm_logprob(tap1t{dt}, hmmrubx.prior, hmmrubx.transmat, hmmrubx.obsmat);
loglikeruby = dhmm_logprob(tap1t{dt}, hmmruby.prior, hmmruby.transmat, hmmruby.obsmat);
logliketap1 = dhmm_logprob(tap1t{dt}, hmmtap1.prior, hmmtap1.transmat, hmmtap1.obsmat);
logliketap2 = dhmm_logprob(tap1t{dt}, hmmtap2.prior, hmmtap2.transmat, hmmtap2.obsmat);
logliketap3 = dhmm_logprob(tap1t{dt}, hmmtap3.prior, hmmtap3.transmat, hmmtap3.obsmat);
logliketap4 = dhmm_logprob(tap1t{dt}, hmmtap4.prior, hmmtap4.transmat, hmmtap4.obsmat);
logliketap5 = dhmm_logprob(tap1t{dt}, hmmtap5.prior, hmmtap5.transmat, hmmtap5.obsmat);
maxloglile=max([loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5]);
if (maxloglile==loglikerollx)
rollxcount=rollxcount+1;
end
if (maxloglile==loglikerubx)
rubxcount=rubxcount+1;
end
if (maxloglile==loglikeruby)
rubycount=rubycount+1;
end
end

```

```

if (maxloglile==logliketap1)
tap1count=tap1count+1;
end
if (maxloglile==logliketap2)
tap2count=tap2count+1;
end
if (maxloglile==logliketap3)
tap3count=tap3count+1;
end
if (maxloglile==logliketap4)
tap4count=tap4count+1;
end
if (maxloglile==logliketap5)
tap5count=tap5count+1;
end
disp(sprintf(['class tap1: %02d-th data] model rollx: %.2f, model rubx: %.2f, model ruby: %.2f, model
tap1: %.2f, model tap2: %.2f, model tap3: %.2f, model tap4: %.2f, model tap5:
%.2f,dt,loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5
));
end
totalcount_tap1=rollxcount+rubxcount+rubycount+tap1count+tap2count+tap3count+tap4count+tap5count;
correct_tap1=tap1count;
wrong_tap1=rollxcount+rubxcount+rubycount+tap2count+tap3count+tap4count+tap5count;
percent_correct_tap1=(correct_tap1*100)/totalcount_tap1;
percent_wrong_tap1=(wrong_tap1*100)/totalcount_tap1;
disp(sprintf('Model tap1 Classification: True: %02d False: %02d TPR:
%.1f,correct_tap1,wrong_tap1,percent_correct_tap1 ));
rollxcount4=rollxcount; rubxcount4=rubxcount; rubycount4=rubycount; tap1count4=tap1count;
tap2count4=tap2count; tap3count4=tap3count; tap4count4=tap4count; tap5count4=tap5count;
rollxcount=0; rubxcount=0; rubycount=0; tap1count=0; tap2count=0; tap3count=0; tap4count=0;
tap5count=0;
% evaluation of class 'tap2'
for dt =1:length(tap2t)
loglikerollx = dhmm_logprob(tap2t{dt}, hmmrollx.prior, hmmrollx.transmat, hmmrollx.obsmat);
loglikerubx = dhmm_logprob(tap2t{dt}, hmnrubx.prior, hmnrubx.transmat, hmnrubx.obsmat);
loglikeruby = dhmm_logprob(tap2t{dt}, hmnruby.prior, hmnruby.transmat, hmnruby.obsmat);
logliketap1 = dhmm_logprob(tap2t{dt}, hmmtap1.prior, hmmtap1.transmat, hmmtap1.obsmat);
logliketap2 = dhmm_logprob(tap2t{dt}, hmmtap2.prior, hmmtap2.transmat, hmmtap2.obsmat);
logliketap3 = dhmm_logprob(tap2t{dt}, hmmtap3.prior, hmmtap3.transmat, hmmtap3.obsmat);
logliketap4 = dhmm_logprob(tap2t{dt}, hmmtap4.prior, hmmtap4.transmat, hmmtap4.obsmat);
logliketap5 = dhmm_logprob(tap2t{dt}, hmmtap5.prior, hmmtap5.transmat, hmmtap5.obsmat);
maxloglile=max([loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5]);
if (maxloglile==loglikerollx)
rollxcount=rollxcount+1;
end
if (maxloglile==loglikerubx)
rubxcount=rubxcount+1;
end
if (maxloglile==loglikeruby)
rubycount=rubycount+1;
end
if (maxloglile==logliketap1)
tap1count=tap1count+1;
end
if (maxloglile==logliketap2)
tap2count=tap2count+1;
end

```

```

if (maxloglile==logliketap3)
tap3count=tap3count+1;
end
if (maxloglile==logliketap4)
tap4count=tap4count+1;
end
if (maxloglile==logliketap5)
tap5count=tap5count+1;
end
disp(sprintf(['class tap2: %02d-th data] model rollx: %.2f, model rubx: %.2f, model ruby: %.2f, model
tap1: %.2f, model tap2: %.2f, model tap3: %.2f, model tap4: %.2f, model tap5:
%.2f',dt,loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5
));
end
totalcount_tap2=rollxcount+rubxcount+rubycount+tap1count+tap2count+tap3count+tap4count+tap5co
unt;
correct_tap2=tap2count;
wrong_tap2=rollxcount+rubxcount+rubycount+tap1count+tap3count+tap4count+tap5count;
percent_correct_tap2=(correct_tap2*100)/totalcount_tap2;
percent_wrong_tap2=(wrong_tap2*100)/totalcount_tap2;
disp(sprintf('Model tap2 Classification: True: %02d False: %02d TPR:
%.1f',correct_tap2,wrong_tap2,percent_correct_tap2 ));
rollxcount5=rollxcount; rubxcount5=rubxcount; rubycount5=rubycount; tap1count5=tap1count;
tap2count5=tap2count; tap3count5=tap3count; tap4count5=tap4count; tap5count5=tap5count;
rollxcount=0; rubxcount=0; rubycount=0; tap1count=0; tap2count=0; tap3count=0; tap4count=0;
tap5count=0;
% evaluation of class 'tap3'
for dt =1:length(tap3t)
loglikerollx = dhmm_logprob(tap3t{dt}, hmmrollx.prior, hmmrollx.transmat, hmmrollx.obsmat);
loglikerubx = dhmm_logprob(tap3t{dt}, hmmrubx.prior, hmmrubx.transmat, hmmrubx.obsmat);
loglikeruby = dhmm_logprob(tap3t{dt}, hmmruby.prior, hmmruby.transmat, hmmruby.obsmat);
logliketap1 = dhmm_logprob(tap3t{dt}, hmmtap1.prior, hmmtap1.transmat, hmmtap1.obsmat);
logliketap2 = dhmm_logprob(tap3t{dt}, hmmtap2.prior, hmmtap2.transmat, hmmtap2.obsmat);
logliketap3 = dhmm_logprob(tap3t{dt}, hmmtap3.prior, hmmtap3.transmat, hmmtap3.obsmat);
logliketap4 = dhmm_logprob(tap3t{dt}, hmmtap4.prior, hmmtap4.transmat, hmmtap4.obsmat);
logliketap5 = dhmm_logprob(tap3t{dt}, hmmtap5.prior, hmmtap5.transmat, hmmtap5.obsmat);
maxloglile=max([loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,l
ogliketap5]);
if (maxloglile==loglikerollx)
rollxcount=rollxcount+1;
end
if (maxloglile==loglikerubx)
rubxcount=rubxcount+1;
end
if (maxloglile==loglikeruby)
rubycount=rubycount+1;
end
if (maxloglile==logliketap1)
tap1count=tap1count+1;
end
if (maxloglile==logliketap2)
tap2count=tap2count+1;
end
if (maxloglile==logliketap3)
tap3count=tap3count+1;
end
if (maxloglile==logliketap4)
tap4count=tap4count+1;
end
end

```

```

    if (maxloglile==logliketap5)
        tap5count=tap5count+1;
    end
disp(sprintf(['class tap3: %02d-th data] model rollx: %.2f, model rubx: %.2f, model ruby: %.2f, model
tap1: %.2f, model tap2: %.2f, model tap3: %.2f, model tap4: %.2f, model tap5:
%.2f',dt,loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5
));
end
totalcount_tap3=rollxcount+rubxcount+rubycount+tap1count+tap2count+tap3count+tap4count+tap5count;
correct_tap3=tap3count;
wrong_tap3=rollxcount+rubxcount+rubycount+tap1count+tap2count+tap4count+tap5count;
percent_correct_tap3=(correct_tap3*100)/totalcount_tap3;
percent_wrong_tap3=(wrong_tap3*100)/totalcount_tap3;
disp(sprintf('Model tap3 Classification: True: %02d False: %02d TPR:
%.1f',correct_tap3,percent_correct_tap3));
rollxcount6=rollxcount; rubxcount6=rubxcount; rubycount6=rubycount; tap1count6=tap1count;
tap2count6=tap2count; tap3count6=tap3count; tap4count6=tap4count; tap5count6=tap5count;
rollxcount=0; rubxcount=0; rubycount=0; tap1count=0; tap2count=0; tap3count=0; tap4count=0;
tap5count=0;
% evaluation of class 'tap4'
for dt =1:length(tap4t)
loglikerollx = dhmm_logprob(tap4t{dt}, hmmrollx.prior, hmmrollx.transmat, hmmrollx.obsmat);
loglikerubx = dhmm_logprob(tap4t{dt}, hmmrubx.prior, hmmrubx.transmat, hmmrubx.obsmat);
loglikeruby = dhmm_logprob(tap4t{dt}, hmmruby.prior, hmmruby.transmat, hmmruby.obsmat);
logliketap1 = dhmm_logprob(tap4t{dt}, hmmtap1.prior, hmmtap1.transmat, hmmtap1.obsmat);
logliketap2 = dhmm_logprob(tap4t{dt}, hmmtap2.prior, hmmtap2.transmat, hmmtap2.obsmat);
logliketap3 = dhmm_logprob(tap4t{dt}, hmmtap3.prior, hmmtap3.transmat, hmmtap3.obsmat);
logliketap4 = dhmm_logprob(tap4t{dt}, hmmtap4.prior, hmmtap4.transmat, hmmtap4.obsmat);
logliketap5 = dhmm_logprob(tap4t{dt}, hmmtap5.prior, hmmtap5.transmat, hmmtap5.obsmat);
maxloglile=max([loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5]);
    if (maxloglile==loglikerollx)
        rollxcount=rollxcount+1;
    end
    if (maxloglile==loglikerubx)
        rubxcount=rubxcount+1;
    end
    if (maxloglile==loglikeruby)
        rubycount=rubycount+1;
    end
    if (maxloglile==logliketap1)
        tap1count=tap1count+1;
    end
    if (maxloglile==logliketap2)
        tap2count=tap2count+1;
    end
    if (maxloglile==logliketap3)
        tap3count=tap3count+1;
    end
    if (maxloglile==logliketap4)
        tap4count=tap4count+1;
    end
    if (maxloglile==logliketap5)
        tap5count=tap5count+1;
    end
disp(sprintf(['class tap4: %02d-th data] model rollx: %.2f, model rubx: %.2f, model ruby: %.2f, model
tap1: %.2f, model tap2: %.2f, model tap3: %.2f, model tap4: %.2f, model tap5:
%.2f',dt,loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5
));
end

```

```

%.2f,dt,loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5
));
end
totalcount_tap4=rollxcount+rubxcount+rubycount+tap1count+tap2count+tap3count+tap4count+tap5count;
correct_tap4=tap4count;
wrong_tap4=rollxcount+rubxcount+rubycount+tap1count+tap2count+tap3count+tap5count;
percent_correct_tap4=(correct_tap4*100)/totalcount_tap4;
percent_wrong_tap4=(wrong_tap4*100)/totalcount_tap4;
disp(sprintf('Model tap4 Classification: True: %02d False: %02d TPR:
%.1f,correct_tap4,wrong_tap4,percent_correct_tap4));
rollxcount7=rollxcount; rubxcount7=rubxcount; rubycount7=rubycount; tap1count7=tap1count;
tap2count7=tap2count; tap3count7=tap3count; tap4count7=tap4count; tap5count7=tap5count;
rollxcount=0; rubxcount=0; rubycount=0; tap1count=0; tap2count=0; tap3count=0; tap4count=0;
tap5count=0;
% evaluation of class 'tap5'
for dt =1:length(tap5t)
loglikerollx = dhmm_logprob(tap5t{dt}, hmmrollx.prior, hmmrollx.transmat, hmmrollx.obsmat);
loglikerubx = dhmm_logprob(tap5t{dt}, hmmrubx.prior, hmmrubx.transmat, hmmrubx.obsmat);
loglikeruby = dhmm_logprob(tap5t{dt}, hmmruby.prior, hmmruby.transmat, hmmruby.obsmat);
logliketap1 = dhmm_logprob(tap5t{dt}, hmmtap1.prior, hmmtap1.transmat, hmmtap1.obsmat);
logliketap2 = dhmm_logprob(tap5t{dt}, hmmtap2.prior, hmmtap2.transmat, hmmtap2.obsmat);
logliketap3 = dhmm_logprob(tap5t{dt}, hmmtap3.prior, hmmtap3.transmat, hmmtap3.obsmat);
logliketap4 = dhmm_logprob(tap5t{dt}, hmmtap4.prior, hmmtap4.transmat, hmmtap4.obsmat);
logliketap5 = dhmm_logprob(tap5t{dt}, hmmtap5.prior, hmmtap5.transmat, hmmtap5.obsmat);
maxloglile=max([loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5]);
    if (maxloglile==loglikerollx)
        rollxcount=rollxcount+1;
    end
    if (maxloglile==loglikerubx)
        rubxcount=rubxcount+1;
    end
    if (maxloglile==loglikeruby)
        rubycount=rubycount+1;
    end
    if (maxloglile==logliketap1)
        tap1count=tap1count+1;
    end
    if (maxloglile==logliketap2)
        tap2count=tap2count+1;
    end
    if (maxloglile==logliketap3)
        tap3count=tap3count+1;
    end
    if (maxloglile==logliketap4)
        tap4count=tap4count+1;
    end
    if (maxloglile==logliketap5)
        tap5count=tap5count+1;
    end
disp(sprintf('[class tap5: %02d-th data] model rollx: %.2f, model rubx: %.2f, model ruby: %.2f, model
tap1: %.2f, model tap2: %.2f, model tap3: %.2f, model tap4: %.2f, model tap5:
%.2f,dt,loglikerollx,loglikerubx,loglikeruby,logliketap1,logliketap2,logliketap3,logliketap4,logliketap5
));
end
totalcount_tap5=rollxcount+rubxcount+rubycount+tap1count+tap2count+tap3count+tap4count+tap5count;
correct_tap5=tap5count;

```

```

wrong_tap5=rollxcount+rubxcount+rubycount+tap1count+tap2count+tap3count+tap4count;
percent_correct_tap5=(correct_tap5*100)/totalcount_tap5;
percent_wrong_tap5=(wrong_tap5*100)/totalcount_tap5;
disp(sprintf('Model tap5 Classification: True: %02d False: %02d TPR:
%.1f',correct_tap5,wrong_tap5,percent_correct_tap5 ));
rollxcount8=rollxcount;
rubxcount8=rubxcount; rubycount8=rubycount; tap1count8=tap1count; tap2count8=tap2count;
tap3count8=tap3count; tap4count8=tap4count; tap5count8=tap5count;
rollxcount=0; rubxcount=0; rubycount=0; tap1count=0; tap2count=0; tap3count=0; tap4count=0;
tap5count=0;
correct=correct_rollx+correct_rubx+correct_ruby+correct_tap1+correct_tap2+correct_tap3+correct_tap4+correct_tap5;
wrong=wrong_rollx+wrong_rubx+wrong_ruby+wrong_tap1+wrong_tap2+wrong_tap3+wrong_tap4+wrong_tap5;
totalcount=totalcount_rollx+totalcount_rubx+totalcount_ruby+totalcount_tap1+totalcount_tap2+totalcount_tap3+totalcount_tap4+totalcount_tap5;
percent_correct=(correct*100)/totalcount;
percent_wrong=(wrong*100)/totalcount;
% print confusion matrix
disp(sprintf('Overall Classification: True: %02d False: %02d TPR:
%.1f',correct,wrong,percent_correct));
disp(sprintf('Class rollx : True: %02d False: %02d TPR:
%.1f',correct_rollx,wrong_rollx,percent_correct_rollx));
disp(sprintf('Class rubx : True: %02d False: %02d TPR:
%.1f',correct_rubx,wrong_rubx,percent_correct_rubx ));
disp(sprintf('Class ruby : True: %02d False: %02d TPR:
%.1f',correct_ruby,wrong_ruby,percent_correct_ruby ));
disp(sprintf('Class tap1 : True: %02d False: %02d TPR:
%.1f',correct_tap1,wrong_tap1,percent_correct_tap1 ));
disp(sprintf('Class tap2 : True: %02d False: %02d TPR:
%.1f',correct_tap2,wrong_tap2,percent_correct_tap2 ));
disp(sprintf('Class tap3 : True: %02d False: %02d TPR:
%.1f',correct_tap3,wrong_tap3,percent_correct_tap3 ));
disp(sprintf('Class tap4 : True: %02d False: %02d TPR:
%.1f',correct_tap4,wrong_tap4,percent_correct_tap4 ));
disp(sprintf('Class tap5 : True: %02d False: %02d TPR:
%.1f',correct_tap5,wrong_tap5,percent_correct_tap5 ));
disp(sprintf('Confusion Matrix:'))
disp(sprintf(' rollx rubx ruby tap1 tap2 tap3 tap4 tap5'))
disp(sprintf('rollx %02d %02d %02d %02d %02d %02d %02d %02d
%02d',rollxcount1,rubxcount1,rubycount1,tap1count1,tap2count1,tap3count1,tap4count1,tap5count1));
disp(sprintf('rubx %02d %02d %02d %02d %02d %02d %02d %02d
%02d',rollxcount2,rubxcount2,rubycount2,tap1count2,tap2count2,tap3count2,tap4count2,tap5count2));
disp(sprintf('ruby %02d %02d %02d %02d %02d %02d %02d %02d
%02d',rollxcount3,rubxcount3,rubycount3,tap1count3,tap2count3,tap3count3,tap4count3,tap5count3));
disp(sprintf('tap1 %02d %02d %02d %02d %02d %02d %02d %02d
%02d',rollxcount4,rubxcount4,rubycount4,tap1count4,tap2count4,tap3count4,tap4count4,tap5count4));
disp(sprintf('tap2 %02d %02d %02d %02d %02d %02d %02d %02d
%02d',rollxcount5,rubxcount5,rubycount5,tap1count5,tap2count5,tap3count5,tap4count5,tap5count5));
disp(sprintf('tap3 %02d %02d %02d %02d %02d %02d %02d %02d
%02d',rollxcount6,rubxcount6,rubycount6,tap1count6,tap2count6,tap3count6,tap4count6,tap5count6));
disp(sprintf('tap4 %02d %02d %02d %02d %02d %02d %02d %02d
%02d',rollxcount7,rubxcount7,rubycount7,tap1count7,tap2count7,tap3count7,tap4count7,tap5count7));
disp(sprintf('tap5 %02d %02d %02d %02d %02d %02d %02d %02d
%02d',rollxcount8,rubxcount8,rubycount8,tap1count8,tap2count8,tap3count8,tap4count8,tap5count8));

```

#### Receiver Operating Characteristics (ROC) Evaluation

```
% evaluation metrics for the classifiers
```

```

rollx_tp=A(1,1); rubx_tp=A(2,2); ruby_tp=A(3,3); tap1_tp=A(4,4); tap2_tp=A(5,5); tap3_tp=A(6,6);
tap4_tp=A(7,7); tap5_tp=A(8,8);
rollx_fn=A(1,2)+A(1,3)+A(1,4)+A(1,5)+A(1,6)+A(1,7)+A(1,8);
rubx_fn=A(2,1)+A(2,3)+A(2,4)+A(2,5)+A(2,6)+A(2,7)+A(2,8);
ruby_fn=A(3,1)+A(3,2)+A(3,4)+A(3,5)+A(3,6)+A(3,7)+A(3,8);
tap1_fn=A(4,1)+A(4,2)+A(4,3)+A(4,5)+A(4,6)+A(4,7)+A(4,8);
tap2_fn=A(5,1)+A(5,2)+A(5,3)+A(5,4)+A(5,6)+A(5,7)+A(5,8);
tap3_fn=A(6,1)+A(6,2)+A(6,3)+A(6,4)+A(6,5)+A(6,7)+A(6,8);
tap4_fn=A(7,1)+A(7,2)+A(7,3)+A(7,4)+A(7,5)+A(7,6)+A(7,8);
tap5_fn=A(8,1)+A(8,2)+A(8,3)+A(8,4)+A(8,5)+A(8,6)+A(8,7);
rollx_fp=A(2,1)+A(3,1)+A(4,1)+A(5,1)+A(6,1)+A(7,1)+A(8,1);
rubx_fp=A(1,2)+A(3,2)+A(4,2)+A(5,2)+A(6,2)+A(7,2)+A(8,2);
ruby_fp=A(1,3)+A(2,3)+A(4,3)+A(5,3)+A(6,3)+A(7,3)+A(8,3);
tap1_fp=A(1,4)+A(2,4)+A(3,4)+A(5,4)+A(6,4)+A(7,4)+A(8,4);
tap2_fp=A(1,5)+A(2,5)+A(3,5)+A(4,5)+A(6,5)+A(7,5)+A(8,5);
tap3_fp=A(1,6)+A(2,6)+A(3,6)+A(4,6)+A(5,6)+A(7,6)+A(8,6);
tap4_fp=A(1,7)+A(2,7)+A(3,7)+A(4,7)+A(5,7)+A(6,7)+A(8,7);
tap5_fp=A(1,8)+A(2,8)+A(3,8)+A(4,8)+A(5,8)+A(6,8)+A(7,8);
rollx_tn=A(2,2)+A(3,3)+A(4,4)+A(5,5)+A(6,6)+A(7,7)+A(8,8);
rubx_tn=A(1,1)+A(3,3)+A(4,4)+A(5,5)+A(6,6)+A(7,7)+A(8,8);
ruby_tn=A(1,1)+A(2,2)+A(4,4)+A(5,5)+A(6,6)+A(7,7)+A(8,8);
tap1_tn=A(1,1)+A(2,2)+A(3,3)+A(5,5)+A(6,6)+A(7,7)+A(8,8);
tap2_tn=A(1,1)+A(2,2)+A(3,3)+A(4,4)+A(6,6)+A(7,7)+A(8,8);
tap3_tn=A(1,1)+A(2,2)+A(3,3)+A(4,4)+A(5,5)+A(7,7)+A(8,8);
tap4_tn=A(1,1)+A(2,2)+A(3,3)+A(4,4)+A(5,5)+A(6,6)+A(8,8);
tap5_tn=A(1,1)+A(2,2)+A(3,3)+A(4,4)+A(5,5)+A(6,6)+A(7,7);
rollx_tpr=100*[rollx_tp]/[rollx_tp+rollx_fn];
rollx_fpr=100*[rollx_fp]/[rollx_fp+rollx_tn];
rollx_recall=rollx_tpr;
rollx_precision=100*[rollx_tp]/[rollx_tp+rollx_fp];
rollx_specificity=100*[rollx_tn]/[rollx_fp+rollx_tn];
rollx_accuracy=100*[rollx_tp+rollx_tn]/[rollx_tp+rollx_fn+rollx_fp+rollx_tn];
rollx_error_rate=100*1-rollx_accuracy;
rollx_F_score=[2*rollx_precision*rollx_recall]/[rollx_precision+rollx_recall];
rubx_tpr=100*[rubx_tp]/[rubx_tp+rubx_fn];
rubx_fpr=100*[rubx_fp]/[rubx_fp+rubx_tn];
rubx_recall=rubx_tpr;
rubx_precision=100*[rubx_tp]/[rubx_tp+rubx_fp];
rubx_specificity=100*[rubx_tn]/[rubx_fp+rubx_tn];
rubx_accuracy=100*[rubx_tp+rubx_tn]/[rubx_tp+rubx_fn+rubx_fp+rubx_tn];
rubx_error_rate=100*1-rubx_accuracy;
rubx_F_score=[2*rubx_precision*rubx_recall]/[rubx_precision+rubx_recall];
ruby_tpr=100*[ruby_tp]/[ruby_tp+ruby_fn];
ruby_fpr=100*[ruby_fp]/[ruby_fp+ruby_tn];
ruby_recall=ruby_tpr;
ruby_precision=100*[ruby_tp]/[ruby_tp+ruby_fp];
ruby_specificity=100*[ruby_tn]/[ruby_fp+ruby_tn];
ruby_accuracy=100*[ruby_tp+ruby_tn]/[ruby_tp+ruby_fn+ruby_fp+ruby_tn];
ruby_error_rate=100*1-ruby_accuracy;
ruby_F_score=[2*ruby_precision*ruby_recall]/[ruby_precision+ruby_recall];
tap1_tpr=100*[tap1_tp]/[tap1_tp+tap1_fn];
tap1_fpr=100*[tap1_fp]/[tap1_fp+tap1_tn];
tap1_recall=tap1_tpr;
tap1_precision=100*[tap1_tp]/[tap1_tp+tap1_fp];
tap1_specificity=100*[tap1_tn]/[tap1_fp+tap1_tn];
tap1_accuracy=100*[tap1_tp+tap1_tn]/[tap1_tp+tap1_fn+tap1_fp+tap1_tn];
tap1_error_rate=100*1-tap1_accuracy;
tap1_F_score=[2*tap1_precision*tap1_recall]/[tap1_precision+tap1_recall];
tap2_tpr=100*[tap2_tp]/[tap2_tp+tap2_fn];

```

```

tap2_fpr=100*[tap2_fp]/[tap2_fp+tap2_tn];
tap2_recall=tap2_tpr;
tap2_precision=100*[tap2_tp]/[tap2_tp+tap2_fp];
tap2_specificity=100*[tap2_tn]/[tap2_fp+tap2_tn];
tap2_accuracy=100*[tap2_tp+tap2_tn]/[tap2_tp+tap2_fn+tap2_fp+tap2_tn];
tap2_error_rate=100*1-tap2_accuracy;
tap2_F_score=[2*tap2_precision*tap2_recall]/[tap2_precision+tap2_recall];
tap3_tpr=100*[tap3_tp]/[tap3_tp+tap3_fn];
tap3_fpr=100*[tap3_fp]/[tap3_fp+tap3_tn];
tap3_recall=tap3_tpr;
tap3_precision=100*[tap3_tp]/[tap3_tp+tap3_fp];
tap3_specificity=100*[tap3_tn]/[tap3_fp+tap3_tn];
tap3_accuracy=100*[tap3_tp+tap3_tn]/[tap3_tp+tap3_fn+tap3_fp+tap3_tn];
tap3_error_rate=100*1-tap3_accuracy;
tap3_F_score=[2*tap3_precision*tap3_recall]/[tap3_precision+tap3_recall];
tap4_tpr=100*[tap4_tp]/[tap4_tp+tap4_fn];
tap4_fpr=100*[tap4_fp]/[tap4_fp+tap4_tn];
tap4_recall=tap4_tpr;
tap4_precision=100*[tap4_tp]/[tap4_tp+tap4_fp];
tap4_specificity=100*[tap4_tn]/[tap4_fp+tap4_tn];
tap4_accuracy=100*[tap4_tp+tap4_tn]/[tap4_tp+tap4_fn+tap4_fp+tap4_tn];
tap4_error_rate=100*1-tap4_accuracy;
tap4_F_score=[2*tap4_precision*tap4_recall]/[tap4_precision+tap4_recall];
tap5_tpr=100*[tap5_tp]/[tap5_tp+tap5_fn];
tap5_fpr=100*[tap5_fp]/[tap5_fp+tap5_tn];
tap5_recall=tap5_tpr;
tap5_precision=100*[tap5_tp]/[tap5_tp+tap5_fp];
tap5_specificity=100*[tap5_tn]/[tap5_fp+tap5_tn];
tap5_accuracy=100*[tap5_tp+tap5_tn]/[tap5_tp+tap5_fn+tap5_fp+tap5_tn];
tap5_error_rate=100*1-tap5_accuracy;
tap5_F_score=[2*tap5_precision*tap5_recall]/[tap5_precision+tap5_recall];
overall_tpr=[rollx_tpr+rubx_tpr+ruby_tpr+tap1_tpr+tap2_tpr+tap3_tpr+tap4_tpr+tap5_tpr]/8;
overall_fpr=[rollx_fpr+rubx_fpr+ruby_fpr+tap1_fpr+tap2_fpr+tap3_fpr+tap4_fpr+tap5_fpr]/8;
overall_precision=[rollx_precision+rubx_precision+ruby_precision+tap1_precision+tap2_precision+tap3_precision+tap4_precision+tap5_precision]/8;
overall_specificity=[rollx_specificity+rubx_specificity+ruby_specificity+tap1_specificity+tap2_specificity+tap3_specificity+tap4_specificity+tap5_specificity]/8;
overall_accuracy=[rollx_accuracy+rubx_accuracy+ruby_accuracy+tap1_accuracy+tap2_accuracy+tap3_accuracy+tap4_accuracy+tap5_accuracy]/8;
overall_error_rate=[rollx_error_rate+rubx_error_rate+ruby_error_rate+tap1_error_rate+tap2_error_rate+tap3_error_rate+tap4_error_rate+tap5_error_rate]/8;
overall_F_score=[rollx_F_score+rubx_F_score+ruby_F_score+tap1_F_score+tap2_F_score+tap3_F_score+tap4_F_score+tap5_F_score]/8;
% printing evaluation metrics as a table
disp(sprintf('          TPR    FPR    Precision    Specificity    Accuracy    Error_Rate
F_Score'));
disp(sprintf('    TP:%03d FN:%03d ',rollx_tp,rollx_fn));
disp(sprintf('rollx  FP:%03d TN:%03d    %05.1f    %05.1f    %05.1f    %05.1f    %05.1f
%05.1f
%05.1f,rollx_fp,rollx_tn,rollx_tpr,rollx_fpr,rollx_precision,rollx_specificity,rollx_accuracy,rollx_error_rate,rollx_F_score));
disp(sprintf('    TP:%03d FN:%03d ',rubx_tp,rubx_fn));
disp(sprintf('rubx  FP:%03d TN:%03d    %05.1f    %05.1f    %05.1f    %05.1f    %05.1f
%05.1f
%05.1f,rubx_fp,rubx_tn,rubx_tpr,rubx_fpr,rubx_precision,rubx_specificity,rubx_accuracy,rubx_error_rate,rubx_F_score));
disp(sprintf('    TP:%03d FN:%03d ',ruby_tp,ruby_fn));
disp(sprintf('ruby  FP:%03d TN:%03d    %05.1f    %05.1f    %05.1f    %05.1f    %05.1f
%05.1f

```

```

%05.1f,ruby_fp,ruby_tn,ruby_tpr,rubx_fpr,ruby_precision,ruby_specificity,ruby_accuracy,ruby_error_
rate,ruby_F_score));
disp(sprintf('    TP:%03d FN:%03d ',tap1_tp,tap1_fn));
disp(sprintf('tap1  FP:%03d TN:%03d   %05.1f   %05.1f   %05.1f   %05.1f   %05.1f
%05.1f
%05.1f,tap1_fp,tap1_tn,tap1_tpr,tap1_fpr,tap1_precision,tap1_specificity,tap1_accuracy,tap1_error_ra
te,tap1_F_score));
disp(sprintf('    TP:%03d FN:%03d ',tap2_tp,tap2_fn));
disp(sprintf('tap2  FP:%03d TN:%03d   %05.1f   %05.1f   %05.1f   %05.1f   %05.1f
%05.1f
%05.1f,tap2_fp,tap2_tn,tap2_tpr,tap2_fpr,tap2_precision,tap2_specificity,tap2_accuracy,tap2_error_ra
te,tap2_F_score));
disp(sprintf('    TP:%03d FN:%03d ',tap3_tp,tap3_fn));
disp(sprintf('tap3  FP:%03d TN:%03d   %05.1f   %05.1f   %05.1f   %05.1f   %05.1f
%05.1f
%05.1f,tap3_fp,tap3_tn,tap3_tpr,tap3_fpr,tap3_precision,tap3_specificity,tap3_accuracy,tap3_error_ra
te,tap3_F_score));
disp(sprintf('    TP:%03d FN:%03d ',tap4_tp,tap4_fn));
disp(sprintf('tap4  FP:%03d TN:%03d   %05.1f   %05.1f   %05.1f   %05.1f   %05.1f
%05.1f
%05.1f,tap4_fp,tap4_tn,tap4_tpr,tap4_fpr,tap4_precision,tap4_specificity,tap4_accuracy,tap4_error_ra
te,tap4_F_score));
disp(sprintf('    TP:%03d FN:%03d ',tap5_tp,tap5_fn));
disp(sprintf('tap5  FP:%03d TN:%03d   %05.1f   %05.1f   %05.1f   %05.1f   %05.1f
%05.1f
%05.1f,tap5_fp,tap5_tn,tap5_tpr,tap5_fpr,tap5_precision,tap5_specificity,tap5_accuracy,tap5_error_ra
te,tap5_F_score));
disp(sprintf('
'));
disp(sprintf('Overall          %05.1f   %05.1f   %05.1f   %05.1f   %05.1f   %05.1f
%05.1f',overall_tpr,overall_fpr,overall_precision,overall_specificity,overall_accuracy,overall_error_rat
e,overall_F_score));
% plotting ROC
x1=0; y1=0; x3=100; y3=100;
x21=rollx_fpr; y21=rollx_tpr; x22=rubx_fpr; y22=rubx_tpr; x23=ruby_fpr; y23=ruby_tpr;
x24=tap1_fpr; y24=tap1_tpr; x25=tap2_fpr; y25=tap2_tpr; x26=tap3_fpr; y26=tap3_tpr;
x27=tap4_fpr; y27=tap4_tpr; x28=tap5_fpr; y28=tap5_tpr;
NumberNewPoints = 5;
% red rollx
figure(1)
xrollxv = linspace(x1, x21, NumberNewPoints+2);
yrollxv = linspace(y1, y21, NumberNewPoints+2);
rollxv=plot(xrollxv,yrollxv);
set(rollxv, 'LineWidth',2,'LineSmoothing','on','Color',[1 0 0],'Marker','+','MarkerSize',10)
hold on
xrollxh = linspace(x21, x3, NumberNewPoints+2);
yrollxh = linspace(y21, y3, NumberNewPoints+2);
rollxh=plot(xrollxh,yrollxh);
set(rollxh, 'LineWidth',2,'LineSmoothing','on','Color',[1 0 0],'Marker','+','MarkerSize',10)
hleg1 = legend([rollxv],rollx);
xlabel('FPR'); ylabel('TPR'); axis([-9 109 -9 109]);
grid on
% blue rubx
figure(2)
xrubxv = linspace(x1, x22, NumberNewPoints+2);
yrubxv = linspace(y1, y22, NumberNewPoints+2);
rubxv=plot(xrubxv,yrubxv);
set(rubxv, 'LineWidth',2,'LineSmoothing','on','Color',[0 0 1],'Marker','d','MarkerSize',10)
hold on
xrubxh = linspace(x22, x3, NumberNewPoints+2);

```

```

yrbxh = linspace(y22, y3, NumberNewPoints+2);
rubxh=plot(xrubxh,yrbxh);
set(rubxh, 'LineWidth',2,'LineSmoothing','on','Color',[0 0 1],'Marker','d','MarkerSize',10)
hleg2 = legend([rubxv],'rubx');
xlabel('FPR'); ylabel('TPR'); axis([-9 109 -9 109]);
grid on
% green ruby
figure(3)
xrbyv = linspace(x1, x23, NumberNewPoints+2);
yrbyv = linspace(y1, y23, NumberNewPoints+2);
rbyv=plot(xrbyv,yrbyv);
set(rbyv, 'LineWidth',2,'LineSmoothing','on','Color',[0 1 0],'Marker','o','MarkerSize',10)
hold on
xrbyh = linspace(x23, x3, NumberNewPoints+2);
yrbyh = linspace(y23, y3, NumberNewPoints+2);
rbyh=plot(xrbyh,yrbyh);
set(rbyh, 'LineWidth',2,'LineSmoothing','on','Color',[0 1 0],'Marker','o','MarkerSize',10)
hleg3 = legend([rbyv],'rby');
xlabel('FPR'); ylabel('TPR'); axis([-9 109 -9 109]);
grid on
% peach tap1
figure(4)
xtap1v = linspace(x1, x24, NumberNewPoints+2);
ytap1v = linspace(y1, y24, NumberNewPoints+2);
tap1v=plot(xtap1v,ytap1v);
set(tap1v, 'LineWidth',2,'LineSmoothing','on','Color',[0 0.5 0.5],'Marker','>','MarkerSize',10)
hold on
xtap1h = linspace(x24, x3, NumberNewPoints+2);
ytap1h = linspace(y24, y3, NumberNewPoints+2);
tap1h=plot(xtap1h,ytap1h);
set(tap1h, 'LineWidth',2,'LineSmoothing','on','Color',[0 0.5 0.5],'Marker','>','MarkerSize',10)
hleg4 = legend([tap1v],'tap1');
xlabel('FPR'); ylabel('TPR'); axis([-9 109 -9 109]);
grid on
% magenta tap2
figure(5)
xtap2v = linspace(x1, x25, NumberNewPoints+2);
ytap2v = linspace(y1, y25, NumberNewPoints+2);
tap2v=plot(xtap2v,ytap2v);
set(tap2v, 'LineWidth',2,'LineSmoothing','on','Color',[1 0 1],'Marker','<','MarkerSize',10)
hold on
xtap2h = linspace(x25, x3, NumberNewPoints+2);
ytap2h = linspace(y25, y3, NumberNewPoints+2);
tap2h=plot(xtap2h,ytap2h);
set(tap2h, 'LineWidth',2,'LineSmoothing','on','Color',[1 0 1],'Marker','<','MarkerSize',10)
hleg5 = legend([tap2v],'tap2');
xlabel('FPR'); ylabel('TPR'); axis([-9 109 -9 109]);
grid on
% cyan tap3
figure(6)
xtap3v = linspace(x1, x26, NumberNewPoints+2);
ytap3v = linspace(y1, y26, NumberNewPoints+2);
tap3v=plot(xtap3v,ytap3v);
set(tap3v, 'LineWidth',2,'LineSmoothing','on','Color',[0 1 1],'Marker','v','MarkerSize',10)
hold on
xtap3h = linspace(x26, x3, NumberNewPoints+2);
ytap3h = linspace(y26, y3, NumberNewPoints+2);
tap3h=plot(xtap3h,ytap3h);
set(tap3h, 'LineWidth',2,'LineSmoothing','on','Color',[0 1 1],'Marker','v','MarkerSize',10)

```

```

hleg6 = legend([tap3v], 'tap3');
xlabel('FPR'); ylabel('TPR'); axis([-9 109 -9 109]);
grid on
% black tap4
figure(7)
xtap4v = linspace(x1, x27, NumberNewPoints+2);
ytap4v = linspace(y1, y27, NumberNewPoints+2);
tap4v=plot(xtap4v, ytap4v);
set(tap4v, 'LineWidth', 2, 'LineStyle', 'solid', 'Color', [0 0 0], 'Marker', '^', 'MarkerSize', 10)
hold on
xtap4h = linspace(x27, x3, NumberNewPoints+2);
ytap4h = linspace(y27, y3, NumberNewPoints+2);
tap4h=plot(xtap4h, ytap4h);
set(tap4h, 'LineWidth', 2, 'LineStyle', 'solid', 'Color', [0 0 0], 'Marker', '^', 'MarkerSize', 10)
hleg7 = legend([tap4v], 'tap4');
xlabel('FPR'); ylabel('TPR'); axis([-9 109 -9 109]);
grid on
% orange tap5
figure(8)
xtap5v = linspace(x1, x28, NumberNewPoints+2);
ytap5v = linspace(y1, y28, NumberNewPoints+2);
tap5v=plot(xtap5v, ytap5v);
set(tap5v, 'LineWidth', 2, 'LineStyle', 'solid', 'Color', [1 0.6 0], 'Marker', 's', 'MarkerSize', 10)
hold on
xtap5h = linspace(x28, x3, NumberNewPoints+2);
ytap5h = linspace(y28, y3, NumberNewPoints+2);
tap5h=plot(xtap5h, ytap5h);
set(tap5h, 'LineWidth', 2, 'LineStyle', 'solid', 'Color', [1 0.6 0], 'Marker', 's', 'MarkerSize', 10)
hleg8 = legend([tap5v], 'tap5');
xlabel('FPR'); ylabel('TPR'); axis([-9 109 -9 109]);
grid on

```

## Appendix J: C Codes

### K-Mean Algorithm

```
#   define H 400

int i, j;

point p, c;

double min_x, max_x, min_y, max_y, scale, cx, cy;

double *colors = malloc(sizeof(double) * n_cluster * 3);

for_n {

    colors[3*i + 0] = (3 * (i + 1) % 11)/11.;
    colors[3*i + 1] = (7 * i % 11)/11.;
    colors[3*i + 2] = (9 * i % 11)/11.;

}

max_x = max_y = -(min_x = min_y = HUGE_VAL);

for_len {

    if (max_x < p->x) max_x = p->x;
    if (min_x > p->x) min_x = p->x;
    if (max_y < p->y) max_y = p->y;
    if (min_y > p->y) min_y = p->y;

}

scale = W / (max_x - min_x);

if (scale > H / (max_y - min_y)) scale = H / (max_y - min_y);

cx = (max_x + min_x) / 2;
cy = (max_y + min_y) / 2;

printf("%% !PS-Adobe-3.0\n%% %%BoundingBox: -5 -5 %d %d\n", W + 10, H + 10);

printf( "/l {rlineto} def /m {rmoveto} def\n"

        "/c { .25 sub exch .25 sub exch .5 0 360 arc fill } def\n"

        "/s { moveto -2 0 m 2 2 1 2 -2 1 -2 -2 1 closepath "

        "    gsave 1 setgray fill grestore gsave 3 setlinewidth"

        "    1 setgray stroke grestore 0 setgray stroke } def\n"

);

for_n {

    printf("%g %g %g setrgbcolor\n",
```

```

        colors[3*i], colors[3*i + 1], colors[3*i + 2]);
    for_len {
        if (p->group != i) continue;
        printf("%.3f %.3f c\n",
            (p->x - cx) * scale + W / 2,
            (p->y - cy) * scale + H / 2);
    }
    printf("\n0 setgray %g %g s\n",
        (c->x - cx) * scale + W / 2,
        (c->y - cy) * scale + H / 2);
}
printf("\n% % % EOF");
free(colors);
# undef for_n
# undef for_len
}
#define PTS 100000
#define K 11
int main()
{
    int i;
    point v = gen_xy(PTS, 10);
    point c = lloyd(v, PTS, K);
    print_eps(v, PTS, c, K);
    // free(v); free(c);
    return 0;
}

```

Forward Algorithm

```

#include <stdio.h>

double HMM_forwardAlgorithm(double* &pi, double** &A, double ** &B, int* &obs, int sN, int
oN, int obs_N);

void main()

```

```

{
//Input HMM ramda
int stateN, observeN, ob_seq_N;

double *pi;
double **A;
double **B;

int *obS;
FILE * fp;

fp = fopen("HMM_Ramda2.txt", "r");
fscanf(fp, "%d %d %d", &stateN, &observeN, &ob_seq_N);

//alloc buffer

pi = new double[stateN];
A = new double*[stateN];
B = new double*[stateN];
obS = new int[ob_seq_N];

for(int i=0; i<stateN; ++i)
{
    A[i] = new double[stateN];
    B[i] = new double[observeN];
}

//read Data

//pi
for(int i=0; i<stateN; ++i)
    fscanf(fp, "%lf", &(pi[i]) );

//A
for(int i=0; i<stateN; ++i)
    for(int j=0; j<stateN; ++j)
        fscanf(fp, "%lf", &(A[i][j]) );

//B
for(int i=0; i<stateN; ++i)
    for(int j=0; j<observeN; ++j)
        fscanf(fp, "%lf", &(B[i][j]) );

```

```

//observe sequence
for(int i=0; i<ob_seq_N; ++i)
    fscanf(fp, "%d", &(obS[i]));
HMM_forwardAlgorithm(pi, A, B, obS, stateN, observeN, ob_seq_N);
fclose(fp);
for(int i=0; i<stateN; ++i)
{
    delete[] A[i];
    delete[] B[i];
}
delete[] A;
delete[] B;
delete[] pi;
delete[] obS;
}

double HMM_forwardAlgorithm(double* &pi, double** &A, double ** &B, int* &obS, int sN, int
oN, int obs_N)
{
    int i,j,t;
    //make forward matrix
    double** fMtx = new double *[sN];
    for(i=0; i<sN; ++i)
        fMtx[i] = new double[obs_N];
    //first step
    for(i=0; i<sN; ++i)
    {
        fMtx[i][0] = pi[i] * B[i][ obS[0]-1 ];
    }
    //routine
    double sum;
    for(t=1; t<obs_N; ++t)
    {

```

```

    for(j=0; j<sN; ++j)
    {
        sum=0;
        for(i=0; i<sN; ++i)
        {
            sum += (fMtx[i][t-1]*A[i][j]);
        }
        fMtx[j][t] = sum * B[j][ obs[t]-1];
    }
}
//final
sum=0;
for(j=0; j<sN; ++j)
{
    sum += (fMtx[j][obs_N-1]);
}
//report
printf("state N=%d / Observe N=%d\n", sN, oN);
printf("Initial Matrix\n");
for(int i=0; i<sN; ++i)
    printf("%lf ", pi[i] );
printf("\n\n");
printf("A matrix\n");
for(int i=0; i<sN; ++i)
{
    for(int j=0; j<sN; ++j)
    {
        printf("%lf ", A[i][j]);
    }
    printf("\n");
}
printf("\n");

```

```

printf("B matrix\n");
for(int i=0; i<sN; ++i)
{
    for(int j=0; j<oN; ++j)
    {
        printf("%lf ", B[i][j]);
    }
    printf("\n");
}
printf("\n");
printf("Observation sequence\n");
for(int i=0; i<obs_N; ++i)
    printf("%d ", obs[i]);

printf("\n\n");
printf("*** Trellis matrix ***\n");
for(j=0; j<sN; ++j)
{
    for(t=0; t<obs_N; ++t)
    {
        printf("%lf ", fMtx[j][t]);

        if(t!=obs_N-1)
            printf("-> ");
    }
    printf("\n");
}
printf("\nLikelihood = %lf\n", sum);
for(i=0; i<sN; ++i)
    delete[] fMtx[i];
delete[] fMtx;
return sum;}

```

## **Vita**

Waqqas Munir Khan was born in Lahore, Pakistan. He was educated in a local public school and graduated with roll of honor from Air University, Pakistan in 2007. His degree was a Bachelor of Engineering in Mechatronics.

Mr. Waqqas joined Knowledge Platform as a hardware design engineer in 2008 and designed and implemented ZigBee-based clicker hardware. He moved to United Arab Emirates in 2010 where he worked as Graduate Research Assistant at the American University of Sharjah.

In 2012, Mr. Waqqas joined Aker Solutions as a field engineer based in Abu Dhabi. As a field engineer, he was involved in the design, testing and maintenance of Aker's down hole tools. Mr. Waqqas has both onshore and offshore oil field work experience in the Middle East. His research interests are in the design for embedded systems, sensor integration, and modeling of non-linear control systems.