

Gesture Recognition and Interaction with a Glove Controller

An Approach with a Glove Based on Accelerometers

Gerard Llorach Tó

Electronic Music Unit, Elder Conservatorium
University of Adelaide
July 2013

CONTENTS

- ABSTRACT..... 1**

- STATEMENT..... 2**

- ACKNOWLEDGMENTS..... 3**

- INTRODUCTION..... 4**

- HISTORY AND EVOLUTION OF GLOVE CONTROLLERS..... 5**

- METHODOLOGY AND PROCESSES..... 18**
 - 1. GLOVE PHYSICAL CONSTRUCTION..... 18**
 - 1.1 DEVICES..... 19
 - 1.2 PLACEMENT OF THE ACCELEROMETERS..... 21
 - 1.3 PLACEMENT OF TEENSY, XBEE AND BATTERY..... 23
 - 1.4 CIRCUIT DESIGN..... 24
 - 1.5 THREAD MATERIAL..... 26
 - 1.6 SEWING..... 27
 - 1.7 PROBLEMS..... 30

 - 2. COMMUNICATION..... 32**
 - 2.1 NETWORK STRUCTURE..... 32
 - 2.2 PROTOCOL..... 36
 - 2.3 COMMUNICATION PROCESS..... 39

 - 3. INTERPRETATION..... 42**
 - 3.1 GLOVE DATA..... 42
 - 3.2 GESTURE RECOGNITION..... 44

4. SOFTWARE.....	51
4.1 PD-EXTENDED & PROCESSING.....	51
4.2 OPENFRAMEWORKS.....	52
4.3 ARDUINO AND TEENSY LOADER.....	53
5. INTERFACE.....	54
5.1 FUNCTIONS OF THE PROGRAM.....	55
CONCLUSION AND RESULTS.....	63
APPENDIX A – SOURCE CODE.....	65
APPENDIX B - LIST OF FIGURES.....	142
APPENDIX C - DATA COMPACT DISC.....	145
BIBLIOGRAPHY.....	146

ABSTRACT

Composing and interacting with music through electronic devices mapped on the body has been a new aspect of music this last 50 years. Technologies have improved greatly, making the interaction with computers within the context of music very attractive.

Human communication and interaction has a lot to do with the body expression. Hands are one of the most expressive and active parts of the body. They are our tool to interact with the environment.

However, most the electronic devices to create music are not intuitive and they require experience and learning. This project designs a glove controller, which allows to create a natural human way to interact and communicate in the context of music technology.

The work in this project aims to map and recognize the expression of the hands, and their movements and gestures to interact, create and perform music. This thesis covers the process of creating an interactive glove controller, its communications processes and the interpretation of the gestures of the hand movements.

STATEMENT

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis, when deposited in the University Library, being available for loan.

A handwritten signature in black ink, appearing to be 'Gerard Llorach Tó', written in a cursive style.

Gerard Llorach Tó , 13 July 2013

ACKNOWLEDGMENTS

Sebastian Tomczak - for his excellence as a mentor, guidance and teacher.

Narcís Parés - for his guidance and suggestions.

Stephen Whittington, Sebastian Tomczak, The Electronic Music Unit, the Elder Conservatorium and the University of Adelaide - for providing support and resources.

Escola Superior Politècnica, Universitat Pompeu Fabra - for their teaching and knowledge, as well as guidance and support.

Carles Llorach, Gemma Tó, Marcel Llorach - for their support, guidance, patience and understanding.

INTRODUCTION

The first aim of the project was to create an interactive device with basic electronic components, map the data into a computer and control music and graphics. Having in mind that interaction requires human actions and movements, the project went into the direction of mapping the hand actions with sensors.

Music has been expressed through dancing or with body movements throughout these last centuries and probably through all of human history. The reasoning behind using gloves controllers is that the hands are a very expressive part of the body. The vision in this project is that music should be created and controlled intuitively, without having to consider which button to press. The usage of accelerometers is challenging because a lot of information can be extracted from them.

The goal of this project is to use accelerometers as capturing devices, improve the sampling rate, recognize gestures and develop a wireless device. As most of the non-commercialized gloves are not very aesthetic it is one of the goals to improve that aspect using conductive thread and using as few devices as possible.

This paper will go through all the steps taken in the development of the glove and its interaction with a computer. This will involve the physical construction of the glove, the creation of a communication protocol, the interpretation of the glove and the software. The research about the physical construction of the glove is related with the existing gloves. The best features of them are taken and put together. A research about the gesture recognition will go through the related papers. This project will approach the recognition from a different perspective as the other papers.

HISTORY AND EVOLUTION OF GLOVE CONTROLLERS

The idea of controlling technologies with the hands has been there since the 70s. The film *Minority Report* is a well-known example in science fiction¹. The fact that the communication with a computer could be done by our hand movements is fascinating. The difference about using a device and wearing a glove is that the glove is an accessory for the hand, not an external object that doesn't follow the body. The idea of not having to use a physical external object that constraints the movement of the hand is one of the reasons why a glove controller is attractive.

Some companies² have tried to build glove controllers but none of them have been successfully commercialized for home users or everyday use. Most of them are created for industrial or research purposes with high prices in the market. The ones destined to the consumer market haven't reach a lot of places mostly because of several reasons; not a lot of software is prepared for them or there aren't applications for them, the glove is not comfortable, the system requires intermediate software (not a plug-and-go system) and other factors.

¹ *Minority Report*, 2002. Film. Directed by Steven Spielberg. USA: Twenty Century Fox Film Corporation.

² Nintendo Entertainment System (Power Glove), Essential Reality (P5 Gaming Glove), Iron Will Innovations (The Peregrine)

SAYRE GLOVE (1977)¹

This was the first wired glove or dataglove. It was created by Electronic Visualization Laboratory. It wasn't relatively expensive, it was light and provided a multidimensional control. It used flex sensors (light based sensors with flexible tubes).

DIGITAL DATA ENTRY GLOVE (1983)²

It was the first recognized device for detecting hand positions. It had flex sensors, tactile sensors, orientation sensing and wrist-positioning sensors. It was created by Bell Labs and the main goal was to emulate a keyboard with the hand.

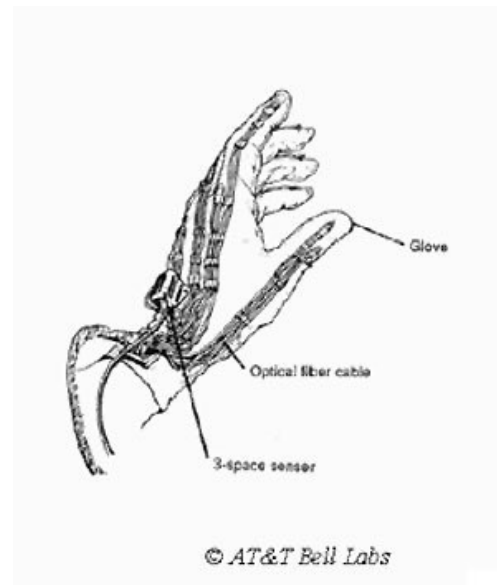


Figure 1. Digital Data Entry Glove

POWER GLOVE (1989)³

Nintendo was one of the first companies to release a glove controller. It has flex sensors, a position tracker with ultrasonic sensors and buttons on the forearm. The glove was developed by other companies. Thomas G. Zimmerman created a prototype with flex sensors and hand position tracking with ultrasonic sensors. Nintendo's Power Glove was based on this prototype⁴.

¹ Sturman, D.J., Zeltzer, D. (January 1994). "A survey of glove-based input". *IEEE Computer Graphics and Applications*

² Grimes G, Digital Data Entry Glove interface device, Patent 4,414,537, AT&T Bell Labs, November 1983.

³ Mattel Co., The PowerGlove, Nintendo Entertainment System, Retrieved 2013-04-01
<www.ebay.com/bhp/nintendo-power-glove>

⁴ Zimmerman T.G and Lanier J, Computer data entry and manipulation apparatus method, Patent Application 5,026,930, VPL Research Inc, 1992.



Figure 2. Power Glove

The flex sensors (carbon-based ink on plastic) only have a resolution of 2 bits. The glove has four flex sensors, thus 1 byte. There are two ultrasonic transmitters in the glove and three ultrasonic receivers around the TV monitor. Pulses at 40 kHz are sent and the system measures the time delay. Through triangulation the system can determine X, Y and Z location of the two transmitters thus, determining the yaw and roll. The pitch rotation cannot be calculated.

The Power Glove was considered a commercial failure, it was too big and uncomfortable for its possibilities¹. The ultrasonic receivers were badly designed and could barely stand on the TV. Also a only a few games were designed for it. Some artists use it and map their own controls² and tutorials of how to hack them can be found³.

DATAGLOVE (1989)⁶

The Power Glove was based on the DataGlove. It was developed by VPL. Young L. Harvey (a VPL researcher) was the one that invented the flex sensors based on optical fibers. The DataGlove was created with a neoprene fabric with two fiber optic loops on each finger. For each knuckle there as a loop dedicated, which caused that the position of the knuckle would change on users and the process of recognition would fail. The



Figure 3. DataGlove

¹ ABC editor, "Backwards Compatible - The Power Glove". ABC website - Good Game. Australian Broadcasting Corporation (ABC). 19 May 2008. Retrieved 2009-06-06.

² DubSpot, 'Ableton Live + Nintendo Power Glove: Meet Controllerist Yeuda Ben-Atar aka Side Brain @ Dubspot', video, YouTube, Oct 23 2012, Retrieved 2013-05-08.

³ How to build an instrumented glove based on the Powerglove flex sensors. PCVR 16 pp 10–14. Stoughton, WI, USA: PCVR Magazine, 1994

resolution of the flex sensors has 8 bits for each one. The DataGlove was expensier than the Power Glove as it was not designed for the home user (\$9000). It was the first commercially available glove.

CYBERGLOVE (1990)¹

This device was the glove that came next. CyberGlove has been a line of products by CyberGlove Systems LLC. The company kept working on the glove and released different products, such as:



Figure 4. CyberGlove II

CyberGlove II² has a lot of flex sensors, between 18 or 22, with a resolution of less than 1 degree. The 22-sensor model has three flex sensors per finger. The sampling rate is 90 Hz and it is wireless. The receiver can support two gloves working at the same time. The glove itself looks like a normal glove but the battery is placed in a wrist band which breaks with the aesthetic of the glove.

CyberGrasp³ has a very accurate mapping of the movement of the fingers. It has 5 actuators for each finger that also allow to make the glove move from the computer. It can apply until 12N of force. It is a complement of the CyberGlove II.



Figure 5. CyberGrasp

¹ Kessler G.D, Hodges L.F, Walker N., Evaluation of the CyberGlove as a Whole Hand Input Device, ACM Transactions on Computer-Human Interaction. 2(4), 1995, pp. 263-283.

² CyberGlove Systems, CyberGlove II Wireless Glove, 2010, Retrieved 2013-04-05, <cyberglovesystems.com/?q=products/cyberglove-ii/overview>

³ CyberGlove Systems, CyberGrasp, 2010, Retrieved 2013-04-05, <cyberglovesystems.com/products/cybergrasp/overview>



Figure 6. CyberForce

CyberForce¹: it is a complex armature based on the CyberGrasp. The system provides a 6 degree control (6 DOF).

5DT DATA GLOVE(1999)²

Fifth Dimension Technologies has some glove controllers:

5DT Data Glove 5 and 14 Ultra: 5 or 14 flex sensors, with a 12 bit resolution. The sampling rate is 75 Hz. The 5DT Data Glove 5 has one flex sensor per finger and a 2-axis tilt sensor. It can detect pitch and roll. The sampling can be up to 200 samples per second.

The 5DT Data Glove 16 has 14 flex sensor but it doesn't have a tilt sensor. It maps the fingers better than the last one. The sampling frequency is 100Hz.

All this gloves have a wireless kit that consists in a belt with a battery. The gloves are connected through wires.

¹ CyberGlove Systems, CyberForce Force Feedback System, Retrieved 2013-04-05, <cyberglovesystems.com/products/cyberforce/overview>

² Fifth Dimension Technologies, The 5th Glove, Retrieved 2013-04-05, <www.5dt.com/hardware.html#glove>



Figure 7. 5DT Data Glove with the wireless kit.

P5 GLOVE (2002)¹

It is an affordable glove controller (\$60). It has 6 degrees of tracking: X, Y, Z, Yaw, Pitch and Roll that are tracked with an infrared control receptor. It also has bend sensors on each finger and 4 buttons. The ownership and production of this glove has been going from one company to another. There are few games prepared for the glove, although it is a good mouse and joystick emulator².



Figure 8. P5 Glove

¹ Essential Reality, P5 Gaming Glove, 2009, Retrieved 20013-05-08
<www.gizmag.com/go/1148/>

² Biotecmexico, 'P5 Glove', video, YouTube, 7th of Nov 2007, Retrieved 20013-05-08

SHAPE HAND (2005)¹

This device is another wireless glove with flex sensors, 1 per finger. The company promotes its flexibility as it can be used in different glove sizes and also left and right with the same device. ShapeHandPlus includes a flex sensor for the arm, so movements and positions of the arm can be detected. The sampling rate changes depending on the computer it is used. In an average laptop it can reach 100Hz.



Figure 9. Shape Hand in different positions.

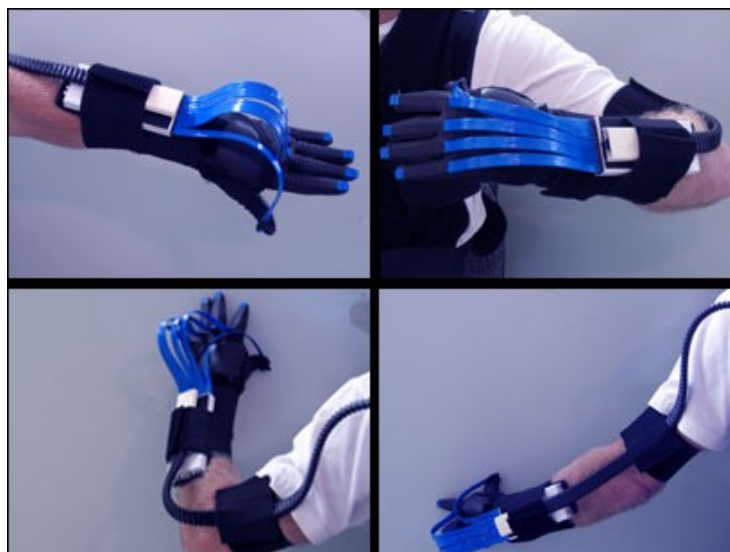


Figure 10. Shape Hand with the arm accessory.

¹ Measurand Inc, ShapeHand, 2009, Retrieved 20013-05-08
<www.finger-motion-capture.com/shapehand.html>

PEREGRINE GLOVE (2010)²

This is one of the gloves that its company is still doing marketing and is active. It has a completely different approach than the others. This glove doesn't try to track motion or hand positions. It is full of touch sensors that act as buttons. The user can use the glove as a keyboard and map all kind of actions from it. Each glove is customizable because you can store the data in the device that connects the glove with the computer. It has 30 touchpoints.

The product offers tutorials about how to write letters and use it. It is not relatively expensive (\$150).



Figure 11. The Peregrine

INATION¹

The company collects all kind of interactive motion capture devices. It sells some of the gloves mentioned before but it also has some others, such as:



Figure 12. Pinchglove.

Pinchglove²: detects when two or more fingers make contact. It uses a RS232 communication protocol, 9600 and 19600 baud rates and optional accessories such as wireless and motion tracker mounts.

Didjiglove³: the glove uses bend sensors. Each sensor has a 10 bit resolution and 200 samples per second.



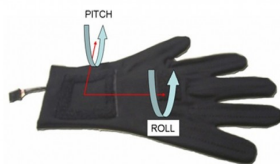
Figure 13. Didjiglove

² Iron Will Innovations, The Peregrine, 2012, Retrieved 20013-05-08 <theperegrine.com>

¹ Inition Co. 2001, Retrieved 2013-04-28, <inition.co.uk/3D-Technologies/productsection/43>

² Inition Co., Fakespace Labs PINCH Glove, Retrieved 2013-04-28, <inition.co.uk/3D-Technologies/fakespace-labs-pinch-gloves>

³ Inition Co., Didjiglove, Retrieved 2013-04-28, <inition.co.uk/3D-Technologies/didjiglove>



DGTech Vhand¹: it has a 3 axes accelerometer and has the bend sensors internally embedded.

Figure 14. DGTech Vhand

X-IST Data Glove²: 24 sensors can be integrated in the glove: bend sensors, pressure sensors and 2 axis tilt sensors. It needs a physical processing box to transfer the data to the computer through USB. 100-200 Hz of sampling rate and 10 bit resolution. This glove has a variant, the X-IST MIDIGlove, that maps the data into MIDI through the same sensors.



Figure 15. X-IST Data Glove

IGLOVE/ACCELEGLOVE³

This glove uses 6 accelerometers (fingers and palm) with a sampling rate of 35Hz. The iGlove is a patented accelerometer technology detects motion of the fingers, hand, wrist and arm that is now being refined by health institutes⁴.

¹ Inition Co., DGTech Vhand, Retrieved 2013-04-28, <inition.co.uk/3D-Technologies/dgtech-vhand>

² Inition Co., X-IST Data Glove, Retrieved 2013-04-28, <inition.co.uk/3D-Technologies/x-ist-data-glove>

³ Hernandez-Rebollar, Jose L., Kyriakopoulos, N., Lindeman, Robert W. (2002) The AcceleGlove: a whole-hand input device for virtual reality

⁴ Abolfathi, Peter P., Interpreting sign language is just the beginning for the AcceleGlove open source dataglove, *Gizmag website*, July 23, 2009, Retrieved 2013-06-14



Figure 16. Acceleglove.

The acceleglove is no longer on sale and can only be ordered as a custom glove to the company that owns it. There isn't much information about the glove in the official web page but there is information on their youtube channel¹.

The company has developed software to control the mouse and other programs such as CAD.

¹ AnthroTronix Inc., AnthroTronix YouTube Channel, YouTube, Retrieved 2013-06-14

NON-COMMERCIALIZED PRODUCTS



Figure 17. KeyGlove.

THE KEYGLOVE¹

The keyglove is based in touch sensors. It has 37 touch sensors and uses Teensy++. It has a tutorial² to teach how to use the glove as a keyboard. It also has a tilt sensor.

BEN'S GLOVE OF POWER³

The glove uses arduino with an accelerometer and a gyroscope. Also it has conductive threading into the fingertips and palm to recreate buttons. The glove was created to help interaction with kinect.

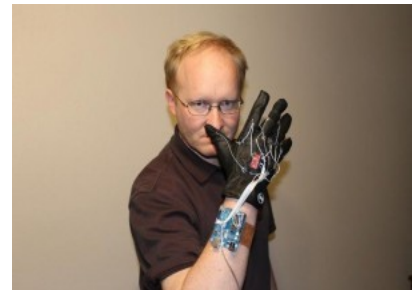


Figure 18. Ben's Glove of Power.



Figure 19. Clove 2.

CLOVE 2⁴

This device has multiple touch sensors to recreate a keyboard. It uses a bluetooth device so it is a wireless device. The web page demonstrates how to build one.

¹ Rowberg, J., "keyglove, freedom in the palm of your hand", Keyglove website, Retrieved 2013-05-09

² Rowberg, J., Keyglove Promo #01, video, Vimeo, 2013 Retrieved 2013-04-05

³ The Ben Heck Show, Episode Power Glove for Xbox, video, Revision 3 Internet Television March 6, 2012, Retrieved 2013-04-18

⁴ Cemetch, Clove 2 (Cemetch Bluetooth Dataglove), project site, July 8, 2008, Retrieved 2013-04-18

MISTER GLOVES⁵

This glove improves the functionalities of touch gloves like Peregrine. It has included an accelerometer that allows mouse simulations as well as keystrokes. Their project web page is very detailed. It is wireless and has audio feedback on the glove and also one flex sensor. It uses its own USB protocol and its a very complete interaccional glove.

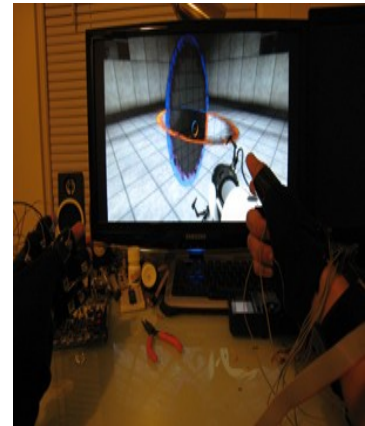


Figure 20. Mister Gloves.

There are lot of more open source projects out there, like this ones above, but I included the most significant ones.

The most commonly used sensors are flex sensors, tilt sensors, accelerometers, gyroscopes, external positioning sensors and touch sensors. Looking through history the first sensors used where the flex sensors, which they are still currently being used.

The ones that work with touch sensors fall into a different category as they require a physical contact in the device (haptics). The information to process about this gloves is much more simpler as the glove is just a bunch of button emulators put together. The Peregrine is one of the commercialized gloves that has reached some video game users as it allows the user to commands faster than writing in the keyboard.

The other category where the gloves can be classified is if they use or not an external device to provide information about positioning and rotation. This will restrict the user into a determinate space, as they depend on the external receiver. The P5 Glove is one example that combines infrared sensors, flex sensors and buttons. This is one of the other gloves that has been commercialized for video game users and that is still in the market. It is an improved version of the Power Glove, as it is based in the same sensors.

⁵ Chen, S., Levine, E., Mister Gloves - A Wireless USB Gesture Input System, project site, 2010, Retrieved 2013-04-18

The last category would be if the device is wireless or not. The fact of using a wireless device will mean that the glove will have more components, like a wireless device and a power supply. The advantage a wireless system has is that will allow the user to be wherever he wants while he keeps himself into the range of the receiver. It should be more comfortable as there won't be any cables around.

Only The Peregrine and the P5 Glove are still active in the market for non-professional users. All the other commercialized gloves have high prices, above \$400, and are intended for other companies or research.

They combine different sensors, gather different information but they have some common features. The gloves reach up to 300 Hz sampling rates and 10-12 bits resolution per sensor. The wireless gloves require an extra device attached to the body, if they didn't before. Most of the non-commercialized gloves are not aesthetic.

The primary method of data acquisition on glove controllers is based on flex sensors or contact sensors. Accelerometers, gyroscopes and tilt sensors usually act as a reinforcement, but only one of the commercialized gloves uses only accelerometers. The Acceleglove has a very low rate (35Hz) and doesn't have any software that is able to recognize gestures.

METHODOLOGY AND PROCESSES

This section explains all the processes and steps that has been followed to build the glove itself, the creation of a network protocol and the research and development of gesture recognition based on accelerometers.

The first section explains the devices it uses, the design of the electronic circuit, sewing conductive thread and the problems that arose. The second section talks about the process of communication and the advantages and disadvantages of using different protocols. The third section goes through all the research and methods of gesture recognizing and describes how to interpret the data from the glove. The fourth section mentions briefly the IDEs used to write the code for the microcontrollers and the the software. The last section explains the functionalities of the graphical interface.

1. GLOVE PHYSICAL CONSTRUCTION

There are a number of sensors and components that are used to construct the glove. Each component has been chosen based on a combination of cost analysis, ease of use and integration, and physical constraints of size.

The first and most important device is a programmable microcontroller. As a microcontroller, Teensy¹ will be the best option, as it is much smaller than Arduino². For transmitting the data the XBee³ series will give a wireless connection. This will avoid the necessity of being near the computer.

¹ Coon, R., Stoffregen, P., Teensy, PJRC Electronic Projects Components Available Worldwide, 2012 - Retrieved 2013-04-05, <www.pjrc.com/teensy/>

² Banzi, M., Cuartielles, D., Zambetti, N., Arduino, 2012, Retrieved 2013-04-05, <www.arduino.cc/>

³ Digi International Inc, XBee, 2012, Retrieved 2013-04-05, <<http://www.digi.com/xbee/>>

The Teensy only has 12 analog inputs, which is why the glove will only be able to use four accelerometers. The idea of introducing a fifth accelerometer and using a multiplexer would be a problem, as the multiplexer may not fit in the glove and that it would involve more threading.

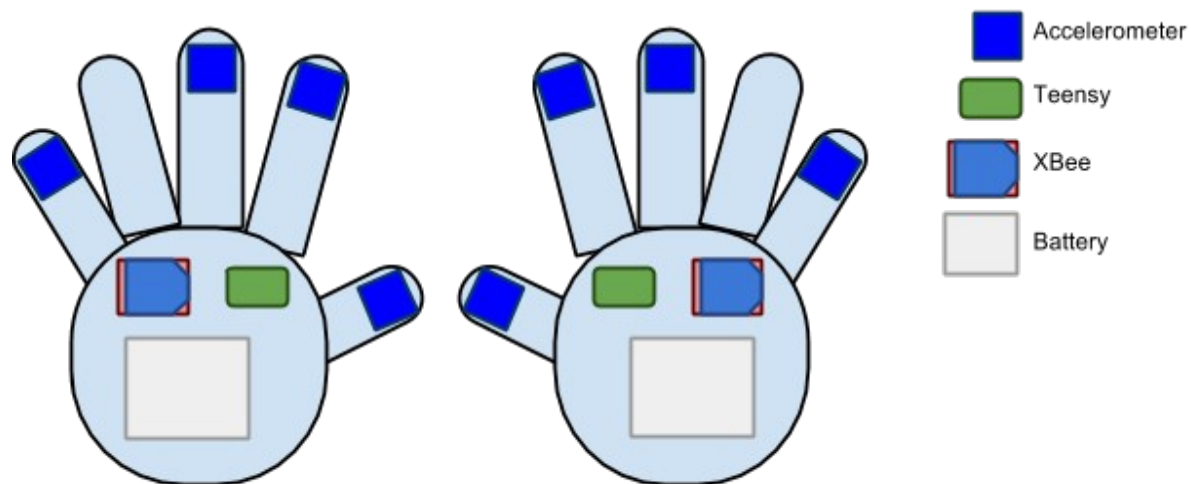


Figure 21. Device placement diagram

1.1 DEVICES

TEENSY 2.0

This device is a programmable microcontroller. It has 25 digital I/O, 12 analog inputs and 7 PWM. It works at 16 MHz, has 32kB of flash memory, 2.5kB of Ram and 1 kB of EEPROM. It operates at 5V and can consume currents between 4 and 20mA approx. depending on its use. It is programmed using the Arduino IDE and the Teensy Loader.

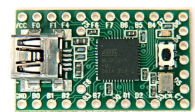


Figure 22.
Teensy 2.0

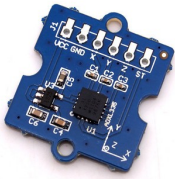


Figure 23.
Accelerometer

ACCELEROMETER ADXL335¹

Provides information about the acceleration of 3 axis. It accepts input voltages of 3.3V and 5V. It measures up to $\pm 3g$. The output signal is analog. The typical bandwidth rates are 1600 Hz for the X and Y axis and 500 Hz for the Z axis. It has a sensitivity of 300mV/g. It consumes 350 μ A.

XBEE SERIES 1²

This XBee has a PCB trace antenna instead of a chip antenna at 2.4GHz which makes it smaller. The input voltage is 3.3V and operates at 50 mA. It can operate at 250 kbps and can reach up to 100m. This device allows point-to-point and multipoint networks. It is not compatible with XBee Series 2. This model has been taken out of the market and replaced by another.



Figure 24. XBee
S1

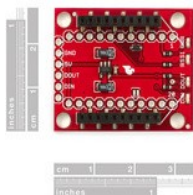


Figure 25. XBee
Explorer Regulated

XBEE EXPLORER REGULATED³

Regulates the incoming voltage for the XBee chip. It also has activity leds that indicate power, receiving, transmitting and RSSI. It makes a 5V signal into a 3.3V signal for the XBee. This model has been taken out of the market and replaced by another.

¹ Little Bird Electronics , Triple Axis Accelerometer Breakout - ADXL335, 2012, Rerieved 2013-04-05, <littlebirdelectronics.com/products/breakout-3-axis-analog-accelerometer-adxl335>

² Sparkfun, XBee 1mW Chip Antenna - Series 1 (802.15.4), 2012, Rerieved 2013-04-05, <www.sparkfun.com/products/8664>

³ Sparkfun, XBee Explorer Regulated WRL -09132, 2012, Rerieved 2013-04-05, <www.sparkfun.com/products/9132>

POLYMER LITHIUM ION BATTERY⁴

It is a very light and slim battery. It outputs a voltage of 3.3V at 2000mAh. It includes a protection circuit for over voltage, over current and minimum voltage. It has to be charged with a specific battery charger.



Figure 26. Polymer Lithium Ion Battery

1.2 PLACEMENT OF THE ACCELEROMETERS

As the glove will only have four accelerometers it is important to research about where to place them. The part that contains more information about the motion of the hand are the fingertips. Placing the accelerometers in the fingertips will allow the mapping of finger movement as well as hand movement and hand positions.

It is important to differentiate between the two types of recognition that can be done on the hand. The hand can be in a static position and give information about a certain position and orientation of the fingers (static). The other information that can be obtained is based on dynamic information, thus when a movement is done and it is interpreted as a gesture.

Analysing the problem from the dynamic perspective two different dynamic informations can be obtained: the wrist and the fingers. In almost all humans, the least independent finger is the ring finger. Usually when the middle finger or the pinky are moved the ring finger follows its movement. Also the movement in the pinky finger makes the ring finger follow.

The placement on the thumb, index finger and middle finger is almost mandatory, so last accelerometer will be on the ring finger or the pinky.

⁴ Sparkfun, Polymer Lithium Ion Battery - 2000mAh PRT-08483, 2012, Rerieved 2013-04-05, </www.sparkfun.com/products/8483>

The following figure analyses different finger configurations of a hand, showing the static information that can be obtained.

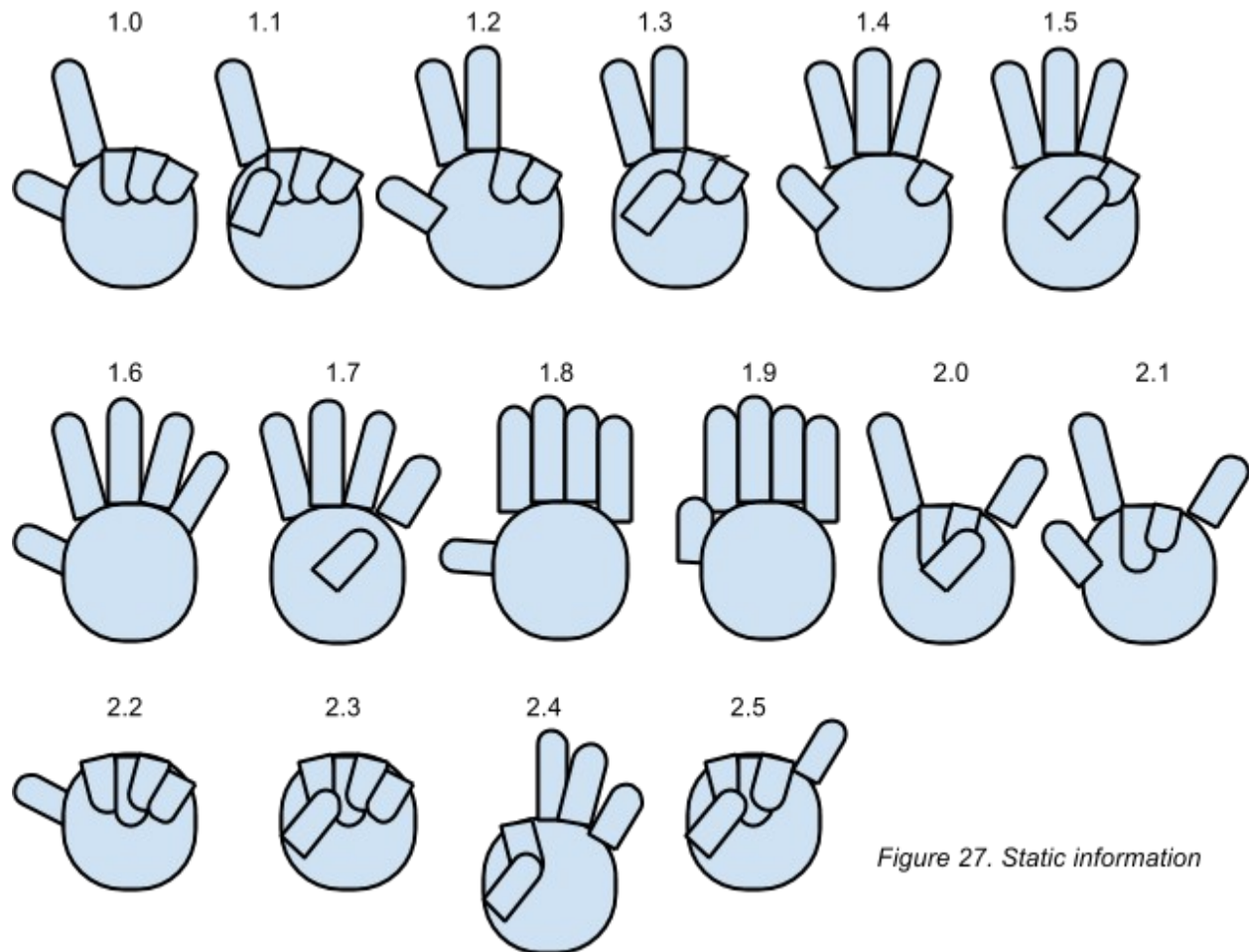


Figure 27. Static information

Working with accelerometers allows to know the relationship between fingers, but not their exact position, only the orientation. For example, positions 1.6, 2.2 and 2.3 of Figure 1 might be confused because the information the accelerometers transmit is very similar. The vectors between the thumb and the other fingers is 90° .

This are the positions that could be recognizable whilst omitting the following finger data:

No thumb: 1.0, 1.2, 1.4, 1.6, 1.8, 2.0

No little finger: 1.0, 1.1, 1.2, 1.3, 1.6, 1.7, 1.8, 1.9, 2.2, 2.3, 2.4

No ring finger: 1.0, 1.1, 1.2, 1.3, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5

From the static point of view avoiding to place an accelerometer in the ring finger would allow to collect more static information. The numbers from 0 to 5 will be recognizable, whereas avoiding the pinky finger it would only have been able to capture from 0 to 4.

Using accelerometers in the thumb, index finger, middle finger and pinky will make the most of the hand information.

1.3 PLACEMENT OF THE TEENYS, XBEE AND BATTERY

The less flexible part of the hand is its back. That is where the rest of the components will be placed. As the teensy will have 12 incoming connections from the fingers it is the device that has a priority above the others. The Teensy will be located right below the end of the index and middle fingers. This will make it easily accessible for the accelerometers.

To the right of the Teensy the XBee will take advantage of the space left, because the battery will occupy the rest of the back of the hand.

The most important thing that it has to be taken into account is where the threads will have to go. As the thread is not isolated, crossings will have to be avoided as much as they can be. Also large connection might involve more problems and interferences or undesired contacts between other threads.

1.4 CIRCUIT DESIGN

The use of the back of the hand is also justified because of the non-isolated threads, as it is the less flexible part of the hand and undesired contacts will not happen.

On the palm of the hand the voltage and ground of the accelerometer will be threaded. It is very important to take into consideration that when the fingers are moved, the voltage threads never establish contact with the ground threads as they will create a short circuit.

To have a clear idea of how the circuit was going to be and know where the crossings were going to happen a 3D model of the hand circuit in Autodesk Maya was designed. The first design (Figure 2) is done with the 3D Paint Tool Brush. The rest of the design was done with the normal Paint Brush because of a lack of computational power (Figure 3).

The yellow, green and blue are the Z, Y and X axis of the accelerometer respectively. The black thread is the ground and the red one is the voltage. Purple and cyan are the transmission and receiver connexions of the XBee.

The crossings between voltage and ground are done in the junctions of the fingers. When the fingers are bended the glove folds where the thread is, covering it from undesired connections. The threads always try to follow the natural folds of the hand for the same reason as before (see Figures 28 and 29).

When threading the fingers, their laterals cannot be always used because they could touch other lateral finger connections.

All the threading paths were studied carefully with the movements of the hand. It is important to bear in mind that the fingers can reach a lot of places in the palm of the hand and other fingers. The voltage and ground threads should never establish contact. This are the threads that go on the palm of the hand and in the lateral and inside of the fingers.



Figure 28. First maya prototype with the 3D Paint Tool Brush. Right hand with Teensy and accelerometers.

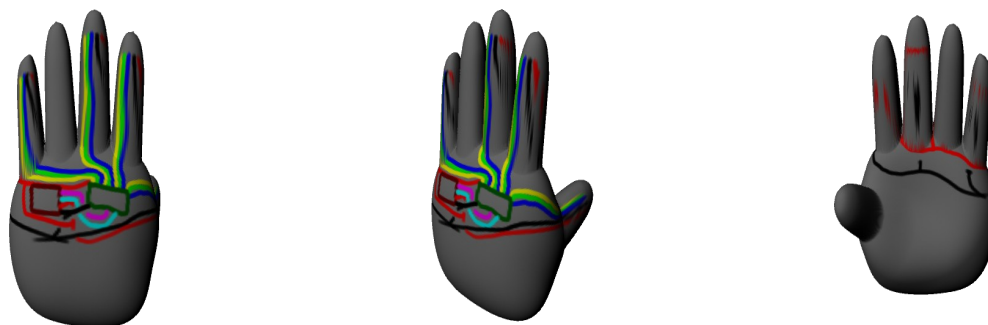


Figure 29. Maya circuit design with the paint brush. Left hand with Teensy, XBee and accelerometers.

1.5 THREAD MATERIAL

There are two different types of conductive thread in the market nowadays: stainless steel conductive thread and plated silver conductive thread. Their characteristics, advantages and disadvantages will be explained in this section.



Figure 30. Stainless steel thread

STAINLESS STEEL THREAD¹

This one is not suitable for machine sewing as it is much thicker. It is composed by stainless steel fibers spun together. This makes it somewhat 'hairy'. It cannot be soldered but it doesn't break in contact with a soldering iron. Fibers burn when a short circuit happens.

PLATED SILVER THREAD²

It is a finer thread that can be used for a sewing machine. It has a Nylon core and it is easier to sew as it is thinner and more compact (not hairy). It breaks when it touches the soldering iron and it doesn't react to short circuits.



Figure 31. Plated silver thread

¹ Sparkfun, Conductive Thread (Thin) - 50' DEV-10118, 2012 - Retrieved 2013-04-05, <www.sparkfun.com/products/10118>

² Sparkfun, Conductive Thread 117/17 2ply DEV-08544, 2012 - Retrieved 2013-04-05, <www.sparkfun.com/products/8544>

1.6 SEWING

This process has followed some of the instructions of the e-Textile youtube channel¹. Some of the techniques and thread selection have been really useful in the construction of the glove.

The stainless steel thread was the first option as it was cheaper than the plated silver thread. They both have a resistance that can be negligible.

As solder could be applied to the stainless steel thread it was much easier to sew the connections with the devices. With the plated silver thread the connection needed to be done sewing a lot around the connection. The result was negative because the devices weren't stuck to the cloth and the quality of the connections changed when the fingers moved.

Sewing and working with the plated silver thread was much easier, as it isn't 'hairy' and much thinner, more like a textile thread. A lot of care had to be taken with the stainless steel thread to leave enough space between threads and to cut all the little hairs that could create an undesired connection between paths.

PARALLEL PATHS

When sewing parallel paths the method shown in Figure 4 is used whenever it is possible. This technique is done to reduce the risk of loose fibers from one path touching the other path.



Figure 32. Parallel paths sewing technique. The sewing technique is over (black) and under (grey).

SEWING THE CONNECTIONS

For the stainless steel thread, the connections are made sewing twice around the connection hole and then soldering. The ending of the thread can be cut very close to the solder material. At the beginning the solder was not used properly and the threads weren't stuck by the solder.

¹ Bruning, Lynne, eTextile Lounge, channel, YouTube

The technique (pull-and-solder) was improved by pulling the thread as the solder was applied. This way the thread was stuck in the solder and didn't slip away. This technique requires practice and is complicated. It involves pulling the end of the thread and applying solder with one hand as the other holds the soldering iron.

With the plated silver thread the connection has 4 or 5 times turns around. The connections were doing bad contact as the tension of the device with the threads changed. The connections were reinforced with more sewing. The end of the thread is sewn again to the fabric and finished with a typical nod on the fabric.

SEWING THE PALM OF THE HAND

The paths that are on the palm of the hand try to follow the natural folds of the hand. This avoided that when the hand closed bad connections were made. The advantage of using the stainless steel thread is that when voltage and ground threads make contact the fibers burn and create a hot point on the glove. This was an advantage because loose fibers were burnt if they were creating a bad connection.



Figure 33. Crossing sewing technique.



Figure 34. Parallel lines sewing technique.



Figure 35. Back of the right glove.



Figure 36. Palm of the right glove.



Figure 37. Back of the left glove.



Figure 38. Palm of the left glove

1.7 PROBLEMS

In this project, one of the goals was to design a functional and aesthetic glove. Using conductive thread could make a glove without standard electrical wiring. The use of conductive thread brought some problems.

The first problem was threading the connections. There are specific devices that have connection holes prepared for conductive thread¹. These devices are usually bigger and this could be an issue given that there isn't much space in the glove. In this project the devices are not prepared for conductive thread. The holes are small and very close to each other.

If a knot was made at the end of thread, the thread left was usually too long and could establish contact with the surrounding connections. Applying the technique 'pull-and-solder' made possible to cut the thread near the soldered connection, as the thread wouldn't slip away through the connection hole.

Some of the connections were made and the teensy didn't recognize the signal properly. After taking out the thread and sewing again some of them worked properly. Even if looking closely non errors could be seen when it didn't work. That might be cause by the tiny loose fibers of the stainless steel thread.

¹ Buechley, L., SparkFun Electronics, LilyPad Arduino, Retrieved 2013-04-12, <arduino.cc/en/Main/arduinoBoardLilyPad>

In some cases (little finger right hand, XBee) the voltage and ground circuits were very long. This fact added to a weak connection between threads made some of the signals unsteady and very variable. The XBee lost power randomly which made the communications not to work properly.

Sewing the connections took up to 6 hours of work per finger.



Figure 39. Glove with hook-up wires

Due a lack of time, the connections that weren't sewn properly were taken out and were substituted with hook-up wire¹. The glove achieved full functionality but lost the aesthetics.

In the next section the connection with the glove and the computer will be explained. Creating the proper communications is a very important step, as they will determine the data transmission rates.

¹ SparkFun, Hook-up Wire, 2012 - Retrieved 2013-06-05, <<https://www.sparkfun.com/products/8022>>

2 COMMUNICATION

The glove controller makes use of wireless communication architecture, made up of software, protocol and hardware components. This section will examine these components and how they can be set up, giving rise to a sturdy yet efficient wireless communication setup.

The devices used are the combination of an XBee S1 and the XBee Explorer Regulated. The last device only regulates the incoming voltage and gives visual signals of the communication processes. The XBee S1 is a device for peer-to-peer or point-to-multipoint connections. It uses the IEEE 802.15.4 networking protocol. The XBee used has a chip antenna integrated and it can reach up to 100m.

2.1 NETWORK STRUCTURE

The baud of the XBee, Uart and Serial is set at 38400 bps for the receiver device. The transmitters work at 19200 bps. The problem when receiving the data is that there is no synchronization between the two transmitters. The software isn't able to interpret the incoming data because the bytes are not in order.

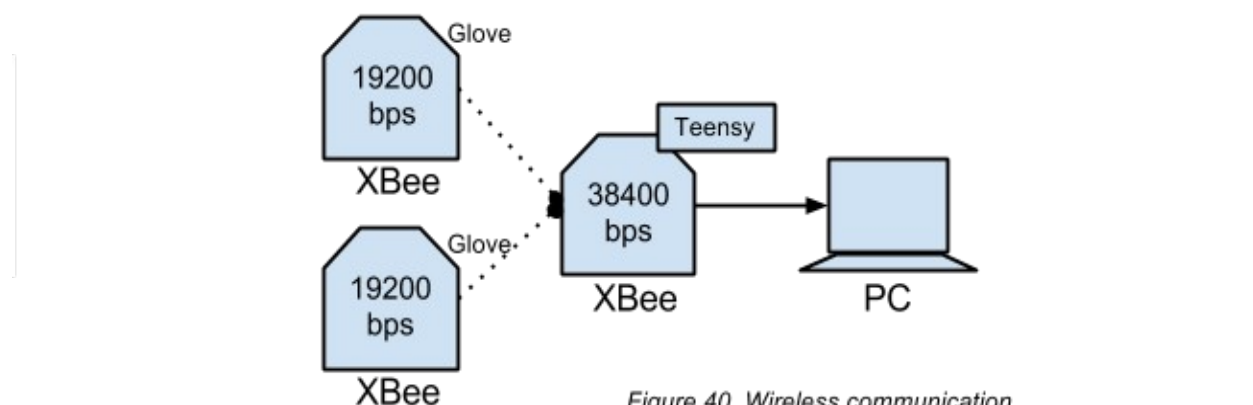


Figure 40. Wireless communication diagram. The gloves send the data to the receiver.

Nevertheless a communication protocol could be designed to fix this. For example: the first bit indicates the transmitter thus the data can be identified by the receiver and organized:

0 . xxxxxxx for the messages from the left hand and 1 . yyyyyyy for the ones from the right hand. This would reduce the quantity of usable bits to 7 because one would be used to indicate from which hand the message is coming from.

The other solution would be to send the data from one transmitter to the other and make only point-to-point connections. This way the data can come in order to the last receiver.

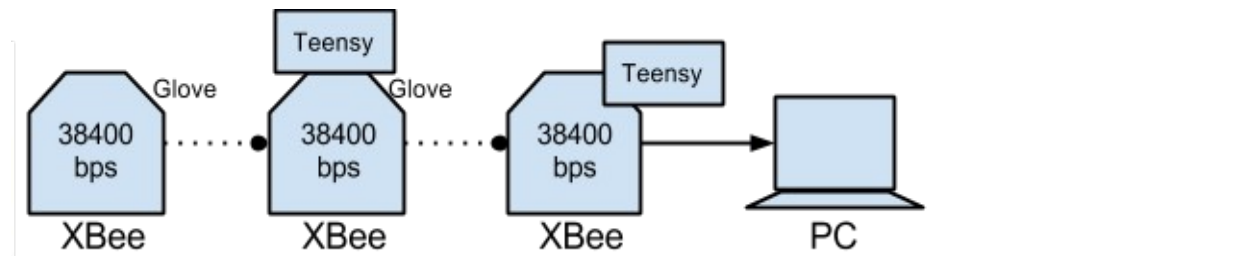
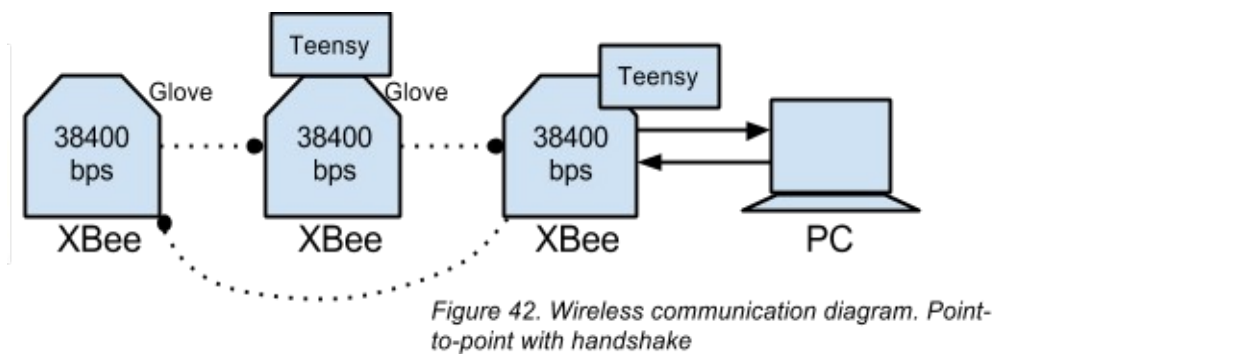


Figure 41. Wireless communication diagram. Point-to-point without handshake

This network is not designed for a protocol that sends confirmation messages, handshaking (communication messages) or requests more data. Using MIDI or Serial received by Pd-Extended¹ the data is not buffered and discarded if the program doesn't have time to process it. However if working with C++ the data is buffered and received with a big delay if the program doesn't run at the same speed as the incoming data.

To create request for more data once the buffer is free or almost empty a circular network can be designed as the XBee devices work as point-to-point. The first glove sends to the second, the second to the computer, the computer to the first glove and so on:

¹ Puckette, M., Pd-Extended, Pure Data, Retrieved 2013-02-02, <puredata.info/downloads/pd-extended>



CONFIGURING THE XBEEES

To configure the XBee modules it's necessary to use a program called X-CTU¹. It's not easy to configure the XBee. I tried first with Windows and I couldn't configure it properly. I could configure them with Linux using wine (which allows Windows software to run on linux).

The XBees are configured in the beginning all with the same addresses². The parameters that are modified in the XBee are the baud rates and the addresses. This are the parameters related to the network addresses:

ID - PAN ID: 16 bits for an identification number for the network.

DL - Destination Address Low: 16 bits for the destination address.

MY - Source Address: 16 bits for the source address.

The XBee devices are set up with this default parameters:

PAN ID: 3332

DL: 0

MY: 0

¹ Digi International Inc, X-CTU Software, 2012, Retrieved 2013-04-21, <www.digi.com/support/productdetail?pid=3352&osvid=57&type=utilities>

² Fakhri Hamad, O., Analog, Digital and Multimedia Telecommunications: Basic and Classic Principles, Xlibris Corporation

Now because there is a XBee that has to receive and send at the same time it is needed to change the configuration of the module. The first module only acts as a transmitter. It will be set like this:

DL: 01

MY: 0

The second one, that acts as a receiver and sender at the same time will have this settings:

DL: 10

MY: 01

The source address of the second XBee is receiving data from the destination address from the first device. The last device receives data from the second and is configured to send information to the first. The last one, that only receives will have this configuration:

DL: 0

MY: 10

First XBee	Second XBee	Third XBee
DL: 01	DL: 10	DL: 0
MY: 0	MY: 01	MY: 10

This will actually create a loop, as the last module has been configure to send data to the first XBee. This will be useful in case a handshake is needed.

The XBee hardware setup creates a useable wireless, multi-point network that is ready to be used for data transfer. However, in order for this to be efficient and fully functional, a suitable protocol will either have to be chosen or developed.

2.2 PROTOCOL

Implementing a set of rules for the messages that are going to be transmitted is essential to achieve high baud rates. The information needs to be space-efficient. Coding and decoding from a specific protocol will be part of this section, as well as developing the protocol.

MIDI¹

The greatest advantage of using MIDI is that there is no need to work with serial and thus avoid subsequent compatibility issues with drivers. The main problem that MIDI brings is that it has only 16 channels. If we are working with 4 accelerometers in each hand it means that we'll have 24 different signals. This could be fixed by fitting two signals in one MIDI message and then decoding it. Tests were done and the rate obtained was 250 samples per second when sending 6 signals (2 accelerometers) using Pd-Extended.

SERIAL²

The main advantage of using a serial connection is that a protocol can be developed, thereby allowing the data to be sent as specifically formatted messages. It is more flexible and gives option to more possibilities whereas MIDI messages have a structure, thus information has to fit into this structure.

The main problem is that some drivers have to be installed sometimes and the configuration might be difficult. The devices will come with different names depending on the computer and OS. Serial connections might be blocked by default and the user would have to manually set them up.

Each glove was simulated with one accelerometer to check functionality. The rate is 360 with 6 channels of information using Pd-Extended, which is superior than MIDI. This is as expected, as there aren't any redundant bits.

¹ MIDI Manufacturers Association Inc, MIDI Messages, Retrieved 2013-05-06

² Pinouts.ru, RS-232 and other serial ports and interfaces pinouts, 2009, Retrieved 2013-05-05, <pinouts.ru/SerialPorts>

Additional problems with serial include that Teensy uses the serial communication standard RS-232. This method has a software handshaking that uses the decimals 17 and 19 to send XON and XOFF (they indicate if the device is ready to receive or it's full). The standard also uses the decimal 3, 26 and 28 for other purposes. Those numbers cannot be used.

Different systems of coding the data have been proposed. This is just a brief mention of the different protocols proposed. The protocol used is explained in depth. The following paragraphs indicate the usage of bytes. Each byte contain 8 bits whose can be 0 or 1. Each bit represents different information inside a byte.

- Using 2 bytes: 1 byte to tell the channel where data is being transmitted and 1 byte to send the data.

 - x . yyyy. zz → x indicates type of byte, y indicates the channel, z is the variable to give more range to the data byte.

 - i . jjjjjj → i indicates type of byte, j represents the data.

The data can go from 0 to 512.

+Speed - Resolution +Byte control

- Using 3 bytes: same as MIDI but using one more bit for channels.

- Speed +Resolution +Byte control

- Using 5 bytes for a finger (3 streams):

 - xx . yyy . zzz → x indicates type of byte, y indicates channel, z used in third byte.

 - aa. bbb . ccc → a indicates type of byte, zzz, bbb, ccc variable to give more range to the following bytes.

 - i . jjjjjj → i indicates type of byte, j represents data.

 - i . jjjjjj → i indicates type of byte, j represents data.

 - i . jjjjjj → i indicates type of byte, j represents data.

+Speed +Resolution -Byte control

The fastest method is the last one and the used in the program. Only the first byte of the 5 will be affected by the software handshaking of the RS-232 serial communication standard because the rest will be bigger than 64. The bytes unusable will be: 00000011, 00010001, 00010010,

00011010, 00011100. This means that the problem will be when the data of the x axis of the third finger of the right hand is above 127 and below 384.

To resolve this, special bytes are going to be used:

00000011 (3) → usage: the value of the last seven bits of the third byte will be added to 128×3 → Substituted for 00000111 (7) → This byte would only be used hypothetically when the accelerometer value is above 128×7 (896)

00010001 (17) → usage: the value of the last seven bits of the third byte will be added to 128 → Substituted for 00010110 (22) → This byte would only be used hypothetically when the accelerometer value is above 128×6 (768)

00010001 (19) → usage: the value of the last seven bits of the third byte will be added to 128 → Substituted for 00010111 (23) → This byte would only be used hypothetically when the accelerometer value is above 128×7 (896)

00011010 (26) → usage: the value of the last seven bits of the third byte will be added to 128×2 → Substituted for 00011110 (30) → This byte would only be used hypothetically when the accelerometer value is above 128×6 (768)

00011100 (28) → usage: the value of the last seven bits of the third byte will be added to 128×4 → Substituted for 00011111 (31) → This byte would only be used hypothetically when the accelerometer value is above 128×7 (896)

Example of a simple message (coding and decoding):

Structure	Message	Coded message	Decoded message										
<table border="1"> <tr><td>00 . yyy . zzz</td></tr> <tr><td>01 . bbb . ccc</td></tr> <tr><td>1 . jjjjjj</td></tr> <tr><td>1 . jjjjjj</td></tr> <tr><td>1 . jjjjjj</td></tr> </table>	00 . yyy . zzz	01 . bbb . ccc	1 . jjjjjj	1 . jjjjjj	1 . jjjjjj	Channel 1 X axis: 192 Y axis: 127 Z axis: 385	<table border="1"> <tr><td>00 . 001 . 001</td></tr> <tr><td>01 . 000 . 010</td></tr> <tr><td>1 . 1000000</td></tr> <tr><td>1 . 1111111</td></tr> <tr><td>1 . 0000001</td></tr> </table>	00 . 001 . 001	01 . 000 . 010	1 . 1000000	1 . 1111111	1 . 0000001	Channel: 001 = 1 X axis: $001 * 128 + 1000000 = 128 + 64 = 192$ Y axis: $000 * 128 + 1111111 = 0 + 127 = 127$ Z axis: $010 * 128 + 0000001 = 128 * 2 + 0000001 = 384 + 1 = 385$
00 . yyy . zzz													
01 . bbb . ccc													
1 . jjjjjj													
1 . jjjjjj													
1 . jjjjjj													
00 . 001 . 001													
01 . 000 . 010													
1 . 1000000													
1 . 1111111													
1 . 0000001													

2.3 COMMUNICATION PROCESS

With these kind of networks bottlenecks can happen if they are not taken care of. In Figure 32 the bottleneck might happen in the receiver if the baud rates of the XBee devices in the gloves are too high. In Figures 33 and 34 the bottleneck might happen in the glove that has to receive data from the other glove and send all the data.

The first communication processes didn't involve any kind of handshaking from the final receiver. Using the network designed in Figure 33 the data from the first glove will be sent straight away by the second glove. Once the first glove finishes it sends a synchronization byte and the second glove sends its data.

The first glove will have a delay in its program to wait for the second glove to send the data. This delay can be adjusted to find the highest samples per second rate without compromising the data of the second glove.

The final communication process is done with C++, thus the diagram in Figure 34 is used. The communication process will be similar as the one explained above. Instead of having a delay in the first glove, it will wait for a synchronization byte from the final receiver. The final receiver will send this byte before the buffer in the serial connection is empty. This way the best baud rate will be achieved.

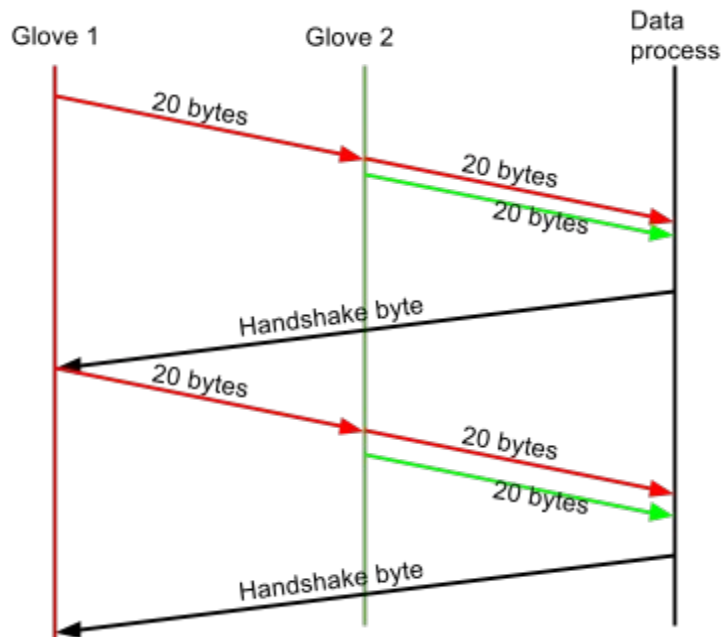


Figure 43.
Communication
process diagram

The advantage of serial communication is obvious if the size of the packets is compared with MIDI. The size of the network packets of this protocol is much smaller than the size of the MIDI messages needed to transfer the same information. MIDI would take three bytes to transmit only one stream of data. Each glove has twelve streams of data, meaning that in total MIDI would need 72 bytes.

The section above determines the protocol and communication channel that is used. This will have an impact in the software as it will have to decode the data accordingly. It is important to be able to reach high levels of transmission rates because the system will run in real time. The rate of the incoming data can always be slowed down if needed (the software might have difficulties processing too much information).

3 INTERPRETATION

Once the format in which the data is going to arrive is determined, the first step would be decoding the information to obtain the raw data. A filter might be useful to regulate the data. In this project an exponential moving average (EMA¹) is used to reduce noise. This filter averages the raw data to obtain a better state of the accelerometer.

This is one of the most important parts of the project, as the virtual possibilities of the glove will be determined by this section.

3.1 GLOVE DATA

Each accelerometer transfers data of the acceleration on each axis, X, Y and Z. Each glove has 4 accelerometers. That means that the glove will transfer 12 streams of data, and that two gloves will transfer 24 streams.

ORIENTATION OF THE FINGERS (static information)

When the hand is still, the accelerometers will measure the gravity. This will proportion information about the orientation of each accelerometer in relation to the gravity (floor). Orientation of the fingers can be achieved.

¹ Tham. M.T., Dealing with measurement noise (A gentle introduction to noise filtering), School of chemical Engineering and Advanced Materials, Newcastle University



Case 1		Case 2		
				
Thumb: X: 0.707*gravity Y: 0.707*gravity Z: 0	Index, middle finger, ring finger and pinky: X: 0 Y: - gravity Z: 0	Thumb: X: 0.707*gravity Y: 0.707*gravity Z: 0	Index and middle finger: X: 0 Y: gravity Z: 0	Ring finger and pinky: X: 0 Y: - gravity Z: 0

Figure 44. Information about orientation of each finger. The numbers are illustrative.

POSITION OF THE FINGERS (static and/or dynamic information)

With the orientation of each finger the position of the fingers in the hand is obtainable. The 3D vectors of each accelerometer will preserve the same relationship even if the position changes its orientation. Knowing the position of the fingers in the hand when it is still doesn't depend on the orientation. There will be 6 relations between fingers. This can be calculated by this formula:

$$\frac{N(N-1)}{2} = \text{number of connections}$$

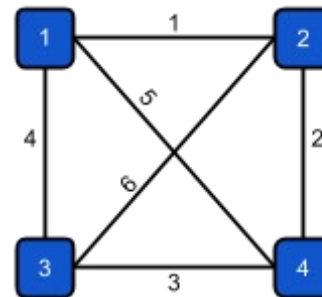


Figure 45. Relations between accelerometers

Where N is the number of accelerometers.

Even if the whole hand moves in the same direction the position of the hand will be know, as it would be the same as changing the gravity point. In Figure 11 the measured acceleration would represent the new gravity point.

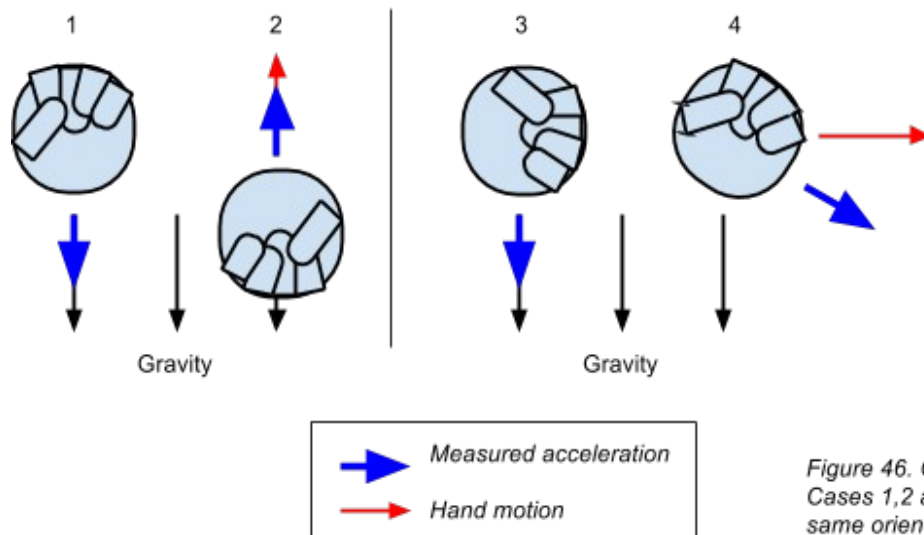


Figure 46. Gravity vs Acceleration. Cases 1,2 and 3,4 are perceived as the same orientation. The relation between fingers is the same.

ORIENTATION OF THE HAND (static information and user input)

To interpret the orientation of the hand it is needed to know the relation between a finger and the palm of the hand. The hand has to be still (static information). For example if the hand is in the position shown in the case 2 of Figure 9, the orientation of the hand will be the same as the middle finger.

This kind of information should be an input of the user when storing a static position of the hand. As shown in Figure 11 the orientation of the hand and fingers can be confused when the hand is in motion.

3.2 GESTURE RECOGNITION

In the previous cases, all the information that could be obtained was related to the static information of the accelerometers or the relation between themselves. Gesture recognition is about movements of the hand in the space in relation to time and how to identify them.

Recognizing the gestures will require certain steps and processes. The first step is to determine when a movement starts and ends. The next steps and processes will involve algorithms and signal processing that will classify and interpret the captured gestures.

START/END OF THE GESTURE

Most of the gesture recognition systems¹ have the user tell when the movement starts and ends. This facilitates the gesture recognition because the system won't have to be rejecting random gestures that are not identifiable. If the gesture recognition system is not having a user input, it will have to implement its own trigger or algorithm.

For example, one algorithm that could be used would be to use windows of the stream of data. For each window check if that can be the beginning of an identifiable gesture. This could mean that the algorithm should be able to process more than one gesture comparison at a time. Some other algorithms use the variance of the data of the window to know when a movement starts².

Figure 47 shows an example of windowing or segmenting the data. When a window or segmentation has a high probability of being the start of a gesture the algorithm checks the following windows to find if it is an identifiable gesture. In case 1 the system started recognizing the beginning of a gesture, but it is rejected as it doesn't match the consecutive windows or segmentations. In case 2 the gesture is identified.

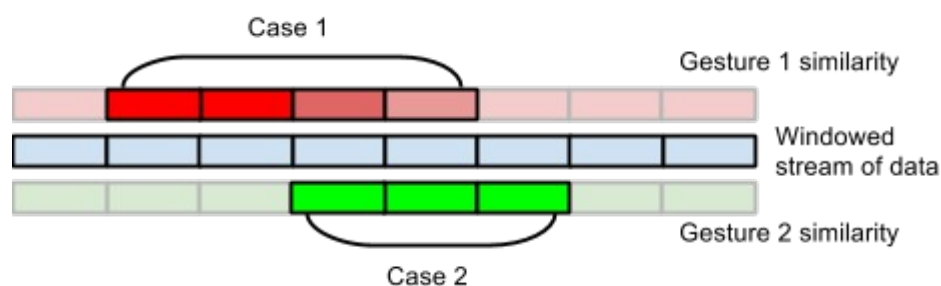


Figure 47. Data segmentation/windowing algorithm.

¹ Nintendo Wii video games, EasyStorke, Sony Ericsson Gesture sensing and others

² Benbasat, A.Y., Paradiso, J.A. An inertial measurement framework for gesture recognition and applications. *Gesture and Sign Language in Human-Computer Interaction, International Gesture Workshop, 2001.*

Usually gestures are initiated with a fast movement. The values of the accelerometer will change quickly. The peaks on the the derivative of the accelerometer data could be used as a trigger to start detecting a movement¹.

In our case the changes in the derivative of the accelerometer are used as a trigger. When the derivative is above a threshold it is considered that the gesture has begun, if it is below a threshold it is interpreted as the gesture is done.

This creates a problem about the range of gestures that can be identified. Some gestures will have a different motion speed that might not trigger the start or might trigger the stop too soon.

METHOD OF RECOGNITION

There are a lot of different algorithms that have been applied to recognize gestures: Hidden Markov Models^{51 52 2}, feature-based statistical classifiers³, neural networks, support vector machines², \$1 unistroke recognizer⁴, and others. The most commonly used is the Hidden Markov Model (HMM). The gestures are segmented in unities that are represented as states. The algorithm has different states and probabilities to go from one state to another. A gesture will be recognized when it has gone through the proper states in the right order. The main problem of the HMM is that the algorithm needs to be trained.

In this project one of the priorities of the gesture recognition is real time recognition. Also it is important for the user to be easy to record new gestures to identify in the future, so the application can be more personalized (custom gestures).

The algorithm applied in this project is the KNN (k-nearest neighbour). This algorithm doesn't need a lot of computation and doesn't need any training. Given a set of attributes for a gesture,

¹ Zoltan Prekopcsák (2008) Accelerometer Based Real-Time Gesture Recognition, *Budapest University of Technology and Economics*

² Anderson, D., Bailey, C. and Skubic, M. (2004) Hidden Markov Model symbol recognition for sketch-based interfaces. *AAAI Fall Symposium. Menlo Park, CA: AAAI Press, 15-21.*

³ Rubine, D. (1991) Specifying gestures by example. *Proc. SIGGRAPH '91. New York: ACM Press, 329-337*

⁴ O. Wobbrock, J., D. Wilson, A., Li, Yang (2007) Gestures without Libraries, Toolkits or Training: A \$1 Recognizer for User Interface Prototypes

the distances between attributes of the incoming gesture and the recorded ones will be calculated. The k-nearest neighbours of the incoming gesture will determine to which recorded gesture it is assigned. For example, if in Figure 13 a 3-nn algorithm is applied the incoming gesture will be classified as Gesture 1. If a 9-nn algorithm is applied it will be classified as Gesture 2, as there are more samples of the Gesture 2 closer to the incoming gesture. The KNN has its own problems, but if the samples are close to each other, certain issues are avoided. The algorithm is not recommended if a lot of dimensionalities are used.

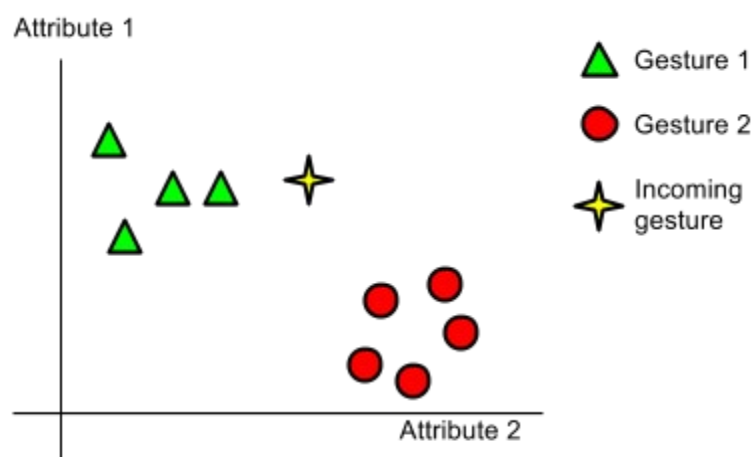


Figure 48. Gesture attribute points (2D) with an incoming gesture.

ATTRIBUTES

The attributes of each stream of data will be low-feature attributes. They are fast to calculate and they give the necessary information about the signals¹. These attributes are:

- Energy: the total sum of values of the signal.

$$E = \sum_{i=1}^N s(i);$$

¹ Peeters, G., Rodet, X. (2002) Automatically selecting signal descriptors for Sound Classification

- Log-Attack Time: represents the time between the minimum and the maximum value in a logarithmic scale.

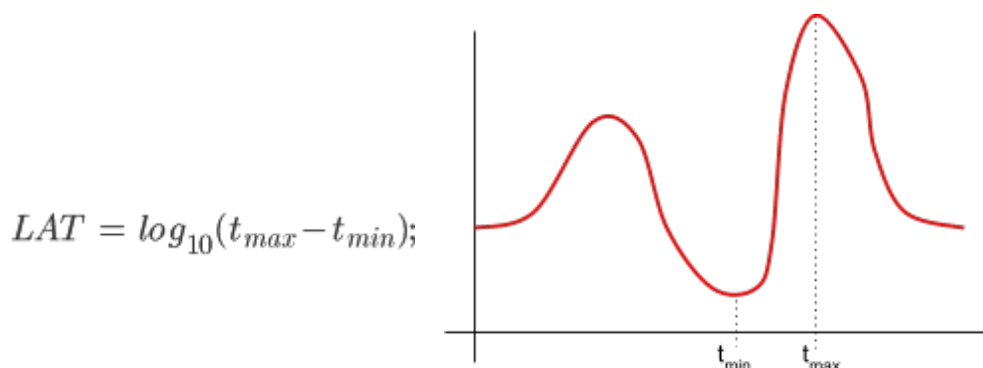


Figure 49. Peak and valley of a signal

- Temporal Centroid: is the weighted average of the instantaneous temporal envelope of the signal. It indicates where the signal has more energy.

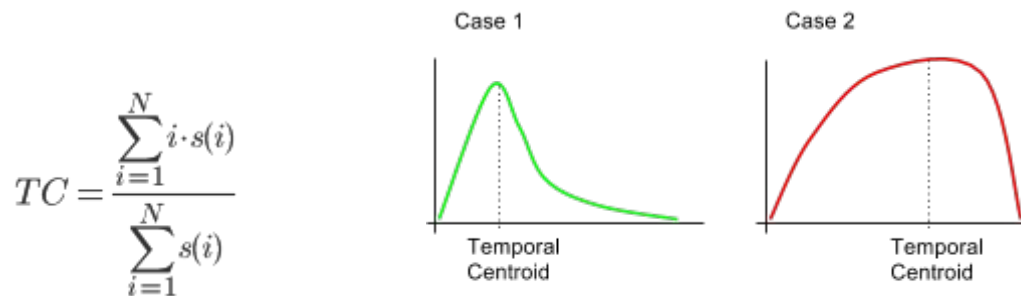


Figure 50. Temporal centroids of two signals

- Zero-Crossing Rate (ZCR): calculates how many times the signal goes from positive to negative. As the signal from the accelerometer will never cross zero this is calculated to its derivative.

This attributes were selected by experimentation. Gestures were recorded and stored. For each gesture 20 samples were used. 5 different gestures were used. The signals were upsampled and interpolated to make them similar to an audio signal. The Timbre Toolbox¹ library for Matlab extracted several different signal descriptors in a .csv file.

Once obtained all the descriptors the data is processed with Weka². Different classification algorithms are tested. The KNN is faster than SVM and mostly equally efficient. The attributes more often used are the energy and the temporal centroid, as well as others.

The efficiency of the algorithm is proven again creating a .csv file with the signal attributes directly from the program, without using the Timbre Toolbox, upsampling and Matlab. The results are about an 80% of accuracy with Weka and the KNN classification method. The gestures that are misclassified could be avoided with a similarity threshold and rejecting them.

The KNN algorithm is not recommend to be working with high dimensionality data. Our case has 12 streams (4 fingers with 3 axis each) and each stream has 4 attributes. Each gesture sample has 48 dimensions. Computationally speaking 48 difference operations will have to be done to compare the incoming signal with one sample. If there are 10 different gestures with 20 samples each 9600 operations will have to be done for every incoming gesture.

To reduce computations a CNN Data Reduction algorithm can be applied. This algorithm takes away the samples that are not in the conflict with other samples or areas when a 1NN is applied (Figure 51).

¹ Peeters, G., Giordano, B., Susini P. and Misdariis N., McAdams, S. (2010) The Timbre Toolbox: Extracting audio descriptors from musical signals

² University of Waikato, Weka 3: Data Mining Software, 2013, Retrieved 2013-05-15 <www.cs.waikato.ac.nz/ml/weka/index.html>

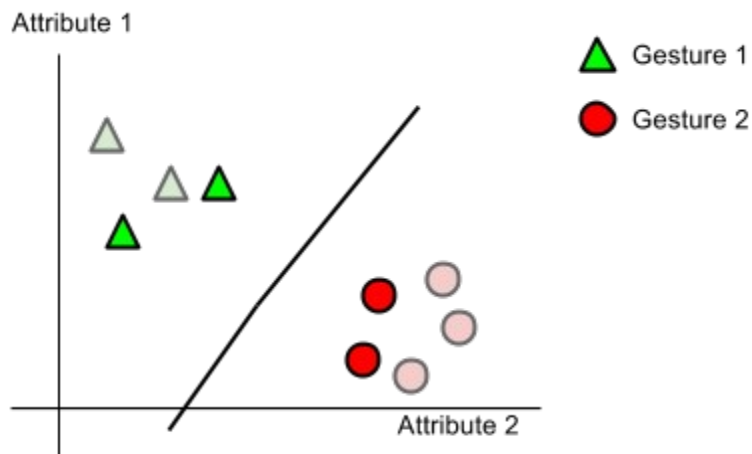


Figure 51. CNN Data reduction. The pale samples are discarded when running the knn algorithm.

All the processing and calculations about interpreting the movements are implemented in the software. The knn algorithm doesn't require high-complexity mathematics. This is an advantage when programming, as it will be less complicated to write the code. The following section goes through the different interfaces and software used.

4 SOFTWARE

This section will talk about the software used in the project to design a user-end interface. The software used for testing was mentioned briefly in the section above (Matlab and Weka).

The software have to be able to create graphics, make connections with Teensy, design a GUI, reproduce sound or MIDI messages, allow music interactivity, do fast computations for the recognition algorithm, write and load presets.

4.1 PD-EXTENDED & PROCESSING¹

Initially, the intended software integration was planned for Pd-Extended (audio engine) and Processing (computations). Pd-Extended is a visual programming language that allows to create audiovisual interaction and computing. Processing is a programming language based on Java design to create visual effects and media art with simple computer programming.

The first steps had the channels of communication shown in Figure 16.



Figure 52. First communication setup. Teensy transmits through MIDI to pd-extended. Pd-extended and processing communicate via OSC messages.

Pd-Extended interprets the MIDI data and sends the raw accelerometer data to Processing via OSC messages. Processing is in charge of filtering the data, creating the graphic engine and interpreting and recognizing positions and gestures. Processing can send OSC message to Pd-Extended to create audio.

Using serial instead of MIDI gave better results in terms of the baud.

¹ Reas, C., Benjamin, F., Processing, MIT Media Lab, 2013, Retrieved 2013-02-02 <www.processing.org/>

The option of using these two programs was discarded as the coding started to be more complex. The IDE of Processing is not suitable for having complex programs. It makes use of Java, which is slower than C++. These programs were really useful at the beginning, as they are not very complicated to program with. As final applications they didn't work, as they cannot export a final application.

The program reached the stage of filtering and storing the data, recording and storing gestures, showing graphically the inputs and triggering sound samples. When showing the input data with processing the rates drastically dropped.

4.2 OPENFRAMEWORKS¹

Initial attempts were made with raw C++. It was very hard to connect the teensy via serial and there were some unsolved problems that stopped the programming at that moment (RS-232 usage of bytes). Working with Openframeworks made things clear, as there were examples of how to do serial communications.

One of the many advantages of working with Openframeworks is that it has a similar structure as Processing, but it works with C++. Programming with C++ also allows the program to be able to work in different platforms if it is specified in the code and run faster. It also can deliver final applications, so the user doesn't need to open any extra software.

As the code grew bigger, every change in the overall structure was more complex to make and coding became slow in the last steps, as it had to fit the structure and many variables and functions had to be recoded to fit in the structure. In this project the IDE Code::Blocks is used to write and debug the code.

¹ Lieberman, Z., Watson T., Castro, A., openFrameworks, MIT License, 2013, Retrieved 2013-04-25 <www.openframeworks.cc/>

4.3 ARDUINO¹ AND TEENSY LOADER²

The microcontroller Teensy is programmed with the Arduino IDE and Teensy Loader. The programming language is simple and fast to code.

The first Teensy will be programmed to receive the data from the computer, send a synchronization byte to the glove and send the data received from the gloves back to the computer. A delay controllable delay is coded to control the bandwidth.

The code from the second Teensy will code the information from the sensors and send it to the second glove when the synchronization byte is received.

The program from the microcontroller on the second glove will encode the information from its sensors and send it together with the information of the first glove.

Coding the Arduino is a fast process and the code is very flexible, as it isn't very long and complex.

¹ Banzi, M., Cuartielles, D., Zambetti, N., Arduino IDE, Arduino, <www.arduino.cc>

² Coon, R., Stoffregen, P., Teensy Loader, PJRC Electronic Projects Components Available Worldwide, <www.pjrc.com/teensy/>

5 INTERFACE

It is important to have an easy interface to interact with. The final user should be able to understand the functions of each section of the program by only looking the icons, without having too much confusion about what they could mean.

Once this point had been reached in the project, the options of what can be done in terms of mapping are very open. The data is being received properly and processed. The interaction phase begin now, by programming new interactive features of the program and giving more functionalities to the gloves.

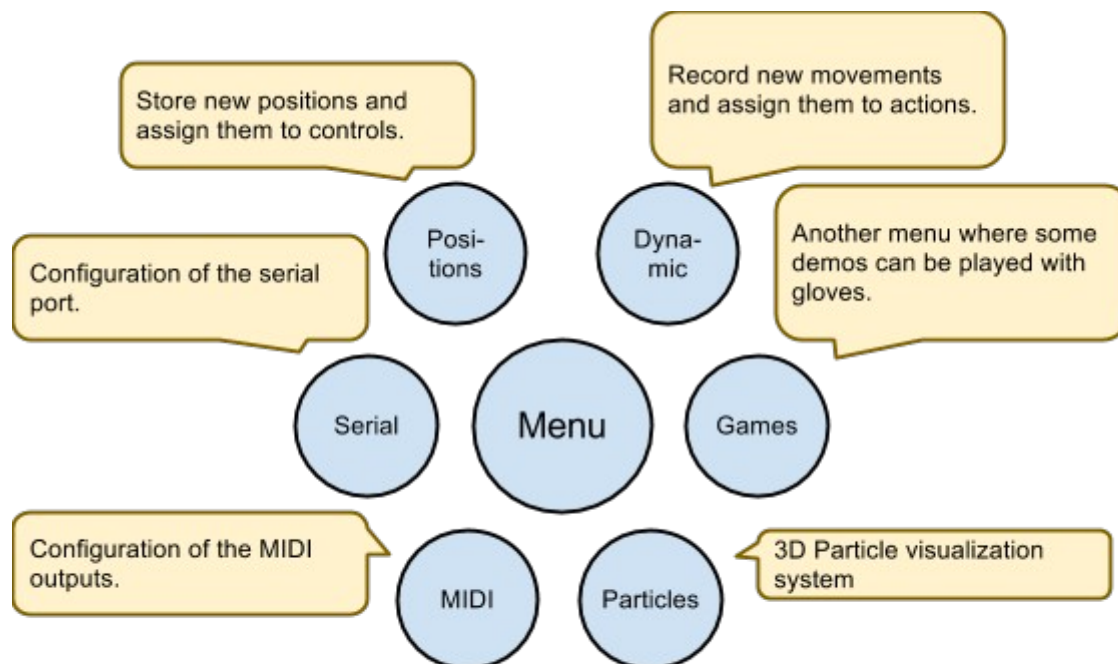


Figure 53. Interface structure of the program

5.1 FUNCTIONS OF THE PROGRAM

There are several functions in this program. It is just a demonstration to show what the glove could be capable of. The icons and buttons are relatively big because the mouse control with the glove can be difficult to operate at first.

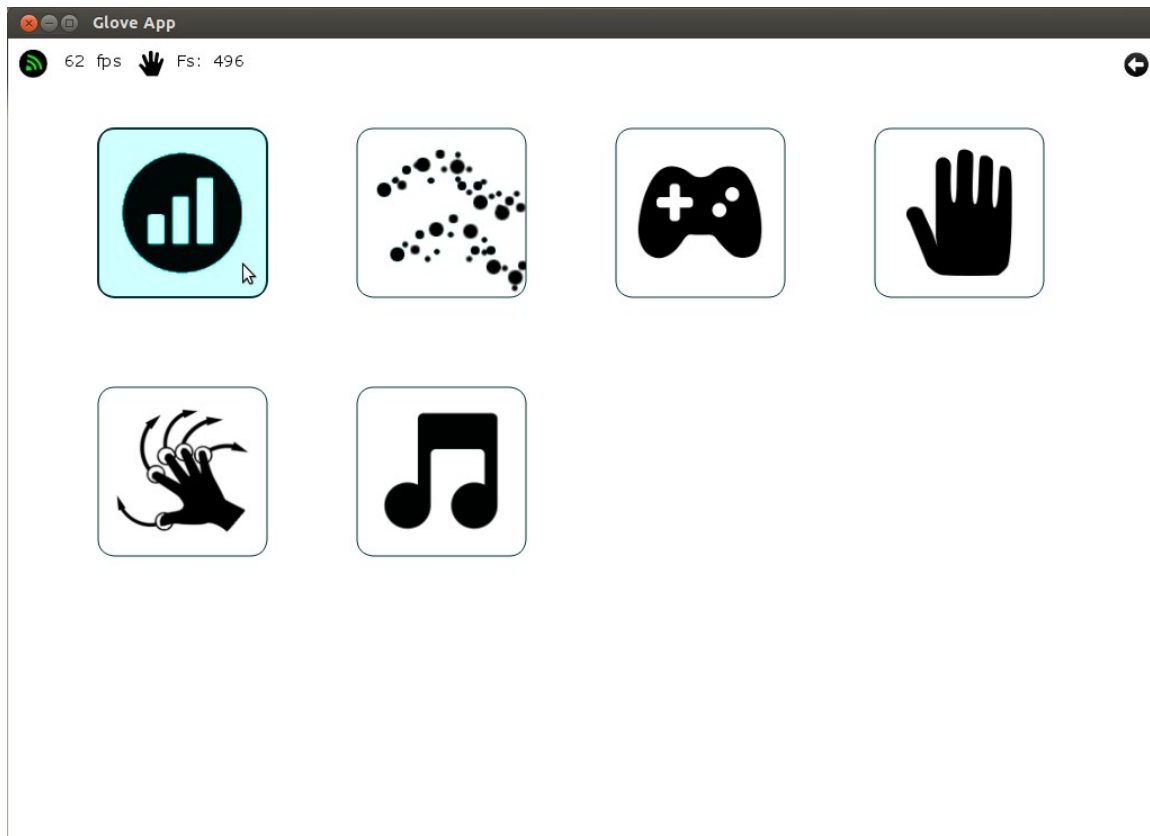


Figure 54. Main menu

The first option is to configure the connection. Depending on the computer the name of the serial port device will change. The interface allows to write the name of the serial port. If the connection is made the box will be green and if it is disconnect it will be red.

The bandwidth can be incremented or decreased with the plus and minus buttons. The rates can be 125, 340, 780, 880, 1000 and 1500 (these numbers are approximate, as the rates are not steady).

The buttons down below are to go back, save the configuration and load configuration presets.

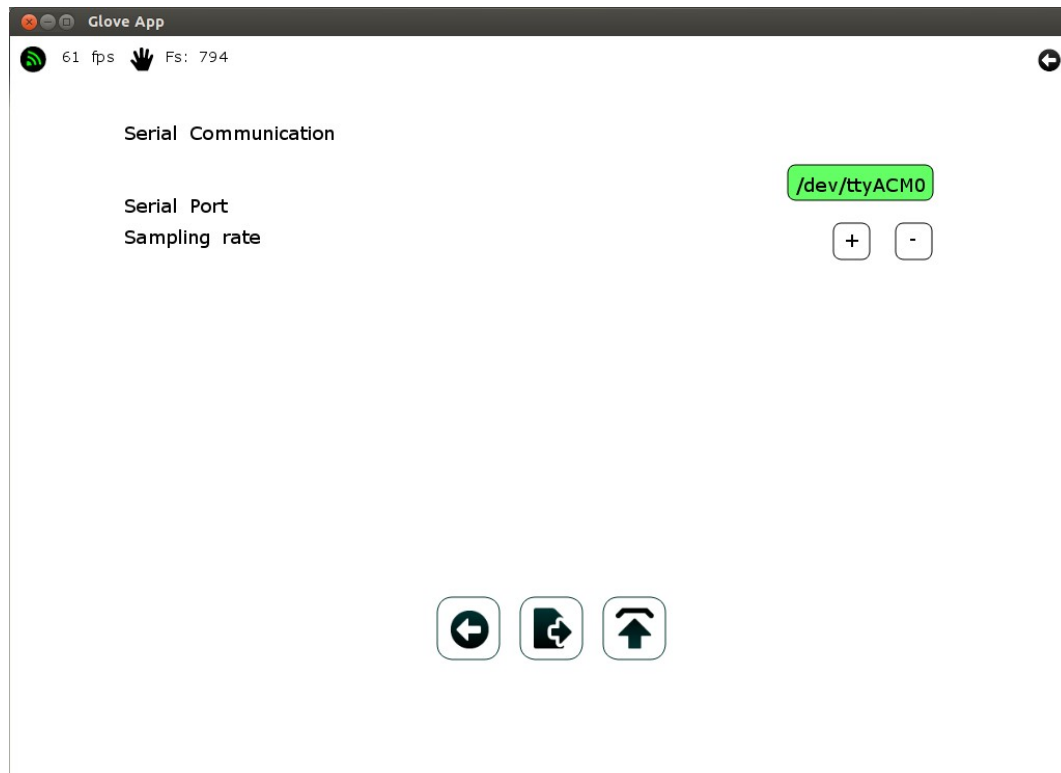


Figure 55. Configuration of the serial port

The second option in the main menu shows the data of the index finger and the middle finger in 3D with a particle system. This is a visualization tool, but is just an example of how particle systems work. The camera can be rotated with the mouse to see the data from any perspective.

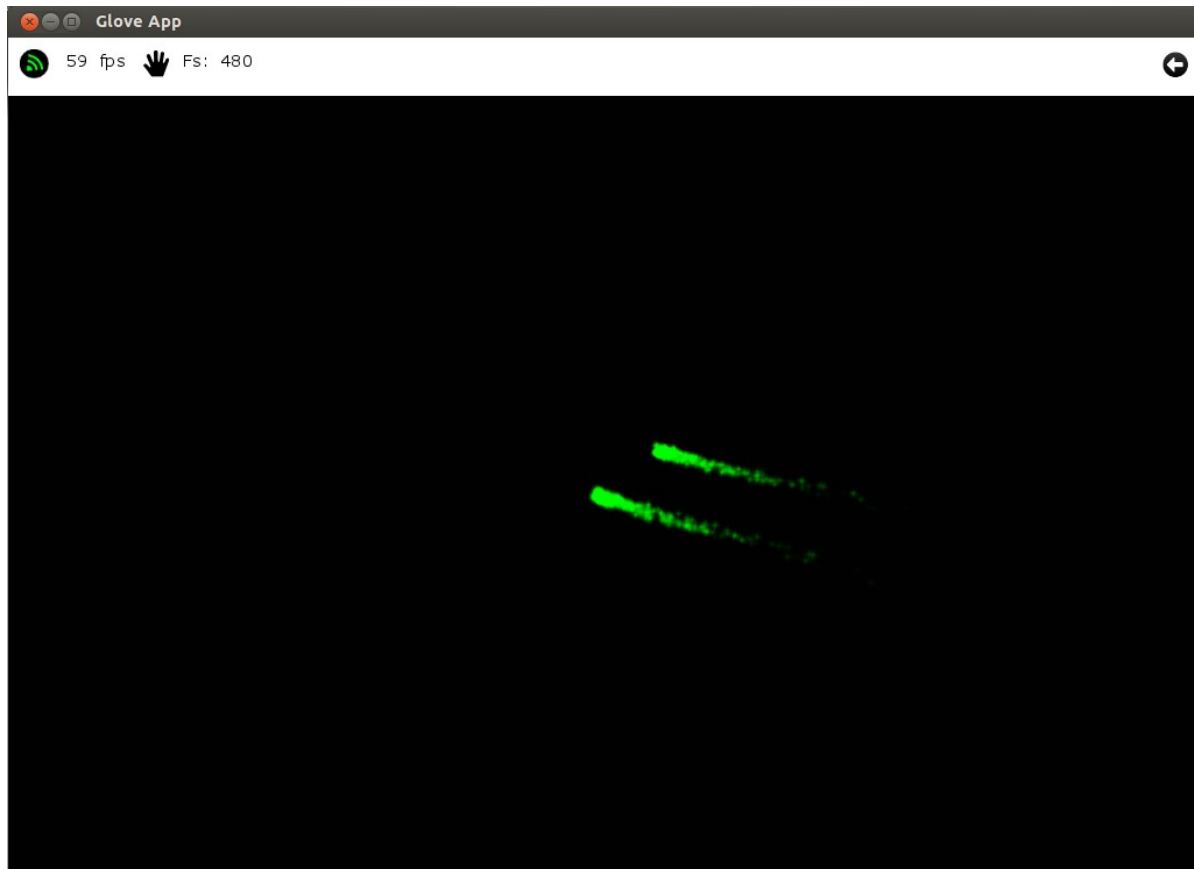


Figure 56. Particle system

Games is the third option. In this demonstration only one game has been implemented. The index finger controls the acceleration of a ball (X and Y axis). The objective of the game is to avoid the holes and reach the exit (red dot). The physical movement of the ball is based on a board game¹. There is a functionality in the upper bar to regenerate the holes, as sometimes there isn't a way to the exit. The game can reach up to level 20.

¹ Toysmith, Labyrinth Board Game, 2013, Retrieved 2013-06-10 <www.amazon.com/Toy-Smith-Labyrinth-Board-Game/dp/B000BXM838/>

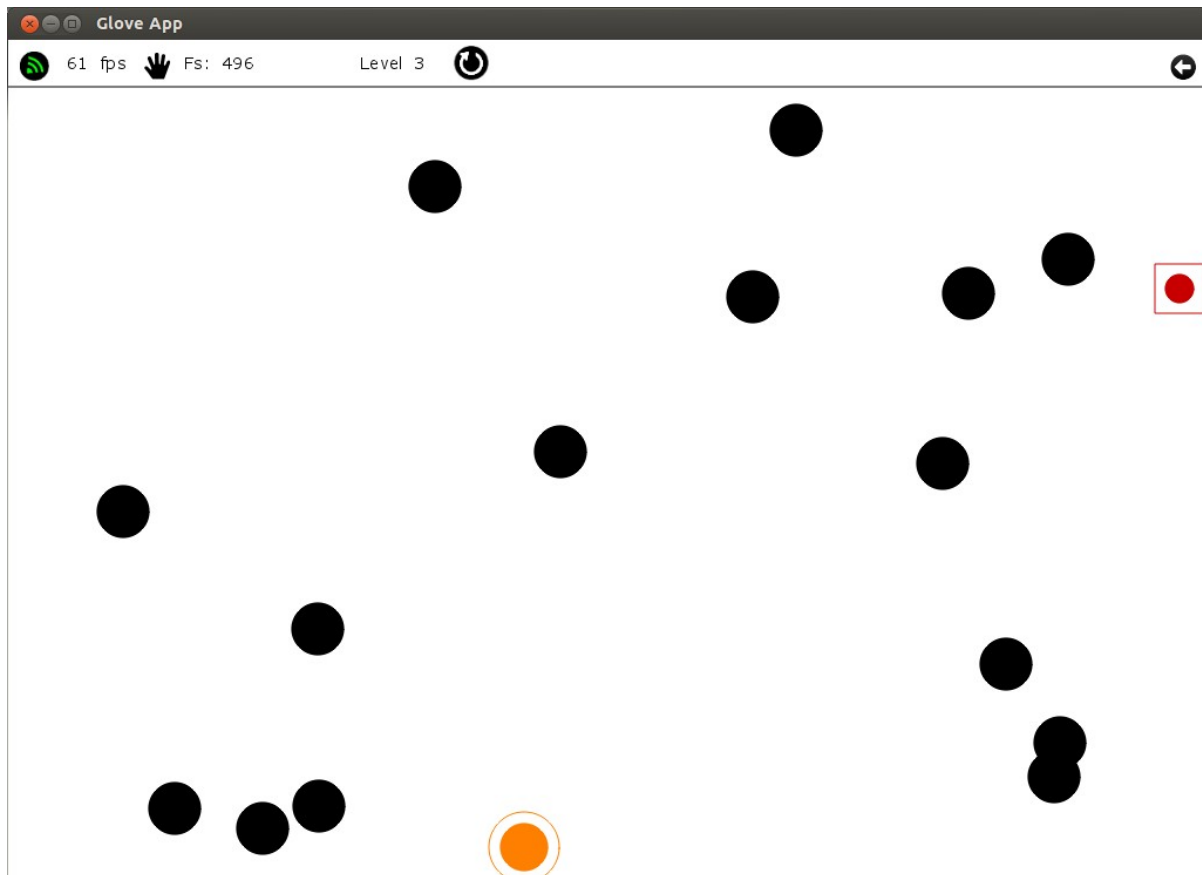


Figure 57. Labyrinth Game

The next menu has the functionality of storing and recognizing hand positions. The relation between the fingers is represented with circles in the bottom. The order of these relations between fingers is index-middle, middle-pinky, pinky-thumb, thumb-index, index-pinky, middle-thumb. The red lines represent the stored position and the black the actual one.

Positions can be stored and saved as presets, so when the program starts it will load the positions previously stored. They are stored in an external XML file. The buttons down below do the functions of going back, saving, storing presets and loading presets.

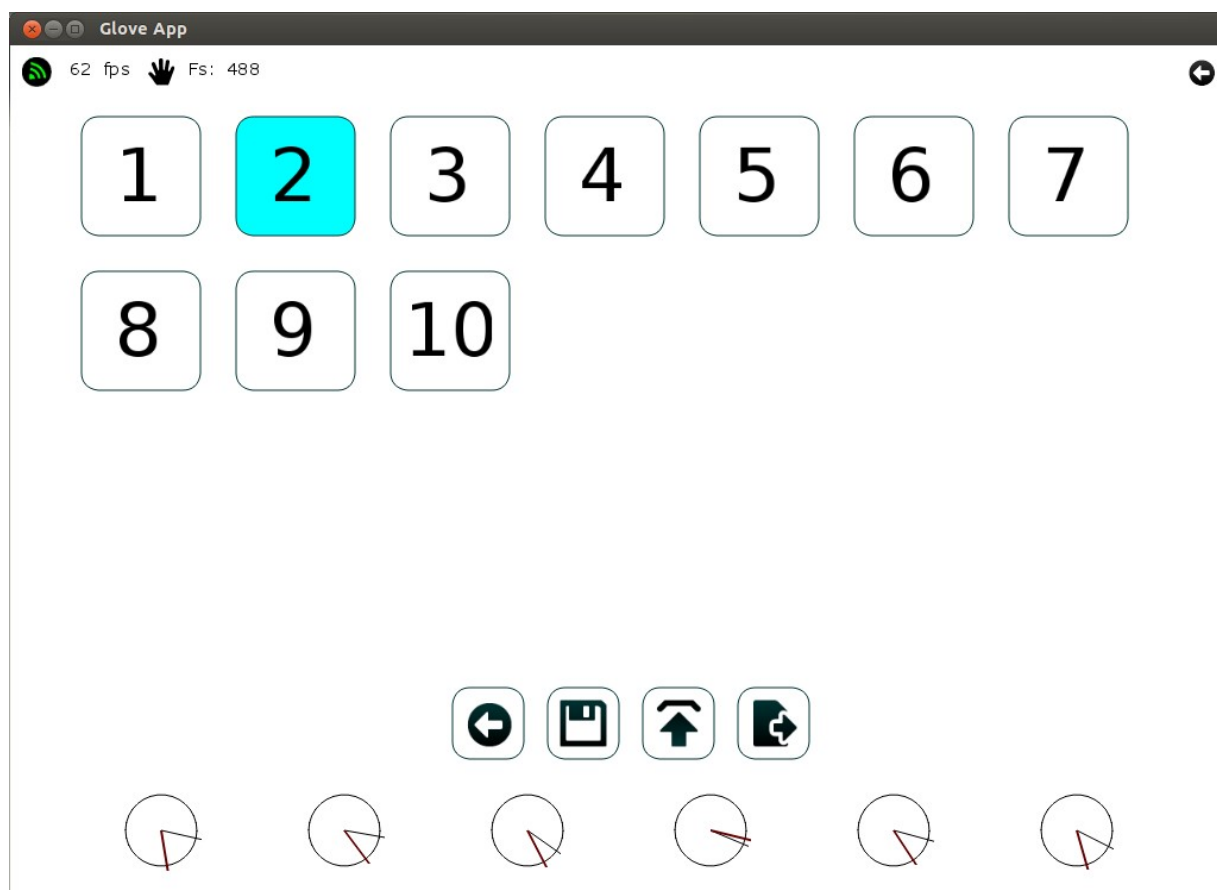


Figure 58. Position storing and loading

The gesture recognition is implemented in the next menu. The menu is very similar to the previous one. Instead of having circles to see the relationship between fingers, this time there is a data input representation. The green lines represent the filtered incoming data. The red lines are the derivative of the green lines. The rows have the order of index, middle, pinky and thumb, and the columns represent Z, Y and X axis.

The buttons down below do the functions of recording, going back, saving, storing presets and loading presets. The recording will be blinking when the program is set to record and it will be red once is recording.

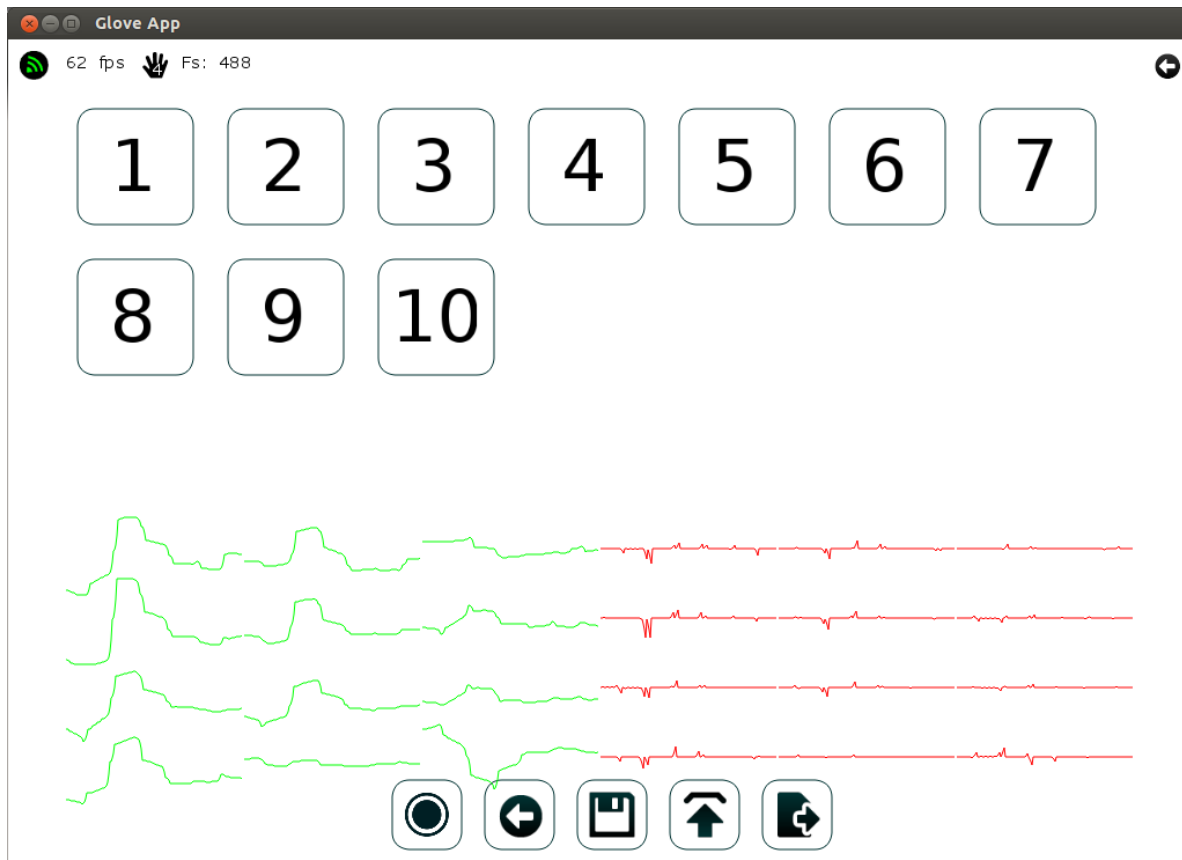


Figure 59. Gesture recording, storing and loading

The last menu is related to MIDI. There are four functionalities of MIDI implemented: NoteOn, two ControlChange and PitchBend. The user can choose the axis of the finger as MIDI message. The data will have a range between 0 and 127. The order of the fingers is index (f0), middle (f1), pinky (f2) and thumb (f3). Audio interaction could be implemented in other programs receiving the MIDI information from this program.

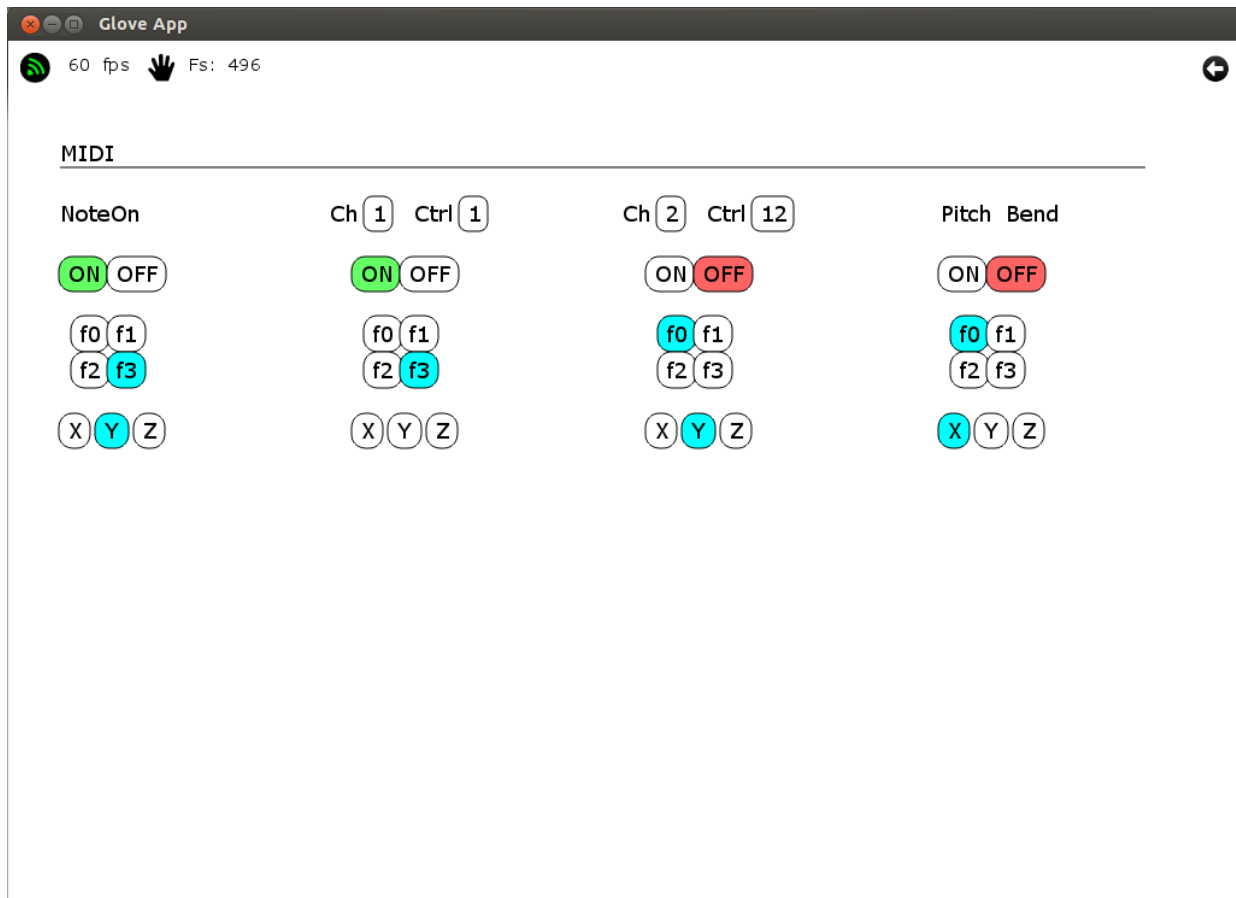


Figure 60. MIDI data selection

The upper bar indicates in all moment if there is communication between the glove and the computer. It indicates the frame rate, the position in which the glove is now and the sampling frequency. On the right there is a go back button, that will return to the previous menu if clicked. The connectivity button can also be clicked. It will show the Serial Configuration menu. The connectivity button will go red if the connection is lost.



Figure 61. Upper bar (left)



Figure 62. Upper bar (right)

The main objective of this demonstration program is to create an interface easy to use and play with. Most of the basic functionalities of the glove are implemented in this demonstration.

CONCLUSION AND RESULTS

This paper has gone through all the aspects of the construction of a prototype of glove controller, the communication process with the computer and the information that can be extracted from the data from the accelerometers.

The process of sewing the connections of the glove has been the most tedious and time inefficient. A lot of problems were driven from bad connections and had a high cost in time. Due lack of time the connections were done with hook-up wires. Aesthetics were lost, but a lot of time was gained to work in other parts of the project.

This project has focused a lot on the communication process. The glove could be used in different situations because the bandwidth rate can be modified. It can reach baud levels that none of the gloves studied before do. This is a large advantage when working with real-time systems, as they require a fast response from the glove.

The motion of the hand can reach high frequency movements in relation of other parts of the body. If the sampling rate was lower a lot of information would be lost. For example, the Aceleglove works at 35 Hz and movements of the hand can reach the 30 Hz (tests were done with this glove at 780 Hz). With the Aceleglove all this information would be lost, even if the best software was used.

Accuracy is also a very important aspect, and as more samples better the accuracy. A noise reduction filter has been applied in our system to obtain a smooth curve. Having a lot of samples to process has made the signal smoother and more coherent.

In this report the coding part is not explained, but it has a lot to do with the whole project. To receive the data a second computational thread¹ has been developed, so it can work concurrently with the rest of the program. Treating the serial connection as an event was coded also, but the events only were executed once every frame, so only 60 Hz could be achieved

¹ David R. Butenhof, *Programming with POSIX Threads*, Addison-Wesley

following the execution of the program. An improvement could be done, making the event ask repeatedly for data, but that would still be gathering data every 1/60 of a second.

Designing the particle system and keeping the frame rate was not obvious. About 3000 small pictures had to be rendered at a time. To achieve a better frame rate pointers have been used to load and render images of the particle system.

Regarding the interface, it has been object oriented designed. The menus are objects that contain buttons. Working with objects has made the coding much smaller and easier to develop. The labyrinth game has been developed with object oriented programming as well as the particle system.

This is just the beginning, as there is plenty of room for development, expansion and further refinement. Once the connections and the data from the glove is properly received by the computer all the interpretation and interaction can start.

APPENDIX A - Source Code

This appendix contains all relevant source code for the hardware and software portions of the project. The source codes used in the initial phases had not been included as they were not developed any further.

TEENSY

Computer Teensy

```
HardwareSerial Uart = HardwareSerial();
elapsedMillis timer;
int delayBaud;
byte serialIn;
boolean noDelay=false;

void setup() {
  Serial.begin(38400);
  Uart.begin(38400);
  pinMode(11,OUTPUT);
  delayBaud=10;
}

void loop() {

  if (Uart.available()){
    digitalWrite(11, HIGH);
    Serial.print(Uart.read(),BYTE);
  } else digitalWrite (11,LOW);

  if (timer>delayBaud&&!noDelay){
    if (Serial.available()){
      serialIn=Serial.read();

      switch (serialIn){
        case '0':
          Uart.print(serialIn,BYTE);
          break;
        case '1':
          delayBaud=delayBaud+1;
          serialIn='3';
          break;
        case '2':
          delayBaud=delayBaud-1;
          serialIn='3';

```

```
        if (delayBaud<7){ delayBaud=7; noDelay=true;}
        break;
    }
}
timer=0.01;
} else if(noDelay){
    if (Serial.available()){
        serialIn=Serial.read();

        switch (serialIn){
            case'0':
                Uart.print(serialIn,BYTE);
                break;
            case '1':
                delayBaud=delayBaud+1;
                if (delayBaud>6){noDelay=false;}
                serialIn='3';
                break;
            case '2':
                delayBaud=delayBaud-1;
                serialIn='3';
                if (delayBaud<7){ delayBaud=7; noDelay=true;}
                break;
        }
    }
}
}
```

Glove Teensy

```
HardwareSerial Uart = HardwareSerial();

int sens;
int sensfix[12];
int fix[12];

void setup() {
  Uart.begin(38400);
}

void loop() {

  if (Uart.read()=='0'){
    for (int i=0; i<12; i++){
      if (i<11) sens=analogRead(21-i); // A11 is 22 and A0 is 21
      if (i==11) sens=analogRead(22);
      fix[i]=floor(sens/128);
      sensfix[i]=sens%128;

      // if (i%3==0) MORE EFFICIENT TO WRITE Serial.print IN HERE
    }

    sens=analogRead(A9);
    fix[9]=floor(sens/128);
    sensfix[9]=sens%128;
    sens=analogRead(A10);
    fix[10]=floor(sens/128);
    sensfix[10]=sens%128;
    sens=analogRead(A11);
    fix[11]=floor(sens/128);
    sensfix[11]=sens%128;

    for (int i=0; i<4; i++){

      if (i*8+fix[3*i]==3) fix[3*i]=7; //Avoid using 3 (Byte used by RS-232)
      if (i*8+fix[3*i]==17) fix[3*i]=6; //Avoid using 17 (XON byte of RS-232)
      if (i*8+fix[3*i]==19) fix[3*i]=7; //Avoid using 19 (XOFF byte of RS-232)
      if (i*8+fix[3*i]==26) fix[3*i]=6; //Avoid using 26 (Byte used by RS-232)
      if (i*8+fix[3*i]==28) fix[3*i]=7; //Avoid using 28 (Byte used by RS-232)

      Uart.print( 0 * 64 +      i*8      + fix[3*i] , BYTE); // 00 . yyy . zzz
      Uart.print( 1 * 64 + fix[3*i+1]*8 + fix[3*i+2] , BYTE);
// 01 . bbb . ccc
      Uart.print( 128 + sensfix[3*i] , BYTE); // 1 . jjjjjjj
      Uart.print( 128 + sensfix[3*i+1], BYTE); // 1 . jjjjjjj
      Uart.print( 128 + sensfix[3*i+2], BYTE); // 1 . jjjjjjj

    }
  }
}
```

OPENFRAMEWORKS

main.cpp

```
#include "ofMain.h"
#include "testApp.h"
#include " ofAppGlutWindow.h"

//=====
int main( ){

    ofAppGlutWindow window;
    ofSetupOpenGL(&window, 1024,720, OF_WINDOW);// <----- setup the GL context

    window.setGlutDisplayString("rgba double samples>=4"); // Smooth

    //window.setGlutDisplayString("rgba double depth samples>=4");
    window.setGlutDisplayString("rgba double depth alpha samples>=4");

    // this kicks off the running of my app
    // can be OF_WINDOW or OF_FULLSCREEN
    // pass in width and height too:
    ofRunApp( new testApp());

}
```

test.cpp

```
#include "testApp.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <unistd.h>

#include <X11/Xlib.h>
#include <X11/Xutil.h>

//-----
void testApp::setup(){

    //Initialize variables

    width=ofGetWidth();
    height=ofGetWindowHeight();

    iniGraphics();
    iniCSV();

    menu_sel=0;
    menu_prev_sel=0;
    samplingfreq_ind=0;
    samplingfreq=0;

    mPositionSel=-1;
    mGestureSel=-1;

    for (int i=0; i<4;i++){
        mMidiFingerSel[i]=-1;
        mMidiAxisSel[i]=-1;
        mMidiOnOffSel[i]=-1;
    }

    //Initialize record variables

    record_state=0;
    tSerial.gestLength=0;
    numclass=0;
    tSerial.input_gest=false;
    tSerial.input_gestLength=0;
    //_____

    // Communications _____

    // Serial communication
```



```
loadSerialPort();

/*#ifdef _WIN32
    strcpy(device, "COM4");
#endif
#ifdef linux
    strcpy(device, "/dev/ttyACM1");
#endif
*/
// Try to connect to the serial port
connectSerial();

// Midi
midiOut.listPorts();
midiOut.openPort(0);

// Audio
explosion.loadSound("explosion.wav");
explosion.setVolume(0.75f);

// Mouse
mouseActive=false;
inClick=true;
}

//-----
void testApp::connectSerial(){

// Try the connection and send a sync byte.
tSerial.lock();
if(serial.setup(device, 38400)) {
    serialConnection=true;
    if (tSerial.isThreadRunning()==0){
        if (serial.available()<25){ serial.writeByte('0'); }
        tSerial.start(serial);
    }
}
else serialConnection=false;
tSerial.unlock();
}

//-----
void testApp::update(){

if (mouseActive){
    moveMouse();
}
```

```
        if(tSerial.input_gest&&inClick){
            inClick=false;
            clickMouse();
        } else if (!tSerial.input_gest){
            inClick=true;
        }
    }

    // Serial Communication
    if (serialConnection){
        // Close the connection after 60 attempts.
        tSerial.lock();
        if (!serial.available()) {
            attempts++;
            if (attempts>10000){
                serial.flush(); serial.close(); serialConnection=false;
tSerial.stop();
                cout << "Serial Connection closed\n";
            }
        } else attempts=0;
        tSerial.unlock();
    } else{
        // Retry connection once every second
        if(prev_sec!=ofGetSeconds()){
            connectSerial();
        }
        prev_sec=ofGetSeconds();
    }

    // Update data
    tSerial.lock();
    if (serialConnection){
        // Record
        gestAnalysis();
    }
    tSerial.unlock();

    // Sampling rate
    if(prev_sec!=ofGetSeconds()){
        //tSerial.lock();
        samplingfreq=tSerial.samplingfreq_ind;
        tSerial.samplingfreq_ind=0;
        //tSerial.unlock();
    }
    prev_sec=ofGetSeconds();

    //Midi
    sendMidi();

    // Audio
```

```
    ofSoundUpdate();
}

//-----
void ofApp::draw(){

    ofBackground(255, 255, 255);

    // Draws the menu selected
    switch (menu_sel){
        case 0:
            ofSetLineWidth(1);
            myMenu.draw();
            break;
        case 1:
            drawSerialConfiguration();
            break;
        case 2:
            drawParticles();
            break;
        case 3:
            drawGBouncingBalls();
            break;
        case 4:
            drawPositions();
            break;
        case 5:
            drawGesture();
            break;
        case 6:
            drawMidi();
            break;
    }

    // Connection Icon
    if (serialConnection) ofSetColor(0,255,0);
    else ofSetColor(255,0,0);
    ofFill();
    ofCircle(22,22,12);

    ofEnableAlphaBlending();
    ofSetColor(255,255,255);
    connection_icon.draw(5,5,35, 35);

    // Calibration Icon
    calibration_icon.draw(width-70, 10, 25, 25);

    // Mouse Icon
    if (mouseActive){
        mouse_icon.draw(width-110, 8, 30, 30);
    }
}
```

```
// Back Icon
back_icon.draw(width-40, 5, 35,35);
ofDisableAlphaBlending();

// Frame Rate
ofSetColor(0,0,0);
char temp[255];
sprintf(temp, "%i fps", (int)ofGetFrameRate());
verdana14.drawString(temp, 50,25);

// Similar Position
ofEnableAlphaBlending();
ofSetColor(255,255,255);

position_icon.draw(110,5,35,35);
ofDisableAlphaBlending();
tSerial.lock();

// Position
if (tSerial.rightHand.difference<1.5){

    sprintf(temp, "%i", (int)tSerial.rightHand.average_position);
    verdana14.drawString(temp, 124, 31);
    if ((int)tSerial.rightHand.average_position!=prev_pos){
        prev_pos=(int)tSerial.rightHand.average_position;
        positionTimer=0;
    }
    positionTimer++;

    // Mouse active
    if (positionTimer>50&&prev_pos==10){
        mLocation.x=ofGetMouseX();
        mLocation.y=ofGetMouseX();
        mouseActive=!mouseActive;
        positionTimer=-100;
    }
}

tSerial.unlock();

// Sampling Rate
ofSetColor(0,0,0);
sprintf(temp, "Fs: %i", samplingfreq);
verdana14.drawString(temp, 150, 25);
}

//-----
void testApp::printByte(int inbyte){
    // Prints a byte as 0s and 1s

    int byte=inbyte;
    if(byte>=128){ cout << "1"; byte-=128;} else cout << "0";
```

```
    if(byte>=64){ cout << "1"; byte-=64;} else cout << "0";
    if(byte>=32){ cout << "1"; byte-=32;} else cout << "0";
    if(byte>=16){ cout << "1"; byte-=16;} else cout << "0";
    if(byte>=8){ cout << "1"; byte-=8;} else cout << "0";
    if(byte>=4){ cout << "1"; byte-=4;} else cout << "0";
    if(byte>=2){ cout << "1"; byte-=2;} else cout << "0";
    if(byte>=1){ cout << "1"; byte-=1;} else cout << "0";
    cout << "\n";
}

//-----
void testApp::gestAnalysis(){
    // Deals with the recording and analysing functions

    tSerial.rightHand.moveStatus();

    // Record
    if (record_state!=0&&menu_sel==5){
        // Recording state

        if (tSerial.rightHand.record_available){
            tSerial.mGestureOpt.myButtons[0].onit=true;
        }
        // Once recorded, process the data

        else if(!
tSerial.rightHand.record_available&&tSerial.mGestureOpt.myButtons[0].onit){
            record_state=0;

            tSerial.rightHand.recordSet(tSerial.gestLength);

            tSerial.rightHand.calcAttr();

            tSerial.rightHand.storeGest(numclass-1);
            tSerial.gestLength=0;
            //tSerial.rightHand.print(numclass);

            // Change the recording button state
            tSerial.mGestureOpt.myButtons[0].c_inside.set(0,255,255);

            if (isOnIt(tSerial.mGestureOpt.myButtons[0], ofGetMouseX(),
ofGetMouseY())){
                tSerial.mGestureOpt.myButtons[0].b_sel=true;
            } else tSerial.mGestureOpt.myButtons[0].b_sel=false;
            tSerial.mGestureOpt.myButtons[0].onit=false;
        }
    }

    // Analysis of an incoming gesture
    if (record_state==0&&tSerial.rightHand.record_available) tSerial.input_gest=true;

    if (!tSerial.rightHand.record_available&&tSerial.input_gest){
```

```
        tSerial.input_gest=false;
        tSerial.rightHand.recordSet(tSerial.input_gestLength);
        tSerial.rightHand.calcAttr();
        tSerial.rightHand.calcGesture();
        tSerial.input_gestLength=0;
    }

}

//-----
void testApp::drawSerialConfiguration(){
    // Displays the menu of the Serial Configuration

    ofSetColor(0,0,0);
    verdana18.drawString("Serial Communication", 110,100);

    // Changes the color of the button to set the serial port
    if (serialConnection){
        selected.set(100,255,100);
    }
    else {
        selected.set(255,100,100);
    }

    ofSetColor(0,0,0);
    verdana18.drawString("Serial Port", 110,170);
    verdana18.drawString("Sampling rate", 110,200);

    // Initiates the button to set the serial port
    if (scdevice.r==0){
        button bdevice=button(width/2-150-verdana18.stringWidth(device),150 ,
                               0, 0 , 8, true, strcat(device,"~"), c_black, selected,
c_black);
        button bplus=button(width-230,180 ,
                             35, 35 , 8, c_black, c_cyan);
        button bminus=button(width-170,180 ,
                              35, 35 , 8, c_black, c_cyan);

        scdevice=bdevice;
        plus=bplus;
        minus=bminus;
        device[strlen(device)-1]='\0';
    }

    // Draw the button to set the serial port
    ofSetLineWidth(0.10f);
    scdevice.c_inside=selected;
    scdevice.x=width-150-verdana18.stringWidth(device);
    scdevice.draw();
    plus.draw();
}
```

```
        minus.draw();
        verdana18.drawString("+", width-220,202);
        verdana18.drawString("-", width-157,202);

        mSerialConf.draw();

}

//-----
void testApp::loadSerialPort(){
    std::ifstream file;
    file.open("serialport.txt");
    char c[255];
    int index=0;
    while(!file.eof()){
        file.read(c+index, 1);
        index++;
    }
    c[strlen(c)]='\0';
    device[0]='\0';
    strcpy(device, c);
    file.close();
    cout << device << endl;
}

void testApp::saveSerialPort(){
    std::ofstream file;
    file.open("serialport.txt");
    file << device;
    file.close();
    cout << device << endl;
}

//-----
void testApp::drawGames(){
    // Displays the menu of the Games
}

//-----
void testApp::drawParticles(){

    // Background
    ofSetColor(c_bgPart);
    ofFill();
    ofBeginShape();
        ofVertex(0, 50);
        ofVertex(width, 50);
        ofVertex(width,height);
        ofVertex(0,height);
    ofEndShape();

    // Change background color gradually
```

```
    if (c_bgPart.r>2) c_bgPart-=2;

    // Draw particle system
    if (c_bgPart.r<4){
        cam.begin();
        cam.setTarget(ofVec3f(0, -290, 290));

        mySystem.update( 0, -tSerial.rightHand.ffilt[0][0].y,
tSerial.rightHand.ffilt[0][0].z);
        mySystem.draw();
        ofPushMatrix();
            ofTranslate(0,0,100);
                mySystem2.update( 0, -tSerial.rightHand.ffilt[1][0].y,
tSerial.rightHand.ffilt[1][0].z);
                mySystem2.draw();
            ofPopMatrix();

        // Audio
        if(tSerial.rightHand.ffilt[0][0].x>600&&!explosion.getIsPlaying()){
            explosion.play();
            myExplosion=pSystem(0, -tSerial.rightHand.ffilt[0][0].y,
tSerial.rightHand.ffilt[0][0].z, 1000, imgExpl);
            myExplosion.setExplosion();
        }
        if (explosion.getIsPlaying()){
            myExplosion.updateExpl();
            myExplosion.draw();
        }
        cam.end();

    }

    glDisable(GL_DEPTH_TEST);
}

//-----
void testApp::iniParticles(){

    // Initialize particle system

    imgPart=new ofImage();
    imgExpl=new ofImage();
    imgPart->loadImage("particle.png");
    imgExpl->loadImage("particleY.png");

    glEnable(GL_DEPTH_TEST);

    c_bgPart.set(255,255,255);
    mySystem=pSystem(0, -tSerial.rightHand.ffilt[0][0].y, tSerial.rightHand.ffilt[0]
[0].z, 500, imgPart);
```



```
    mySystem2=pSystem(0, -tSerial.rightHand.ffilt[1][0].y, tSerial.rightHand.ffilt[1][0].z, 500, imgPart);
}

//-----
void testApp::drawGBouncingBalls(){
    // Display the Bouncing Balls game
    tSerial.lock();

    gravity.x=(tSerial.rightHand.ffilt[0][0].z/1000-
(float)tSerial.rightHand.neutral_acc*0.001);
    gravity.y=(tSerial.rightHand.ffilt[0][0].y/1000-
(float)tSerial.rightHand.neutral_acc*0.001);
    tSerial.unlock();

    // Holes
    for (int i=0; i<gameBalls_level*5+1; i++){
        if (i==gameBalls_level*5) i=100;
        // Attraction
        // The if and else could be avoided with cos and sin
        if (ofDist(location.x, location.y, myHoles[i].x, myHoles[i].y)<38){
            if (location.x>myHoles[i].x){
                gravity.x+=((myHoles[i].x -location.x)+37)/(-25);
                velocity.x*=abs((myHoles[i].x -location.x))/38;
            } else {
                gravity.x+=((myHoles[i].x -location.x)-37)/(-25);
                velocity.x*=abs((myHoles[i].x -location.x))/38;
            }

            if (location.y>myHoles[i].y){
                gravity.y+=((myHoles[i].y -location.y)+37)/(-25);
                velocity.y*=abs((myHoles[i].y -location.y))/38;
            } else {
                gravity.y+=((myHoles[i].y -location.y)-37)/(-25);
                velocity.y*=abs((myHoles[i].y -location.y))/38;
            }
        }
        if (i!=100) myHoles[i].draw();

        // Reset
        if (ofDist(location.x, location.y, myHoles[i].x, myHoles[i].y)<20&&i!=100){
            c_ball.r-=2;
            c_ball.g--;
            if (c_ball.r<2){
                location.set(30,height-30);
                velocity.set(-1.5,0);
                gravity.set(0,0);
                c_ball.set(255,127,0);
            }
        }
    }
}
```

```
// Code based on Processing examples (Bouncing Ball)

location+=velocity;
velocity+=gravity;

// Bounce off edges
if (location.x>width-20){
    velocity.x*=-0.5;
    location.x=width-21;
} else if (location.x<20){
    velocity.x*=-0.5;
    location.x=21;
}
if (location.y > height-20) {
    velocity.y *=-0.5;
    location.y = height-20;
} else if (location.y < 60){
    velocity.y*=-0.5;
    location.y= 60;
}

// Exit/Next Level
if (ofDist(location.x, location.y, myHoles[100].x, myHoles[100].y)<20){
    c_ball.g--;
    if (c_ball.g<2){
        if (gameBalls_level<20) gameBalls_level++;
        iniGBouncingBalls(gameBalls_level);
    }
}
myHoles[100].drawExit();

// Ball
ofSetColor(c_ball);
circle(location.x, location.y, 20, 8, true);
circle(location.x, location.y, 30, 8, false);

//Upper Limit
ofSetColor(c_black);
ofBeginShape();
    ofVertex(0,40);
    ofVertex(width, 40);
ofEndShape();

//Level icon
ofSetColor(0,0,0);
char temp[255];
sprintf(temp, "Level %i", gameBalls_level);
verdana14.drawString(temp, 300,25);

//Restart icon
ofEnableAlphaBlending();
ofSetColor(255,255,255);
```

```
        restart_icon.draw(380,5,30, 30);
        ofDisableAlphaBlending();
    }

//-----
void testApp::iniGBouncingBalls(int level){

    // Initialize variables

    c_ball.set(255,127,0);
    location.set(30,height-30);
    velocity.set(0,0);
    gravity.set(0,0);
    gameBalls_level=level;

    restart_icon.loadImage("restart.png");

    for (int i=0; i<level*5; i++){
        myHoles[i].x=rand()%(width-150)+75;
        myHoles[i].y=rand()%(height-80) + 60;
    }

    myHoles[100].x=width-25;
    myHoles[100].y=rand()%(height-80) + 60;
}

//-----
void testApp::drawPositions(){
    // Display the menu of the Position

    mPositions.draw();
    mPositionOpt.draw();

    // Position Indicators
    int margin=100;
    int linewidth=floor((width-margin)/(6));

    // Position (angle between fingers)
    for (int i=0; i<6; i++){
        // Circle
        ofSetLineWidth(1);
        ofSetColor(0,0,0);
        for (int a=0; a<30; a++){
            ofLine( margin/2 + linewidth/2+i*linewidth + cos((a*12)*PI/180)*30,
                    height-60 + sin((a*12)*PI/180)*30,
                    margin/2 +linewidth/2+i*linewidth + cos((a*12+13)*PI/180)*30,
                    height-60 + sin((a*12+13)*PI/180)*30 );
        }

        // Position Indicator
        ofSetLineWidth(1);
    }
}
```

```
        ofSetColor(0,0,0);
        ofLine(margin/2 +linewidth/2+i*linewidth,      height-60,
              margin/2  +linewidth/2+i*linewidth  +
cos(tSerial.rightHand.position[i])*35,
              height-
60+sin(tSerial.rightHand.position[i])*35 );

        if (mPositionSel>=0){
            ofSetLineWidth(2);
            ofSetColor(100,0,0);
            ofLine(margin/2 +linewidth/2+i*linewidth,      height-60,
                  margin/2  +linewidth/2+i*linewidth  +
cos(tSerial.rightHand.static_positions[mPositionSel][i])*35,
                  height-
60+sin(tSerial.rightHand.static_positions[mPositionSel][i])*35 );
            ofSetLineWidth(1);
        }
    }
}

//-----
void testApp::drawGesture(){

    // Display the menu to record new gestures

    mGesture.draw();

    tSerial.mGestureOpt.draw();

    // Input data
    // f1x f1y f1z f1derx f1dery f1derz
    // f2x f2y f2z f2derx f2dery f2derz
    // ...
    // a01 a12 a23 a30 a02 a13
    int margin=100;
    int linewidth=floor((width-margin)/6);

    ofSetLineWidth(0.5);

    ofSetPolyMode(OF_POLY_WINDING_ODD);
    ofSetColor(0,255,0);

    // Display data

    for (int i=0; i<4; i++){

        // Raw
        ofSetColor(125,255,125);
        ofBeginShape();
        ofNoFill();
        for (int j=0; j<linewidth-1; j++){
```

```
        ofVertex(j+margin/2, (int)tSerial.rightHand.f[i][j].x/8+i*60+400);
    }
    ofEndShape();

    ofBeginShape();
    ofNoFill();
    for (int j=0; j<linewidth-1; j++){
        ofVertex(j+linewidth+margin/2, (int)tSerial.rightHand.f[i]
[j].y/8+i*60+400);
    }
    ofEndShape();

    ofBeginShape();
    ofNoFill();
    for (int j=0; j<linewidth-1; j++){
        ofVertex(j+2*linewidth+margin/2, (int)tSerial.rightHand.f[i]
[j].z/8+i*60+400);
    }
    ofEndShape();

    // Filtered
    ofSetColor(0,255,0);
    ofBeginShape();
    ofNoFill();
    for (int j=0; j<linewidth-1; j++){
        ofVertex(j+margin/2, (int)tSerial.rightHand.ffilt[i][j].x/8+i*60+400);
    }
    ofEndShape();

    ofBeginShape();
    ofNoFill();
    for (int j=0; j<linewidth-1; j++){
        ofVertex(j+linewidth+margin/2, (int)tSerial.rightHand.ffilt[i]
[j].y/8+i*60+400);
    }
    ofEndShape();

    ofBeginShape();
    ofNoFill();
    for (int j=0; j<linewidth-1; j++){
        ofVertex(j+2*linewidth+margin/2, (int)tSerial.rightHand.ffilt[i]
[j].z/8+i*60+400);
    }
    ofEndShape();

    // Derivative
    ofSetColor(255,0,0);
    ofBeginShape();
    ofNoFill();
    for (int j=0; j<linewidth-1; j++){
```

```
                ofVertex(j+3*linewidth+margin/2,(int)tSerial.rightHand.ffilter[i]
[j].x/8+i*60+40+400);
            }
            ofEndShape();
            ofBeginShape();
            ofNoFill();
            for (int j=0; j<linewidth-1; j++){
                ofVertex(j+4*linewidth+margin/2,(int)tSerial.rightHand.ffilter[i]
[j].y/8+i*60+40+400);
            }
            ofEndShape();
            ofBeginShape();
            ofNoFill();
            for (int j=0; j<linewidth-1; j++){
                ofVertex(j+5*linewidth+margin/2,(int)tSerial.rightHand.ffilter[i]
[j].z/8+i*60+40+400);
            }
            ofEndShape();
        }
    }

}

//-----
void testApp::drawMusic(){
}

//-----
void testApp::drawMidi(){

    // Display MIDI menu

    ofColor(0,0,0);
    verdana18.drawString("MIDI", (width-50)/4-200, 100);
    ofLine((width-50)/4-200, 105, 4*(width-50)/4-30, 105);
    verdana18.drawString("NoteOn", (width-50)/4-200, 150);
    verdana18.drawString("Ch", 2*(width-50)/4-220, 150);
    verdana18.drawString("Ctrl", 2*(width-50)/4-150, 150);
    verdana18.drawString("Ch", 3*(width-50)/4-220, 150);
    verdana18.drawString("Ctrl", 3*(width-50)/4-150, 150);
    verdana18.drawString("Pitch Bend", 4*(width-50)/4-200, 150);

    // Display ControlChange buttons
    for (int i=0; i<2; i++){
        // Initialize buttons
        if (channelNum[i].r==0){
            button bdevice=button(0,150 ,
                0, 0 , 8, true, "1~", c_black, c_white, c_black);
            channelNum[i]=bdevice;
            channelCtrl[i]=bdevice;
            channelCtrl[i].text[0]=49+i;
        }
    }
}
```

```
        ofSetLineWidth(0.10f);

        // Display buttons
        if (channelNum[i].b_sel) channelNum[i].c_inside=c_cyan;
        else channelNum[i].c_inside=c_white;
        if (channelCtrl[i].b_sel) channelCtrl[i].c_inside=c_cyan;
        else channelCtrl[i].c_inside=c_white;

        channelCtrl[i].x=(2+i)*(width-50)/4-220+verdana18.stringWidth("Ch: ");
        channelNum[i].x=(2+i)*(width-50)/4-150+verdana18.stringWidth("Ctrl: ");
        channelCtrl[i].draw();
        channelNum[i].draw();
    }

    // Display input data selection buttons
    for (int i=0; i<8; i++){
        if (!midi_onoff[i].onit) midi_onoff[i].c_inside.set(255,255,255);
        midi_onoff[i].draw();
        midi_fingers[i*2].draw();
        midi_fingers[i*2+1].draw();
        if (i<4){
            midi_axis[i*3].draw();
            midi_axis[i*3+1].draw();
            midi_axis[i*3+2].draw();
        }
    }
}

}

//-----
void testApp::sendMidi(){

    // Send Midi data
    int note, ctrl, pbend;
    int max, min;
    tSerial.lock();
    max=tSerial.rightHand.neutral_acc-150;
    min=tSerial.rightHand.neutral_acc+150;
    tSerial.unlock();

    /*if (timer2!=ofGetSeconds()){
        cout << "MidiOnOff" << mMidiOnOffSel[0]%2 << ", " << mMidiOnOffSel[1]%2 << ",
        "<< mMidiOnOffSel[2]%2 << ", " << mMidiOnOffSel[3]%2 << endl;

        cout << "MidiFingerSel" << mMidiFingerSel[0]%4 << ", " << mMidiFingerSel[1]%4
        << ", " << mMidiFingerSel[2]%4 << ", " << mMidiFingerSel[3]%4 << endl;
        cout << "MidiAxisSel" << mMidiAxisSel[0]%3 << ", " << mMidiAxisSel[1]%3 << ",
        "<< mMidiAxisSel[2]%3 << ", " << mMidiAxisSel[3]%3 << endl;
        timer2=ofGetSeconds();
    }*/
}
```

```
// NoteOn events
if (mMidiOnOffSel[0]%2==0){
    int finger=mMidiFingerSel[0]%4;
    switch (mMidiAxisSel[0]%3){
        case 0:
            note = ofMap(tSerial.rightHand.ffilt[finger][0].z, min, max, 0, 127);
            break;
        case 1:
            note = ofMap(tSerial.rightHand.ffilt[finger][0].y, min, max, 0, 127);
            break;
        case 2:
            note = ofMap(tSerial.rightHand.ffilt[finger][0].x, min, max, 0, 127);
            break;
    }
    if (note>127) note=126;
    if (note<0) note=1;
    midiOut.sendNoteOn(1,note,100);
}

// Control Change events
for (int i=1; i<3; i++){
    if (mMidiOnOffSel[i]%2==0){
        int finger=mMidiFingerSel[i]%4;
        switch (mMidiAxisSel[i]%3){
            case 0:
                ctl = ofMap(tSerial.rightHand.ffilt[finger][0].z, min, max, 0,
127);
                break;
            case 1:
                ctl = ofMap(tSerial.rightHand.ffilt[finger][0].y, min, max, 0,
127);
                break;
            case 2:
                ctl = ofMap(tSerial.rightHand.ffilt[finger][0].x, min, max, 0,
127);
                break;
        }
        if (ctl>127) ctl=126;
        if (ctl<0) ctl=1;
        midiOut.sendControlChange(mChannelCtrl[i-1], mChannelNum[i-1], ctl);
    }
}

// Pitch Bend events
if (mMidiOnOffSel[3]%2==0){
    int finger=mMidiFingerSel[3]%4;
    switch (mMidiAxisSel[3]%3){
        case 0:
            pbend = ofMap(tSerial.rightHand.ffilt[finger][0].z, min, max, 0, 127);
            break;
        case 1:
```



```
        pbend = ofMap(tSerial.rightHand.ffilt[finger][0].y, min, max, 0, 127);
        break;
    case 2:
        pbend = ofMap(tSerial.rightHand.ffilt[finger][0].x, min, max, 0, 127);
        break;
    }
    if (pbend>127) pbend=126;
    if (pbend<0) pbend=1;
    midiOut.sendPitchBend(1, pbend);
}

}

//-----
void testApp::keyPressed(int key){

    if (key == OF_KEY_F1){
        mouseActive=!mouseActive;
    }

    // Change the name of the serial port device
    if (scdevice.b_sel){

        if(key == OF_KEY_DEL || key == OF_KEY_BACKSPACE){
            device[strlen(device)-1]='\0';
        }
        else if(key == OF_KEY_RETURN ){
            scdevice.b_sel=false;
            connectSerial();
        }

        }else{
            device[strlen(device)]= (char)key;
        }

        for (int i=0; i<=strlen(device); i++){
            scdevice.text[i]=device[i];
        }

        scdevice.w=verdana18.stringWidth(scdevice.text)+16;

    }

    for (int i=0; i<2; i++){

        // Change the name of the Control Channel
        if (channelCtrl[i].b_sel) {

            if(key == OF_KEY_DEL || key == OF_KEY_BACKSPACE){
                channelCtrl[i].text[strlen(channelCtrl[i].text)-1]='\0';
            }
            else if(key == OF_KEY_RETURN ){
                channelCtrl[i].b_sel=false;
                switch (strlen(channelCtrl[i].text)){
```

```
        case 0:
            mChannelCtrl[i]=1;
            channelCtrl[i].text[0]='1';
            channelCtrl[i].text[1]='\0';
            break;
        case 1:
            mChannelCtrl[i]=channelCtrl[i].text[0]-'0';
            break;
        case 2:
            mChannelCtrl[i]=(channelCtrl[i].text[1]-'0')
+10*(channelCtrl[i].text[0]-'0');
            if (mChannelCtrl[i]>16){
                mChannelCtrl[i]=16;
                channelCtrl[i].text[0]='1';
                channelCtrl[i].text[1]='6';
                channelCtrl[i].text[2]='\0';
            }
            break;
        default:
            mChannelCtrl[i]=16;
            channelCtrl[i].text[0]='1';
            channelCtrl[i].text[1]='6';
            channelCtrl[i].text[2]='\0';
            break;
    }
} else{
    if (key>47&&key<58){
        channelCtrl[i].text[strlen(channelCtrl[i].text)]=(char) key;
    }
}

channelCtrl[i].w=verdana18.stringWidth(channelCtrl[i].text)+16;
}

// Change the number in the Control Value
if (channelNum[i].b_sel) {

    if(key == OF_KEY_DEL || key == OF_KEY_BACKSPACE){
        if (strlen(channelNum[i].text)>0){
            channelNum[i].text[strlen(channelNum[i].text)-1]='\0';
        }
    }
    else if(key == OF_KEY_RETURN ){
        channelNum[i].b_sel=false;
        switch (strlen(channelNum[i].text)){
            case 0:
                mChannelNum[i]=1;
                channelNum[i].text[0]='1';
                channelNum[i].text[1]='\0';
                break;
            case 1:
                mChannelNum[i]=channelNum[i].text[0]-'0';
```

```
                break;
            case 2:
                mChannelNum[i]=(channelNum[i].text[1]-'0')
+10*(channelNum[i].text[0]-'0');
                break;
            case 3:
                mChannelNum[i]=(channelNum[i].text[2]-'0')
+10*(channelNum[i].text[1]-'0')+100*(channelNum[i].text[0]-'0');
                if (mChannelNum[i]>127){
                    mChannelNum[i]=127;
                    channelNum[i].text[0]='1';
                    channelNum[i].text[1]='2';
                    channelNum[i].text[2]='7';
                    channelNum[i].text[3]='\0';
                }
                break;
            default:
                mChannelNum[i]=127;
                channelNum[i].text[0]='1';
                channelNum[i].text[1]='2';
                channelNum[i].text[2]='7';
                channelNum[i].text[3]='\0';
                break;
        }
    } else{
        if (key>47&&key<58){
            channelNum[i].text[strlen(channelNum[i].text)]=(char) key;
        }
    }

    channelNum[i].w=verdana18.stringWidth(channelNum[i].text)+16;
}
}

//-----
void testApp::keyReleased(int key){

}

//-----
void testApp::mouseMoved(int x, int y ){

    int xx, yy, ww, hh;

    switch (menu_sel){
        case 0:
            // Check if the mouse is over a button
            for (int i=0; i<myMenu.numButtons; i++){
                if (isOnIt(myMenu.myButtons[i], x, y)){
                    myMenu.myButtons[i].b_sel=true;
                } else myMenu.myButtons[i].b_sel=false;
            }
        }
    }
}
```

```
    }
    break;

case 1:
    // Check if the mouse is over a button
    if (isOnIt(scdevice, x, y)){
        scdevice.onit=true;
    } else scdevice.onit=false;
    if (isOnIt(plus,x,y)){
        plus.b_sel=true;
    } else plus.b_sel=false;
    if (isOnIt(minus,x,y)){
        minus.b_sel=true;
    } else minus.b_sel=false;

    for (int i=0; i<mSerialConf.numButtons; i++){
        if (isOnIt(mSerialConf.myButtons[i], x, y)){
            mSerialConf.myButtons[i].b_sel=true;
        } else mSerialConf.myButtons[i].b_sel=false;
    }
    break;

case 4:
    // Check if the mouse is over a button
    for (int i=0; i<mPositions.numButtons; i++){

        if (isOnIt(mPositions.myButtons[i], x, y)){
            mPositions.myButtons[i].b_sel=true;
        } else mPositions.myButtons[i].b_sel=false;

        if (i<4){
            if (isOnIt(mPositionOpt.myButtons[i], x, y)){
                mPositionOpt.myButtons[i].b_sel=true;
            } else mPositionOpt.myButtons[i].b_sel=false;
        }
    }
    break;

case 5:
    // Check if the mouse is over a button
    for (int i=0; i<mGesture.numButtons; i++){

        if (isOnIt(mGesture.myButtons[i], x, y)){
            mGesture.myButtons[i].b_sel=true;
        } else mGesture.myButtons[i].b_sel=false;

        if (i<4){
            if (isOnIt(tSerial.mGestureOpt.myButtons[i], x, y)){
                tSerial.mGestureOpt.myButtons[i].b_sel=true;
            } else tSerial.mGestureOpt.myButtons[i].b_sel=false;
            if (record_state==1) tSerial.mGestureOpt.myButtons[0].b_sel=true;
        }
    }
}
```

```
    }
    break;

case 6:
    // Check if the mouse is over a button
    for (int i=0; i<16; i++){

        if (isOnIt(midi_fingers[i], x, y)){
            midi_fingers[i].b_sel=true;
            midi_fingers[i].c_inside.set(0,255,255);
        } else {
            midi_fingers[i].b_sel=false;
            if (!midi_fingers[i].onit)
midi_fingers[i].c_inside.set(255,255,255);
        }

        if (i<12){
            if (isOnIt(midi_axis[i], x, y)){
                midi_axis[i].b_sel=true;
                midi_axis[i].c_inside.set(0,255,255);
            } else {
                midi_axis[i].b_sel=false;
                if (!midi_axis[i].onit)
midi_axis[i].c_inside.set(255,255,255);
            }
        }
    }
    break;
}

//-----
bool testApp::isOnIt(button inb, int x, int y){
    // Return true if the point x,y is over the button. False otherwise
    if(x<inb.x+inb.w&&x>inb.x&&y<inb.y+inb.h&&y>inb.y){
        return true;
    } else {
        return false;
    }
}

//-----
void testApp::mouseDragged(int x, int y, int button){

}

//-----
void testApp::mousePressed(int x, int y, int button){

    switch (menu_sel){
        case 0:
            // Set the menu clicked as selected

```

```
    if (myMenu.myButtons[0].b_sel&&button==0){
        menu_sel=1;
        menu_prev_sel=0;
    } else if(myMenu.myButtons[1].b_sel&&button==0){
        iniParticles();
        menu_sel=2;
        menu_prev_sel=0;
    } else if(myMenu.myButtons[2].b_sel&&button==0){
        gameBalls_level=1;
        iniGBouncingBalls(gameBalls_level);
        menu_sel=3;
        menu_prev_sel=0;
    } else if(myMenu.myButtons[3].b_sel&&button==0){
        menu_sel=4;
        menu_prev_sel=0;
    } else if(myMenu.myButtons[4].b_sel&&button==0){
        menu_sel=5;
        menu_prev_sel=0;
    } else if(myMenu.myButtons[5].b_sel&&button==0){
        menu_sel=6;
        menu_prev_sel=0;
    }
    break;

case 1:
    // Set the text button editable
    if (scdevice.onit){ scdevice.c_inside.set(0,255,255);
scdevice.b_sel=true;}
    else scdevice.b_sel=false;

    if (plus.b_sel&&button==0){
        tSerial.stop();
        ofSleepMillis(10);
        serial.writeByte('2');
        tSerial.start();
    }
    if(minus.b_sel&&button==0){
        tSerial.stop();
        ofSleepMillis(10);
        serial.writeByte('1');
        tSerial.start();
    }

    // Option button actions
    if (mSerialConf.myButtons[0].b_sel&&button==0){
        menu_sel=0;
        menu_prev_sel=1;
    } else if (mSerialConf.myButtons[1].b_sel&&button==0){
        saveSerialPort();
    } else if (mSerialConf.myButtons[2].b_sel&&button==0){
```

```
        loadSerialPort();
    }
    break;

case 3:

    // Restart Icon: restart_icon.draw(380,5,35, 35);
    if (x<(380+25)&&x>380&&y<35&&y>10&&button==0){
        cout << "\n\n  CLICKED!\n\n";
        iniGBouncingBalls(gameBalls_level);
    }
    break;

case 4:
    // Select the position number
    for (int i=0; i<mPositions.numButtons; i++){
        mPositions.myButtons[i].onit=false;
        if (mPositions.myButtons[i].b_sel&&button==0){
            mPositionSel=i;
        }

        // Option button actions
        if (i<4) {
            if (mPositionOpt.myButtons[i].b_sel&&button==0){
                switch (i){
                    case 0:
                        menu_sel=0;
                        menu_prev_sel=4;
                        break;
                    case 1:
                        tSerial.rightHand.storePosition(mPositionSel,true);
                        break;
                    case 2:
                        tSerial.rightHand.loadPositionPresets();
                        break;
                    case 3:
                        tSerial.rightHand.printPosition(mPositionSel);
                        break;
                }
            }
        }
    }
    mPositions.myButtons[mPositionSel].onit=true;
    break;

case 5:
    // Select the gesture number
    for (int i=0; i<mGesture.numButtons; i++){
        mGesture.myButtons[i].onit=false;
        if (mGesture.myButtons[i].b_sel&&button==0){
            mGestureSel=i;
            // Changes the type of movement we are recording
            numclass=i+1;
        }
    }
}
```

```
        /*classname[4]=(floor(float(numclass)/10)) + '0';
        classname[5] = numclass % 10 +'0';
        classname[6] = '\\0';*/
    }
    // Option button actions
    if (i<4) {
        if (tSerial.mGestureOpt.myButtons[i].b_sel&&button==0){
            switch (i){
                case 0:
                    tSerial.mGestureOpt.myButtons[i].c_inside.set(255,50,5
0);

                    if (mGestureSel!=-1) record_state=1;
                    break;
                case 1:
                    record_state=0;
                    menu_sel=0;
                    menu_prev_sel=5;
                    break;
                case 2:
                    record_state=0;
                    //tSerial.rightHand.loadPositionPresets();
                    break;
                case 3:
                    record_state=0;
                    //tSerial.rightHand.printPosition(mGestureSel);
                    break;
            }
        }
    }
    mGesture.myButtons[mGestureSel].onit=true;
    break;

case 6:
    // Select the finger
    for (int i=0; i<16; i++){
        midi_fingers[i].onit=false;
        if (midi_fingers[i].b_sel&&button==0){
            mMidiFingerSel[(int)floor(i/4)]=i;
        }
    }
    // Select the axis
    if (i<12){
        midi_axis[i].onit=false;
        if (midi_axis[i].b_sel&&button==0){
            mMidiAxisSel[(int)floor(i/3)]=i;
        }
    }
    // Turn On/Off MIDI messages
    if (i<8){
        if (isOnIt(midi_onoff[i], x, y)&&button==0){
            midi_onoff[i].onit=true;
            mMidiOnOffSel[(int)floor(i/2)]=i;
        }
    }
}
```



```
    }
    // Set text button editable
    if (i<2){
        if (isOnIt(channelCtrl[i], x, y)&&button==0){
            channelCtrl[i].b_sel=true;
        }
        if (isOnIt(channelNum[i], x, y)&&button==0){
            channelNum[i].b_sel=true;
        }
    }
}

for (int i=0; i<4; i++){
    // Set the selected buttons
    if (mMidiFingerSel[i]>-1) midi_fingers[mMidiFingerSel[i]].onit=true;
    if (mMidiAxisSel[i]>-1) midi_axis[mMidiAxisSel[i]].onit=true;
    if (mMidiOnOffSel[i]>-1) midi_onoff[mMidiOnOffSel[i]].onit=true;

    if(mMidiOnOffSel[i]%2==0){
        midi_onoff[mMidiOnOffSel[i]].c_inside.set(100,255,100);
        midi_onoff[mMidiOnOffSel[i]+1].onit=false;
    }
    else{
        midi_onoff[mMidiOnOffSel[i]].c_inside.set(255,100,100);
        midi_onoff[mMidiOnOffSel[i]-1].onit=false;
    }
}
break;

}

// Connection Icon: connection_icon.draw(5,5,35, 35);
if (x<35&&x>10&&y<35&&y>10&&button==0){
    menu_prev_sel=menu_sel;
    menu_sel=1;
}
// Back Icon: back_icon.draw(width-40, 5, 35,35);
else if (x<width-5&&x>width-35&&y<35&&y>10&&button==0){
    int temp_msel=menu_sel;
    record_state=0;
    menu_sel=menu_prev_sel;
    menu_prev_sel=temp_msel;
}

// Calibration Icon: calibration_icon.draw(width-60, 5, 35,35);
else if (x<width-40&&x>width-70&&y<35&&y>10&&button==0){
    tSerial.lock();
    tSerial.rightHand.calibrate();
    tSerial.unlock();
}
}
```

```
}

//-----
void ofApp::mouseReleased(int x, int y, int button){

}

//-----
void ofApp::windowResized(int w, int h){

}

//-----
void ofApp::gotMessage(ofMessage msg){

}

//-----
void ofApp::dragEvent(ofDragInfo dragInfo){

}

//-----
void ofApp::iniGraphics(){
    // Initialize Graphics

    // Graphics
    ofSetWindowTitle("Glove App");
    ofSetWindowPosition(0, 0);
    ofSetFrameRate(60);
    ofEnableSmoothing();

    // Menu
    myMenu = menu (6, 0, 0, 0, 150, 150, 80, 80, true);

    // Menu Serial conf
    mSerialConf = menu (3, 1, ofGetWidth()/2-120, ofGetHeight()-200,
        60, 60, 20, 20, true); //ofGetWidth()/2-120-
>120=(60+20)*3/2

    // Menu Positions
    mPositions = menu (10, 4, 30, 30, 100, 100, 30, 30, true);
    mPositionOpt = menu (4, 41, ofGetWidth()/2-160, ofGetHeight()-200,
        60, 60, 20, 20, true);

    // Menu Gesture
    mGesture = menu (10, 4, 30, 30, 100, 100, 30, 30, true);
    tSerial.mGestureOpt = menu (5, 51, width/2-200, height-100,
        60, 60, 20, 20, true);

    c_black.set(0,0,0);
    c_cyan.set(0,255,255);
}
```

```
c_white.set(255,255,255);

connection_icon.loadImage("connection.png");
back_icon.loadImage("back.png");
position_icon.loadImage("pos.png");
calibration_icon.loadImage("calibrate.png");
mouse_icon.loadImage("mouse.png");

ofTrueTypeFont::setGlobalDpi(72);

// Initalize font
verdana18.loadFont("verdana.ttf", 18, true, true);
verdana18.setLineHeight(25.0f);
verdana18.setLetterSpacing(1.000);

verdana14.loadFont("verdana.ttf", 14, true, true);
verdana14.setLineHeight(18.0f);
verdana14.setLetterSpacing(1.037);

// Music Midi
ofColor stroke(0,0,0);
ofColor c_on(100, 255, 100);
ofColor c_off(255,100,100);
ofColor nonselected(255,255,255); // Watch out, background might be different.
ofColor textcolor(0,0,0);
mChannelCtrl[0]=1;
mChannelCtrl[1]=2;
mChannelNum[0]=1;
mChannelNum[1]=1;

// Initalize buttons
for (int i=0; i<4; i++){
    button on=button(i*(width-50)/4 + 50, 200, 0, 0, 12,
                    true, "ON-", stroke, nonselected, textcolor);
    button off=button(i*(width-50)/4 + 90, 200, 0, 0, 12,
                    true, "OFF-", stroke, c_off, textcolor);

    midi_onoff[i*2]=on;
    midi_onoff[i*2+1]=off;

    button bf0=button(i*(width-50)/4+60, 250, 0, 0, 12,
                    true, "f0-", stroke, nonselected, textcolor);
    button bf1=button(i*(width-50)/4+90, 250, 0, 0, 12,
                    true, "f1-", stroke, nonselected, textcolor);
    button bf2=button(i*(width-50)/4+60, 280, 0, 0, 12,
                    true, "f2-", stroke, nonselected, textcolor);
    button bf3=button(i*(width-50)/4+90, 280, 0, 0, 12,
                    true, "f3-", stroke, nonselected, textcolor);

    midi_fingers[i*4]=bf0;
    midi_fingers[i*4+1]=bf1;
}
```

```
        midi_fingers[i*4+2]=bf2;
        midi_fingers[i*4+3]=bf3;

        button bx=button(i*(width-50)/4+50, 330, 0, 0, 12,
            true, "X-", stroke, nonselected, textcolor);
        button by=button(i*(width-50)/4+80, 330, 0, 0, 12,
            true, "Y-", stroke, nonselected, textcolor);
        button bz=button(i*(width-50)/4+112, 330, 0, 0, 12,
            true, "Z-", stroke, nonselected, textcolor);

        midi_axis[i*3]=bx;
        midi_axis[i*3+1]=by;
        midi_axis[i*3+2]=bz;
    }
}

//-----
void testApp::iniCSV(){
    // Creates the .csv file. This is for weka testing only.

    std::ofstream file;
    char filename[]="moves";
    char extension[]=".csv";
    strcat(filename,extension);
    file.open(filename);
    char axis[]={'a','y','z'};
    for (int i=1; i<5; i++){
        for (int j=0; j<3; j++){
            file << "f" << i ;
            file << "energy_" <<axis[j] << ", ";
        }
        for (int j=0; j<3; j++){
            file << "f" << i ;
            file << "lat_" <<axis[j]<< ", ";
        }
        for (int j=0; j<3; j++){
            file << "f" << i ;
            file << "tc_" <<axis[j]<< ", ";
        }
        for (int j=0; j<3; j++){
            file << "f" << i ;
            file << "zcr_" <<axis[j]<< ", ";
        }
    }

    file << "class" <<std::endl;
    file.close();
}

//-----
void testApp::exit(){
```

```
    // Close connections and clear memory
    serial.close();
    midiOut.closePort();
    tSerial.stop();

    delete imgPart;
}

//-----
void testApp::circle(int x, int y, float rad, float res, bool opt_fill){
    // Improved circle drawing

    if (opt_fill){ circle(x,y,rad+0.1,res,false); ofFill(); }
    else ofNoFill();

    ofBeginShape();
    for (int i=0; i<(int)(360.0/res+1); i++){

        ofVertex( x + cos((i*res)*PI/180)*rad,
                  y + sin((i*res)*PI/180)*rad);
        ofVertex(x + cos((i*res+res/2)*PI/180)*rad,
                  y + sin((i*res+res/2)*PI/180)*rad );

    }
    ofEndShape();
}

//-----
void testApp::clickMouse(){

    Display *display = XOpenDisplay(0);

    Window root = DefaultRootWindow(display);
    XWarpPointer(display, None, root, 0,0,0,0, mLocation.x, mLocation.y);
    XFlush(display);

    Display *display2= XOpenDisplay(NULL);

    XEvent event;

    if (display2 == NULL){
        cout << "ERRR00000R\n";
    }

    memset (&event, 0x00, sizeof(event));

    event.type = ButtonPress;
    event.xbutton.button=Button1; //? is an int
```

```
    event.xbutton.same_screen = True;

    XQueryPointer (display2, RootWindow(display2, DefaultScreen(display2)),
&event.xbutton.root,
&event.xbutton.window, &event.xbutton.x_root,
&event.xbutton.y_root,
&event.xbutton.x, &event.xbutton.y, &event.xbutton.state);

    event.xbutton.subwindow = event.xbutton.window;

    while (event.xbutton.subwindow){
        event.xbutton.window = event.xbutton.subwindow;

        XQueryPointer (display2, event.xbutton.window, &event.xbutton.root,
&event.xbutton.subwindow,
&event.xbutton.x_root, &event.xbutton.y_root, &event.xbutton.x,
&event.xbutton.y, &event.xbutton.state);
    }

    if (XSendEvent(display2, PointerWindow, True, 0xffff, &event)==0) cout << "Error
2\n\n";

    XFlush (display2);

    ofSleepMillis(100);

    event.type = ButtonRelease;
    event.xbutton.state = 0x100;

    if (XSendEvent(display2, PointerWindow, True, 0xffff, &event)==0) cout << "Error
3\n\n";

    XFlush(display2);

    XCloseDisplay(display2);

    XFlush(display);

    XCloseDisplay(display);

}

//-----
void testApp::moveMouse(){

    Display *display = XOpenDisplay(0);
```

```
Window root = DefaultRootWindow(display);

tSerial.lock();
                                mGravity.x=(tSerial.rightHand.ffilt[0][0].z/1000-
(float)tSerial.rightHand.neutral_acc*0.001);
                                mGravity.y=(tSerial.rightHand.ffilt[0][0].y/1000-
(float)tSerial.rightHand.neutral_acc*0.001);
tSerial.unlock();

if (mGravity.x>0) mSign.x=1;
else mSign.x=-1;
if (mGravity.y>0) mSign.y=1;
else mSign.y=-1;

mLocation+=mSign*(mGravity*100)*(mGravity*100)/2;

//mVelocity+=mGravity;

// Bounce off edges
if (mLocation.x>ofGetWidth()){
    velocity.x=0;
    mLocation.x=ofGetWidth();
} else if (mLocation.x<5){
    velocity.x=0;
    mLocation.x=5;
}
if (mLocation.y > ofGetScreenHeight()) {
    velocity.y =0;
    mLocation.y = ofGetScreenHeight();
} else if (mLocation.y < 5){
    velocity.y=0;
    mLocation.y= 5;
}

XWarpPointer(display, None, root, 0,0,0,0, mLocation.x, mLocation.y);
XFlush(display);
XCloseDisplay(display);
}
```

test.h

```
#pragma once

#include "ofMain.h"
#include "ofxMidi.h"
#include "ofEvents.h"
#include "hand.h"
#include "menu.h"
#include "holes.h"
#include "pSystem.h"
#include "threadSerial.h"
#include <fstream>
#include <string>

#define ERRORTHRES 600

class testApp : public ofApp{

public:

    void setup();
    void update();
    void draw();
    void exit();

    void gestAnalysis();

    void keyPressed (int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    // Serial connection
    ofSerial serial;
    bool serialConnection;

    threadSerial tSerial;
    void newSerialData(std::vector < float > &serialData);

    //hand rightHand;

private:
    int count;

    void iniCSV();
```



```
void iniGraphics();

void clickMouse();
void moveMouse();

void connectSerial();

// Midi
ofxMidiOut midiOut;
void sendMidi();

// Graphic functions
void drawSerialConfiguration();

void drawGames();
void drawGBouncingBalls();
void circle(int x, int y, float rad, float res, bool opt_fill);

void drawParticles();

void drawGesture();

void drawMusic();
void drawMidi();

void drawPositions();

bool isOnIt(button inb, int x, int y);

void printByte(int inbyte);

// Menu
menu myMenu;

// Status icons
ofImage connection_icon;
ofImage back_icon;
ofImage position_icon;
ofImage calibration_icon;
ofImage mouse_icon;

// Global variables
int width, height;
int menu_sel, menu_prev_sel;
int samplingfreq;
int samplingfreq_ind;
ofColor c_black, c_cyan, c_white;

// Audio
ofSoundPlayer explosion;
```

```
// Font
    ofTrueTypeFont    verdana18;
    ofTrueTypeFont    verdana14;

// Serial Configuration
button scdevice, plus, minus;
menu mSerialConf;
ofColor selected;
void loadSerialPort();
void saveSerialPort();

// Serial device
    char device[200];

    // GBouncingBalls
    void iniGBouncingBalls(int level);
    ofPoint location;
ofPoint velocity;
ofPoint gravity;
int gameBalls_level;
ofColor c_ball;
Holes myHoles[101];
ofImage restart_icon;

// Particles
void iniParticles();
ofColor c_bgPart;
ofImage* imgPart;
ofImage* imgExpl;
pSystem mySystem;
pSystem mySystem2;
pSystem myExplosion;
ofEasyCam cam;

// MPositions
menu mPositions;
menu mPositionOpt;
int mPositionSel;
int positionTimer;
int prev_pos;

// MGesture
menu mGesture;
menu mGestureOpt;
int mGestureSel;

// MMidi
button midi_onoff[8];
button midi_fingers[16];
button midi_axis[12];
button channelNum[2];
button channelCtrl[2];
```

```
int mMidiFingerSel[4];
int mMidiAxisSel[4];
int mMidiOnOffSel[4];
int mChannelNum[2];
int mChannelCtrl[2];

// Retry connection
int attempts;
int prev_sec;

// Record variables
int record_state;
//int gestLength;
char classname[7];
int numclass;
//bool input_gest;
//int input_gestLength;

// Mouse
ofPoint mLocation, mVelocity, mGravity, mSign;
bool mouseActive;
bool inClick;

int timer2;

};
```

threadSerial.h

```
#ifndef _THREADED_OBJECT
#define _THREADED_OBJECT

#include "ofMain.h"
#include "hand.h"
#include "menu.h"

// this is not a very exciting example yet
// but ofThread provides the basis for ofNetwork and other
// operations that require threading.
//
// please be careful - threading problems are notoriously hard
// to debug and working with threads can be quite difficult

class threadSerial : public ofThread{

public:

    // threaded fucntions that share data need to use lock (mutex)
    // and unlock in order to write to that data
    // otherwise it's possible to get crashes.
    //
    // also no opengl specific stuff will work in a thread...
    // threads can't create textures, or draw stuff on the screen
    // since opengl is single thread safe

    //-----
    threadSerial(){
    }

    void start(){
    startThread(true,false);
    }

    void start(ofSerial & pserial){
        serial=&pserial;
        if (serial->available()<25) { serial->writeByte('0');}
    startThread(true, false); // blocking, verbose
    }

    void stop(){
        stopThread();
    }
}
```

```
//-----  
void threadedFunction(){  
  
    while( isThreadRunning() != 0 ){  
  
        if( lock() ){  
  
            // . Using 5 bytes for a finger (3 streams):  
            //  xx (00) . yyy . zzz → x indicates type of byte, y indicates  
channel, z used in third byte.  
            //  aa (01) . bbb . ccc → a indicates type of byte, zzz, bbb, ccc  
variable to give more range to the following bytes.  
            //  i (1) . jjjjjjj → i indicates type of byte, j represents  
data.  
            //  i (1) . jjjjjjj → i indicates type of byte, j represents  
data.  
            //  i (1) . jjjjjjj → i indicates type of byte, j represents  
data.  
  
            // Alternative: read all the 5 bytes together and then process  
them  
            if (serial->available(<25) { serial->writeByte('0'); }  
  
            if (serial->available()){  
  
                // First Byte 00 . yyy . zzz  
                byte=serial->readByte();  
  
                err=-1;  
                if (byte>=64) { //err 0  
  
                    // To assure that we start with the right byte  
                    while (byte>=64){  
                        byte=serial->readByte();  
  
                        err=-1;  
                    }  
                }  
            }  
  
            // Byte conversion because RS-232 serial communication  
standard  
  
            if (byte==7) byte=3;  
            if (byte==22) byte=17;  
            if (byte==23) byte=19;  
            if (byte==30) byte=26;  
            if (byte==31) byte=28;  
  
            channel=0;  
            if (byte>= 32){ channel=4; byte-=32;}  
            if (byte>= 16){ channel+=2; byte-=16;}  
            if (byte>=8){ channel+=1; byte-=8;}  
        }  
    }  
}
```

```
fix[0]=byte;

// Second Byte 01 . bbb . ccc
byte=serial->readByte();
//printByte(byte);

if (byte< 64){ // err 1
}
else { byte-=64;}
fix[1]=0;
if (byte>= 32){ fix[1]=4; byte-=32;}
if (byte>= 16){ fix[1]+=2; byte-=16;}
if (byte>=8){ fix[1]+=1; byte-=8; }

fix[2]=byte;

// Rest of the bytes 1 . jjjjjjj

for (int i=0; i<3; i++){
    byte=serial->readByte();
    //printByte(byte);
    if (byte<128){ // err i+2
    }
    else byte-=128;
    sensfix[i]=byte;
}

switch (err){
    case -1:
        if (channel < 5){
            samplingfreq_ind++;
            rightHand.f[channel][0].x=(float) (fix[0]*128
+ sensfix[0]);
            rightHand.f[channel][0].y=(float) (fix[1]*128
+ sensfix[1]);
            rightHand.f[channel][0].z=(float) (fix[2]*128
+ sensfix[2]);

            //cout << (fix[0]*128 + sensfix[0]) << ", "
<< (fix[1]*128 + sensfix[1])<< ", " <<(fix[2]*128 + sensfix[2]) <<"\n";
        } else if (channel > 4){
            //leftHand
        }
        break;
}

rightHand.update();

samplingfreq_ind++;

// gestLength
```

```
        if (gestLength>300||!input_gest){
            rightHand.record_available=false;
        }
        if (mGestureOpt.myButtons[0].onit==true) gestLength++;
        if (input_gest) input_gestLength++;
    }

    if (prev_sec!=ofGetSeconds()){
        prev_ellapsed=ofGetElapsedTimeMillis();
    }
    prev_sec=ofGetSeconds();

        unlock();

    }
    if (serial->available()<25) { serial->writeByte('0');}

}

}

void printByte(int inbyte){
// Prints a byte as 0s and 1s

int byte=inbyte;
if(byte>=128){ cout << "1"; byte-=128;} else cout << "0";
if(byte>=64){ cout << "1"; byte-=64;} else cout << "0";
if(byte>=32){ cout << "1"; byte-=32;} else cout << "0";
if(byte>=16){ cout << "1"; byte-=16;} else cout << "0";
if(byte>=8){ cout << "1"; byte-=8;} else cout << "0";
if(byte>=4){ cout << "1"; byte-=4;} else cout << "0";
if(byte>=2){ cout << "1"; byte-=2;} else cout << "0";
if(byte>=1){ cout << "1"; byte-=1;} else cout << "0";
cout << "\n";
}

// private:

int fix[3];
int sensfix[3];
int channel;
int err;
int byte;
int prev_sec;
float prev_ellapsed;

int samplingfreq_ind;
ofSerial *serial;
hand rightHand;
int gestLength;
bool input_gest;
```

```
    menu mGestureOpt;  
    int input_gestLength;  
  
};  
  
#endif
```


hand.cpp

```
#include "hand.h"

hand::hand()
{
    // Set ofVectors to 0
    for (int i=0; i<4; i++){
        for (int j=0; j<BUFFER; j++){
            f[i][j].set(0,0,0);
            ffilt[i][j].set(0,0,0);
            fder[i][j].set(0,0,0);
            ffiltder[i][j].set(0,0,0);
        }
    }

    //Allocate vectors
    position.resize(6);
    static_positions.resize(MAX_SAMPLES);

    gest_t.resize(4);
    gestfiltder_t.resize(4);
    gest_attr.resize(4);

    diff_attr.resize(MAX_SAMPLES);

    attr.resize(MAX_SAMPLES);

    for (int i=0; i<MAX_SAMPLES; i++){
        attr[i].resize(4);
        diff_attr[i].resize(4);
        for (int j=0; j<4; j++){
            attr[i][j].resize(4);
            diff_attr[i][j].resize(4);
        }
        static_positions[i].resize(6);
        if (i<4){
            gest_t[i].resize(BUFFER);
            gestfiltder_t[i].resize(BUFFER);
            gest_attr[i].resize(4);
        }
    }

    // Initalize variables
    difference=1000;
    //snpshot=0;
    //min_diff_stps=1000000;
    average_position;
```

```
    record_available=false;
    loadCalibration();

    // Load presets
    loadPositionPresets();
}

hand::~hand()
{
    //dtor
}

//-----

void hand::update(){

    filterData();
    calcDer();
    calcStaticPos();

    // Move the data inside the array for new data
    for (int i=BUFFER-1; i>0; i--){

        for (int j=0; j<4; j++){
            f[j][i]=f[j][i-1];
            ffilt[j][i]=ffilt[j][i-1];
            fder[j][i]=fder[j][i-1];
            ffiltder[j][i]=ffiltder[j][i-1];
        }
    }

}

//-----
void hand::filterData(){

    // Exponential moving average

    for (int j=0; j<4; j++){
        ffilt[j][0]=ffilt[j][1]*0.9+f[j][0]*0.1;
    }

}

//-----
void hand::calibrate(){
    //neutral_acc=(int)((ffilt[0][0].x+ffilt[0][0].y+ffilt[0][0].z)/3);
    neutral_acc=(int)(ffilt[0][0].z);
    std::ofstream file;
```

```
        file.open("calibration.txt");
        file << neutral_acc;
        file.close();

        cout << "Neutral Acceleration: " << neutral_acc << endl;
    }

//-----
void hand::loadCalibration(){
    std::ifstream file;
    file.open("calibration.txt");
    char c[255];
    int index=0;
    while(!file.eof()){
        file.read(c+index, 1);
        index++;
    }
    c[strlen(c)]='\0';
    file.close();

    neutral_acc=(c[0]-'0')*100+(c[1]-'0')*10+(c[2]-'0');

    cout << "Neutral Acceleration: " << neutral_acc << endl;
}

//-----
void hand::calcDer(){
    // Calculates the derivative of the raw data and the derivative of the filtered.

    for (int i=0; i<4; i++){
        fder[i][0]=f[i][0]-f[i][1];
        ffilterder[i][0]=(ffilt[i][0]-ffilt[i][1]);
    }
}

//-----
void hand::calcAttr(){

    //Attributes
    // - Energy
    // - Log-Attack Time = log10(tampmax - tampmin);
    // - Temporal Centroid = sum(env(t)*t)/sum(env(t))
    // - Zero-Crossing Rate

    //This means that each point (gesture) has 12*4 dimensions

    // Log-Attack Time variables
    float xMax,yMax,zMax,xMin,yMin,zMin;
    int txAmpMax, tyAmpMax, tzAmpMax, txAmpMin, tyAmpMin, tzAmpMin;
```

```
//Time Centroid variables
ofVec3f tcNum,tcDen;

//ZCR variables
bool xSign,ySign,zSign,xPrevSign,yPrevSign,zPrevSign;

for (int i=0; i<4; i++){

    gest_attr[i][0].set(0,0,0);

    // Search for an easy way to define these values automatically
    xMax=-10000;    yMax=-10000;    zMax=-10000;
    xMin=10000;    yMin=10000;    zMin=10000;

    tcNum.set(0,0,0);    tcDen.set(0,0,0);

    if (gestfiltder_t[i][0].x>0)    xPrevSign=true;
    else                            xPrevSign=false;
    if (gestfiltder_t[i][0].y>0)    yPrevSign=true;
    else                            yPrevSign=false;
    if (gestfiltder_t[i][0].z>0)    zPrevSign=true;
    else                            zPrevSign=false;

    gest_attr[i][3].set(0,0,0);

    for (int j=0; j<gest_t_length; j++){

        // Calculate the Energy
        gest_attr[i][0].x=gest_attr[i][0].x+abs(gest_t[i][j].x);
        gest_attr[i][0].y=gest_attr[i][0].y+abs(gest_t[i][j].y);
        gest_attr[i][0].z=gest_attr[i][0].z+abs(gest_t[i][j].z);

        //Find where the max and minimum values are
        if (gest_t[i][j].x>xMax){
            xMax=gest_t[i][j].x;
            txAmpMax=j;
        }
        if (gest_t[i][j].y>yMax){
            yMax=gest_t[i][j].y;
            tyAmpMax=j;
        }
        if (gest_t[i][j].z>zMax){
            zMax=gest_t[i][j].z;
            tzAmpMax=j;
        }

        if (gest_t[i][j].x<xMin){
            xMin=gest_t[i][j].x;
            txAmpMin=j;
        }
        if (gest_t[i][j].y<yMin){
```

```
        yMin=gest_t[i][j].y;
        tyAmpMin=j;
    }
    if (gest_t[i][j].z<zMin){
        zMin=gest_t[i][j].z;
        tzAmpMin=j;
    }

    // Calculate the numerator and denominator of the temporal centroid
    tcNum=tcNum+(j*gest_t[i][j]);
    tcDen=tcDen+(gest_t[i][j]);

    //Determine the sign of the point
    if (gestfiltder_t[i][j].x>0) xSign=true;
    else xSign=false;
    if (gestfiltder_t[i][j].y>0) ySign=true;
    else ySign=false;
    if (gestfiltder_t[i][j].z>0) zSign=true;
    else zSign=false;

    //Calculate the ZCR
    if(xSign!=xPrevSign) gest_attr[i][3].x++;
    if(ySign!=yPrevSign) gest_attr[i][3].y++;
    if(zSign!=zPrevSign) gest_attr[i][3].z++;

    //Define the previous sign
    xPrevSign=xSign;    yPrevSign=ySign;    zPrevSign=zSign;

}

// Calculate the Log-Attack Time
gest_attr[i][1].set(log10(abs(txAmpMax-txAmpMin)),
                   log10(abs(tyAmpMax-tyAmpMin)),
                   log10(abs(tzAmpMax-tzAmpMin)) );

// Calculate the Temporal Centroid
gest_attr[i][2]=tcNum/tcDen;

//cout    << "Energy: " << gest_attr[i][0] << "    LAt: "<<gest_attr[i][1]
//        << "\nTc: " << gest_attr[i][2] << "    Zcr: " << gest_attr[i][3] <<
endl;
}

}

//-----
void hand::calcStaticPos(){
    // Calculates the angles between the 4 fingers and finds a match.
```

```
    /*for (int i=0; i<3; i++){
        position[i]=(ffilt[i][0]-neutral_acc).angleRad((ffilt[i+1][0]-neutral_acc));
    }
    position[3]=(ffilt[3][0]-neutral_acc).angleRad((ffilt[0][0]-neutral_acc));
    position[4]=(ffilt[0][0]-neutral_acc).angleRad((ffilt[2][0]-neutral_acc));
    position[5]=(ffilt[1][0]-neutral_acc).angleRad((ffilt[3][0]-neutral_acc));
*/
    ofVec3f values1[4];
    float length;

    for (int i=0; i<4; i++){
        values1[i]=ffilt[i][0]-neutral_acc;
        length=sqrt(values1[i].x*values1[i].x+values1[i].y*values1[i].y+values1[i].z*values1[i].z);
        values1[i]=values1[i]/length;
    }
    for (int i=0; i<4; i++){
        position[i]=acos( values1[i].x*values1[(i+1)%4].x +
values1[i].y*values1[(i+1)%4].y + values1[i].z*values1[(i+1)%4].z);
    }
    position[4]=acos( values1[2].x*values1[0].x + values1[2].y*values1[0].y +
values1[2].z*values1[0].z);
    position[5]=acos( values1[1].x*values1[3].x + values1[1].y*values1[3].y +
values1[1].z*values1[3].z);

    // Comparison
    float diff=0;
    difference=1000;
    for (int i=0; i<MAX_SAMPLES; i++){
        for (int j=0; j<6; j++){
            diff+=abs(position[j]-static_positions[i][j]);
        }
        if (diff<difference) {
            difference=diff;
            similar_position=i+1;
        }
        diff=0;
    }
    // Consider changing values depending on the sampling rate
    //average_position=average_position*0.95+similar_position*0.05;
    average_position=similar_position;
}

//-----
void hand::storePosition(int id, bool store){
    static_positions[id]=position;
```

```
    // If several snapshots are taken from the same position:

    // Function to change the positions in the program (doesn't affect the presets).
    Stores several
    // snapshots of the same position. Once bool store is set to true the avg is
    calculated.
    // Is it necessary to make an average?

    // There must be an order followed: id must not change if store is false. It can
    change
    // once you set store to true.
    // storePosition(1, false);
    // storePosition(1, false);
    //...
    // storePosition(1, true);
    // storePosition(2, false);
    //...

    // A condition could be implemented in case that the snapshots of the same
    position are very different.

/*
    if (!store){
        if (snpshot==0){
            static_positions[id]=position;
            snpshot++;
        }
        else {
            for (int i=0; i<6; i++){
                static_positions[id][i]+=position[i];
            }
            snpshot++;
        }
    } else {
        if(snpshot!=0){
            for (int i=0; i<6; i++){
                static_positions[id][i]=static_positions[id][i]/snpshot;
            }
        }
        else cout << "Err: none snapshots taken\n";

        snpshot=0;
    }*/
}

//-----
void hand::printPosition(int id){
    // Prints the recorded position in a XML file.
```

```
    ofxXmlSettings XML;
    XML.loadFile("positions.xml");
    XML.clear();

    XML.addTag("STATIC_POSITION");
    XML.pushTag("STATIC_POSITION");

    int posNum=MAX_SAMPLES;
    for (int i=0; i<posNum; i++){
        XML.addTag("ANGLE");
        XML.pushTag("ANGLE", i);
        XML.addValue("ab", static_positions[i][0]);
        XML.addValue("bc", static_positions[i][1]);
        XML.addValue("cd", static_positions[i][2]);
        XML.addValue("da", static_positions[i][3]);
        XML.addValue("ac", static_positions[i][4]);
        XML.addValue("bd", static_positions[i][5]);
        XML.popTag();
    }
    XML.popTag();

    XML.saveFile("positions.xml");
}

//-----
void hand::calcGesture(){
    // Finds a match for the input gesture
    // Normalize vectors? See how much weight do they have
    for (int i=0; i<MAX_SAMPLES; i++){
        if (i<3) cout << "\n\nGesture number: "<< i<<"\n";
        for (int j=0; j<4; j++){
            if (i<3) cout << "\nFinger: "<< j << "\n";
            for (int k=0; k<4; k++){
                diff_attr[i][j][k]=attr[i][j][k].getNormalized()*100-gest_attr[j]
[k].getNormalized()*100;
                if (i<3) cout << diff_attr[i][j][k].length() << ", ";
            }
        }
    }

    //attr[move][finger][attr]
    //gest_attr[finger][attr]
}

//-----
// Stores the gesture attributes in an array (attr)
void hand::storeGest(int id){
    attr[id]=gest_attr;
}
```



```
}

//-----
void hand::print(int numclass){
    // Prints the new move attributes in a XML file. The structure should be like
    this:

    // <MOVE>
    //     <FINGER>
    //         <ENERGY>
    //             <X> x energy </X>
    //             <Y> y energy </Y>
    //             <Z> z energy </Z>
    //         </ENERGY>
    //         <LAT>
    //             <X> x log-attack time </X>
    //             <Y> y log-attack time </Y>
    //             <Z> z log-attack time </Z>
    //         </LAT>
    //         [...]
    //     </FINGER>
    //     [...]
    // </MOVE>

    // Also saves the new move attributes in the CSV file:
    // f1energy_x, f1energy_y, f1energy_z, f1lat_x, f1_lat_y, ... , class
    // Should I create another function to save the clusters? Are the clusters raw
    averages of all the samples?
    // Is knn using clusters or just cheking k-nearest neighbors?

/*
    // The CSV file contains all the move samples.
    std::ofstream CSV;
    CSV.open("moves.csv", std::ios::out | std::ios::app); //Open the CSV file and
    write at the end

    // One XML per group of samples of the same move
    // One XML per all the clusters (attr)

    ofXmlSettings XML;

    char classname[20];
    strcpy(classname, "move");
    classname[4]=(floor(float(numclass)/10)) + '0';
    classname[5] = numclass % 10 + '0';
    classname[6]='\0';

    strcat(classname, ".xml"); // Adds the extension
    classname[10]='\0';
    cout << "\n In rightHand 1: " << classname << endl;
    XML.loadFile(classname);
    cout << "Is this here?" << endl;
```

```
//classname[strlen(classname)-4]='\0'; //Removes the extension
classname[6]='\0';
cout << "\n In rightHand 2: " << classname << endl;
int moveNum=XML.addTag("MOVE");
int fingNum;
int axisNum;

if( XML.pushTag("MOVE", moveNum) ){

    for (int f=0; f<4; f++){

        fingNum=XML.addTag("FINGER");

        if (XML.pushTag("FINGER", fingNum)){

            axisNum = XML.addTag("ENERGY");
            XML.setValue("ENERGY:X", gest_attr[f][0].x, axisNum);
            XML.setValue("ENERGY:Y", gest_attr[f][0].y, axisNum);
            XML.setValue("ENERGY:Z", gest_attr[f][0].z, axisNum);

            CSV << gest_attr[f][0].x << ", " << gest_attr[f][0].y << ", " <<
gest_attr[f][0].z << ", ";

            axisNum = XML.addTag("LAT");
            XML.setValue("LAT:X", gest_attr[f][1].x, axisNum);
            XML.setValue("LAT:Y", gest_attr[f][1].y, axisNum);
            XML.setValue("LAT:Z", gest_attr[f][1].z, axisNum);

            CSV << gest_attr[f][1].x << ", " << gest_attr[f][1].y << ", " <<
gest_attr[f][1].z << ", ";

            axisNum = XML.addTag("TC");
            XML.setValue("TC:X", gest_attr[f][2].x, axisNum);
            XML.setValue("TC:Y", gest_attr[f][2].y, axisNum);
            XML.setValue("TC:Z", gest_attr[f][2].z, axisNum);

            CSV << gest_attr[f][2].x << ", " << gest_attr[f][2].y << ", " <<
gest_attr[f][2].z << ", ";

            axisNum = XML.addTag("ZCR");
            XML.setValue("ZCR:X", gest_attr[f][3].x, axisNum);
            XML.setValue("ZCR:Y", gest_attr[f][3].y, axisNum);
            XML.setValue("ZCR:Z", gest_attr[f][3].z, axisNum);

            CSV << gest_attr[f][3].x << ", " << gest_attr[f][3].y << ", " <<
gest_attr[f][3].z << ", ";

            XML.popTag();
        }
    }
}
```

```
    }
    XML.popTag();
    CSV << classname << std::endl;
    CSV.close();
}

strcat(classname, ".xml"); // Adds the extension
cout << "\n In rightHand 3: " << classname << endl;
XML.saveFile(classname);
classname[strlen(classname)-4]='\0'; //Removes the extension
cout << "\n In rightHand 4: " << classname << endl;
*/
}

//-----
void hand::recordSet(int ingestLength){
    // Stores the incoming stream in gest_t

    int samples_before_gest=4;
    if (ingestLength>BUFFER){
        ingestLength=BUFFER/2;
    }

    for (int i=0; i<4; i++){
        for (int j=0; j<ingestLength+samples_before_gest; j++){
            gest_t[i][j]=ffilt[i][j];
            gestfiltder_t[i][j]=ffiltder[i][j];
        }
    }

    gest_t_length=ingestLength+samples_before_gest;

    /*
    if (once){
        for (int i=0; i<4; i++){
            for (int j=4; j>0; j--){
                gest_t[i].push_back(ffilt[i][j]);
                gestfiltder_t[i].push_back(ffiltder[i][j]);
            }
        }
        once=false;
    }
    for(int i=0; i<4; i++){
        gest_t[i].push_back(ffilt[i][0]);
        gestfiltder_t[i].push_back(ffiltder[i][0]);
    }
    //Remember to clear the gest_t and gestfiltder_t!

    */
}

//-----
void hand::moveStatus(){
```

```
// Detects if the derivate of the filtered input is above or below a threshold

float maxtrig=0;
float mintrig=100;

ofVec3f sum[4];
ofVec3f stopc[4];

for (int i=0; i<4; i++){

    sum[i].x=abs(ffiltder[i][0].x)+abs(ffiltder[i][1].x)+abs(ffiltder[i][2].x)
+abs(ffiltder[i][3].x);
    sum[i].y=abs(ffiltder[i][0].y)+abs(ffiltder[i][1].y)+abs(ffiltder[i][2].y)
+abs(ffiltder[i][3].y);
    sum[i].z=abs(ffiltder[i][0].z)+abs(ffiltder[i][1].z)+abs(ffiltder[i][2].z)
+abs(ffiltder[i][3].z);

    stopc[i].x=sum[i].x+abs(ffiltder[i][4].x)+abs(ffiltder[i][5].x)
+abs(ffiltder[i][6].x)+abs(ffiltder[i][7].x);
    stopc[i].y=sum[i].y+abs(ffiltder[i][4].y)+abs(ffiltder[i][5].y)
+abs(ffiltder[i][6].y)+abs(ffiltder[i][7].y);
    stopc[i].z=sum[i].z+abs(ffiltder[i][4].z)+abs(ffiltder[i][5].z)
+abs(ffiltder[i][6].z)+abs(ffiltder[i][7].z);

    if (sum[i].x>maxtrig) maxtrig=sum[i].x;
    if (sum[i].y>maxtrig) maxtrig=sum[i].y;
    if (sum[i].z>maxtrig) maxtrig=sum[i].z;

    mintrig=stopc[i].x;
    if (stopc[i].y>stopc[i].x&&stopc[i].y>stopc[i].z) mintrig=stopc[i].y;
    if (stopc[i].z>stopc[i].y&&stopc[i].z>stopc[i].x) mintrig=stopc[i].z;

}

if (maxtrig>50){
    record_available=true;
} if (mintrig<2) {
    record_available=false;
}

//cout << "Maxtrig: " << maxtrig << "   Mintrig: " << mintrig << "   Record
available: " << record_available << endl;

}

//-----
void hand::loadPositionPresets(){
    // Read the preset .xml for static positions

    ofXmlSettings XMLload;
```

```
XMLload.loadFile("positions.xml");

XMLload.pushTag("STATIC_POSITION");
int numberOfSavedPoints = XMLload.getNumTags("ANGLE");
for(int i = 0; i < numberOfSavedPoints; i++){
    XMLload.pushTag("ANGLE", i);

    static_positions[i][0] = XMLload.getValue("ab", 0.0);
    static_positions[i][1] = XMLload.getValue("bc", 0.0);
    static_positions[i][2] = XMLload.getValue("cd", 0.0);
    static_positions[i][3] = XMLload.getValue("da", 0.0);
    static_positions[i][4] = XMLload.getValue("ac", 0.0);
    static_positions[i][5] = XMLload.getValue("bd", 0.0);

    XMLload.popTag();
}

XMLload.popTag(); //pop position
}
```

hand.h

```
#ifndef HAND_H
#define HAND_H

#include "ofMain.h"
#include <vector>
#include <cmath> // Log-attack time
#include <fstream> // Write .csv
#include <string> // Change the name of the files
#include "tinyxml.h"
#include "ofXmlSettings.h"

#define BUFFER 1000
#define MAX_SAMPLES 10

class hand
{
public:
    hand();
    virtual ~hand();

    void update();
    void moveStatus();
    void recordSet(int ingestLength);
    void calcAttr();

    void calcGesture();
    void storeGest(int id);
    void print(int innumclass);

    void storePosition(int id, bool store);
    void printPosition(int id);
    void loadPositionPresets();

    void calibrate();

    bool once;
    bool record_available;

    // Static position variables
    // [position] [angles] This must be pre-loaded
    std::vector< std::vector <float> > static_positions;
    // [angle between vectors]
    std::vector< float > position;
    // Difference between actual position and static
    float difference;
    int similar_position;
    float average_position;
    int neutral_acc;

    // [finger][point], from newest to oldest
```

```
    ofVec3f f[4][BUFFER];
    ofVec3f ffilt[4][BUFFER];
    ofVec3f fder[4][BUFFER];
    ofVec3f ffiltder[4][BUFFER];

    // [finger][point], from newest to oldest
    std::vector< std::vector<ofVec3f> > gest_t;
    std::vector< std::vector<ofVec3f> > gestfiltder_t;

    // [finger][attr]
    std::vector< std::vector<ofVec3f> > gest_attr;

    // [move][finger][attr] Stores all the move samples attributes
    std::vector< std::vector< std::vector<ofVec3f> > > attr;
    std::vector< std::vector< std::vector<ofVec3f> > > diff_attr;

protected:
private:

    void filterData();
    void calcDer();
    void calcStaticPos();

    void loadCalibration();
    void saveCalibration();

    int snpshot;
    int gest_t_length;

};

#endif // HAND_H
```

menu.cpp

```
#include "menu.h"

menu::menu(int innumButtons, int id)
{
// Basic constructor
    numButtons=innumButtons;

    char name[12];
    name[0]=(floor(float(id)/10)) + '0';
    name[1] = id % 10 + '0';
    name[2]='/';

    ofColor instroke(0,50,50);
    ofColor inside(0,255,255);

    for (int i=0; i<numButtons; i++){

        myButtons[i].w=100;
        myButtons[i].h=100;
        myButtons[i].y=(50+floor(i/4)*150);
        myButtons[i].x=(50+(i%4)*150);
        myButtons[i].r=15;
        myButtons[i].c_stroke=instroke;
        myButtons[i].c_inside=inside;

        name[3]=(floor(float(i)/10)) + '0';
        name[4] = i % 10 + '0';
        name[5]='\0';

        strcat(name, ".png");
        settings[i].loadImage(name);

    }
}

menu::menu(int innumButtons, int id, int inx, int iny, int inw, int inh, int sepw, int
seph, bool intexture)
{
// Complex constructor

    texture=intexture;
    numButtons=innumButtons;
    x=inx; y=iny;

    char name[12];
    if (texture){
        name[0]=(floor(float(id)/10)) + '0';
        name[1] = id % 10 + '0';
        name[2]='/';
```



```
    }

    int buttRows=0;

    while(floor(buttRows*(sepw+inw)/ofGetWidth())<1){
        buttRows++;
    }
    buttRows--;

    ofColor instroke(0,50,50);
    ofColor inside(0,255,255);

    for (int i=0; i<numButtons; i++){

        myButtons[i].w=inw;
        myButtons[i].h=inh;
        myButtons[i].y=(y+sepw+floor(i/buttRows)*(inw+sepw));
        myButtons[i].x=(x+seph+(i%buttRows)*(inh+seph));
        myButtons[i].r=15;
        myButtons[i].c_stroke=instroke;
        myButtons[i].c_inside=inside;

        if (texture){
            name[3]=(floor(float(i)/10)) + '0';
            name[4] = i % 10 + '0';
            name[5]='\0';

            strcat(name, ".png");
            settings[i].loadImage(name);
        }

    }

}

menu::menu(){
    numButtons=0;
}

menu::~menu()
{
    //dtor
}

void menu::draw(){

    for (int i=0; i<numButtons; i++){
        myButtons[i].draw();
        if (texture){
            ofEnableAlphaBlending();
            ofSetColor(0, 255, 255, 255);
        }
    }
}
```

```
        settings[i].draw(myButtons[i].x, myButtons[i].y, myButtons[i].w,  
myButtons[i].h);  
        ofDisableAlphaBlending();  
    }  
}  
  
}
```

menu.h

```
#ifndef MENU_H
#define MENU_H

#include "ofMain.h"
#include "button.h"

class menu
{
public:
    menu(int innumButtons, int id);
    menu(int innumButtons, int id, int inx, int iny, int inw, int inh,
        int sepw, int seph, bool intexture);
    menu();
    virtual ~menu();

    void draw();

    int numButtons;
    button myButtons[20];

    ofImage settings[20];

protected:
private:

    bool texture;
    int x,y;

};

#endif // MENU_H
```

button.cpp

```
#include "button.h"

button::button(int inx, int iny, int inw, int inh, ofColor instroke, ofColor ininside)
{
    // Basic constructor
    x=inx; y=iny; w=inw; h=inh; r=15;
    b_sel=false; textin=false; onit=false;
    c_stroke=instroke; c_inside=ininside;
}

button::button(int inx, int iny, int inw, int inh, int inr, ofColor instroke, ofColor
ininside)
{
    // Button without text
    x=inx; y=iny; w=inw; h=inh; r=inr;
    b_sel=false; textin=false; onit=false;
    c_stroke=instroke; c_inside=ininside;
}

button::button(int inx, int iny, int inw, int inh, int inr,
                bool intextin, char *intext, ofColor instroke, ofColor ininside,
ofColor inctext)
{
    // Button with text
    // The char *intext must have as a last character ~

    verdana18.loadFont("verdana.ttf", 18, true, true);
    verdana18.setLineHeight(25.0f);
    verdana18.setLetterSpacing(1.000);

    int i=0;
    while (intext[i]!='~'){
        text[i]=intext[i];
        i++;
    }
    text[i]='\0';

    if (verdana18.stringHeight(text)+16>inw) w=verdana18.stringWidth(text)+16;
    else w=inw+16;
    if (verdana18.stringHeight(text)+16>inh) h=verdana18.stringHeight(text)+16;
    else h=inh+16;

    x=inx-8; y=iny-verdana18.stringHeight(text)-8; r=inr;
```

```
    b_sel=false; textin=intextin; onit=false;
    c_stroke=instroke; c_inside=ininside; c_text=inctext;
}

button::button()
{
    // Default constructor
    x=0; y=0; w=50; h=50; r=0;
    b_sel=false; textin=false; onit=false;
    c_stroke.set(0,0,0); c_inside.set(0,0,0);
}

button::~button()
{
    //dtor
}

void button::update(){
}

void button::draw(){
    // Permanently marked
    if (onit){
        ofSetColor(c_inside);
        ofFill();
        ofRectRounded(x,y,w,h,r);
    }

    // Text display
    if (textin){
        ofSetColor(c_inside);
        ofFill();
        if (b_sel){ ofNoFill();}
        ofRectRounded(x, y, w, h, r);
        ofSetColor(c_text);
        verdana18.drawString(text, x+8,y+verdana18.stringHeight(text)+8);
    }

    // Fading
    if (b_sel){
        ofEnableAlphaBlending();
        ofSetColor(c_inside, abs(int(ofGetElapsedTimeMillis()/10)%100-50)+30);
        ofFill();
        ofRectRounded(x, y, w, h, r);
    }
}
```

```
        ofDisableAlphaBlending();  
        ofSetLineWidth(2);  
    }  
  
    ofSetColor(c_stroke);  
    ofNoFill();  
    ofRectRounded(x, y, w, h, r);  
    ofSetLineWidth(1);  
}
```

button.h

```
#ifndef BUTTON_H
#define BUTTON_H

#include "ofMain.h"

class button
{
public:
    button(int inx, int iny, int inw, int inh, ofColor instroke, ofColor
ininside);
    button(int inx, int iny, int inw, int inh, int inr, ofColor instroke, ofColor
ininside);
    button(int inx, int iny, int inw, int inh, int inr,
        bool intextin, char *intext, ofColor instroke, ofColor ininside,
ofColor inctext);

    button();
    virtual ~button();

    void update();
    void draw();

    int x,y,w,h,r;
    bool b_sel, onit, textin;
    char text[100];

    ofColor c_stroke, c_inside, c_text;
    ofTrueTypeFont verdana18;

protected:
private:

};

#endif // BUTTON_H
```

pSystem.cpp

```
#include "pSystem.h"

pSystem::pSystem(){
}

pSystem::pSystem(float inx, float iny, float inz, int inNumP, ofImage* inImage){
    numP=inNumP;
    myParticles.resize(numP);

    img=inImage;

    for (int i=0; i<numP; i++){
        myParticles[i]= Particle(inx, iny, inz, img);
    }
}

pSystem::~pSystem()
{
    //dtor
}

void pSystem::update(float x, float y, float z){
    // 3D Data Representation
    for (int i=0; i<numP; i++){
        myParticles[i].update();
        if (myParticles[i].isDead()){
            myParticles[i]=Particle(x,y,z,img);
        }
    }
}

void pSystem::updateExpl(){
    // Explosion
    for (int i=0; i<numP; i++){
        myParticles[i].updateExpl();
    }
}

void pSystem::draw(){
    for (int i=0; i<numP; i++){
        myParticles[i].draw();
    }
}
```



```
void pSystem::setExplosion(){
    for (int i=0; i<numP; i++){
        myParticles[i].velocity*=ofPoint(ofRandomf()*0.5+1,ofRandomf()*0.5+1,ofRandomf
()*0.5+1);
    }
}
```

pSystem.h

```
#ifndef PSYSTEM_H
#define PSYSTEM_H

#include "ofMain.h"
#include "Particle.h"

class pSystem
{
public:
    pSystem();
    pSystem(float inx, float iny, float inz, int inNumP, ofImage* inImage);
    virtual ~pSystem();

    void update(float x, float y, float z);
    void updateExpl();
    void draw();
    void setExplosion();

    std::vector <Particle> myParticles;
    int numP;
    ofImage* img;
};

#endif // PSYSTEM_H
```

Particle.cpp

```
#include "Particle.h"

Particle::Particle(){
    x=0;    y=0;    z=0;
    life=255;
    gravity.x=-0.1;

    int angle=rand()%1000;
    velocity.set(ofRandomf(), cos(angle), sin(angle));
}

Particle::Particle(float inx, float iny, float inz, ofImage* inImage){
    iniX=inx;    iniY=iny;    iniZ=inz;
    x=inx;    y=iny;    z=inz;
    life= rand()%200 + 55;
    gravity.x=-0.1;

    img = inImage;

    int angle=rand()%1000;
    velocity.set(ofRandomf(), cos(angle), sin(angle));
}

Particle::~Particle()
{
    //dtor
}

void Particle::update(){
    // Update motion
    life-=4;
    velocity+=gravity;

    if (y>iniY) velocity.y-=0.1;
    else velocity.y+=0.1;
    if (z>iniZ) velocity.z-=0.1;
    else velocity.z+=0.1;
    x+=velocity.x;
    y+=velocity.y;
    z+=velocity.z;
}

void Particle::updateExpl(){
    // Update explosion motion
    if (life>4) life-=4;
    velocity+=velocity*0.1;
    x+=velocity.x;
    y+=velocity.y;
    z+=velocity.z;
}
```

```
void Particle::draw(){

    ofEnableAlphaBlending();
    ofSetColor(255,255,255, rand() % life);
    int rad_text=rand() % 5 + 10;
    // Draw in the three axes
    ofPushMatrix();
        ofTranslate(x,y,z);
        img->draw(-rad_text/2, -rad_text/2, 0, rad_text, rad_text);
        ofRotateX(90.0f);
        img->draw(-rad_text/2, -rad_text/2, 0, rad_text, rad_text);
        ofRotateY(90.0f);
        img->draw(-rad_text/2, -rad_text/2, 0, rad_text, rad_text);
    ofPopMatrix();
    ofDisableAlphaBlending();
}

bool Particle::isDead(){
    if (life<1) return true;
    else return false;
}
```

Particle.h

```
#ifndef PARTICLE_H
#define PARTICLE_H

#include "ofMain.h"

class Particle
{
public:
    Particle();
    Particle(float inx, float iny, float inz, ofImage* inImage);
    virtual ~Particle();

    float iniX, iniY, iniZ, x, y, z;
    void update();
    void updateExpl();
    void draw();
    bool isDead();

    ofPoint gravity;
    ofPoint velocity;

private:
    ofImage* img;
    int life;
};

#endif // PARTICLE_H
```

holes.cpp

```
#include "holes.h"

Holes::Holes(){
    x=-22;    y=-22;
}

Holes::Holes(int inx, int iny)
{
    x=inx;    y=iny;
}

Holes::~Holes()
{
    //dtor
}

void Holes::draw(){
    ofSetColor(0,0,0);
    circle(x, y, 22, 8, true);
}

void Holes::drawExit(){
    // Exit point
    ofSetColor(200,0,0);
    circle(x,y,12, 8, true);
    ofNoFill();
    ofRect(x-21,y-21, 42, 42);
}

void Holes::circle(int x, int y, float rad, float res, bool opt_fill){
    // Improved circle drawing

    if (opt_fill){ circle(x,y,rad+0.1,res,false); ofFill(); }
    else ofNoFill();

    ofBeginShape();
    for (int i=0; i<(int)(360.0/res+1); i++){

        ofVertex( x + cos((i*res)*PI/180)*rad,
                  y + sin((i*res)*PI/180)*rad);
        ofVertex(x + cos((i*res+res/2)*PI/180)*rad,
                  y + sin((i*res+res/2)*PI/180)*rad );
    }
    ofEndShape();
}
```


holes.h

```
#ifndef HOLES_H
#define HOLES_H

#include "ofMain.h"

class Holes
{
public:
    Holes();
    Holes(int inx, int iny);
    virtual ~Holes();
    int x, y, r;
    void draw();
    void drawExit();

protected:
private:
    void circle(int x, int y, float rad, float res, bool opt_fill);
};

#endif // HOLES_H
```


APPENDIX B - List of figures

Figure 1. Digital Data Entry Glove	6
Figure 2. Power Glove	7
Figure 3. DataGlove	7
Figure 4. CyberGlove II	8
Figure 5. CyberGrasp	8
Figure 6. CyberForce	9
Figure 7. 5DT Data Glove with the wireless kit	10
Figure 8. P5 Glove	10
Figure 9. Shape Hand in different positions	11
Figure 10. Shape Hand with the arm accessory	11
Figure 11. The Peregrine	12
Figure 12. Pinchglove	12
Figure 13. Didjiglove	12
Figure 14. DGTech Vhand	13
Figure 15. X-IST Data Glove	13
Figure 16. Acceleglove	14
Figure 17. KeyGlove	15
Figure 18. Ben's Glove of Power	15
Figure 19. Clove 2	15
Figure 20. Mister Gloves	16
Figure 21. Device placement diagram	19
Figure 22. Teensy 2.0	19
Figure 23. Accelerometer	20
Figure 24. XBee S1	20
Figure 25. Xbee Explorer Regulated	20

Figure 26. Polymer Lithium Ion Battery	21
Figure 27. Static information	22
Figure 28. First Maya prototype with the 3D Paint Tool. Right hand	25
Figure 29. Maya circuit design with the paint brush. Left hand	25
Figure 30. Stainless steel thread	26
Figure 31. Plated silver thread	26
Figure 32. Parallel paths sewing technique	27
Figure 33. Crossing sewing technique	28
Figure 34. Parallel lines sewing technique	28
Figure 35. Back of the right glove	29
Figure 36. Palm of the right glove	29
Figure 37. Back of the left glove	29
Figure 38. Palm of the left glove	29
Figure 39. Glove with hook-up wires	30
Figure 40. Wireless communication diagram (1)	32
Figure 41. Wireless communication diagram (2)	33
Figure 42. Wireless communication diagram (3)	34
Figure 43. Communication process diagram	40
Figure 44. Information about orientation of each finger	43
Figure 45. Relations between accelerometers	43
Figure 46. Gravity vs Acceleration	44
Figure 47. Data segmentation/windowing algorithm	45
Figure 48. Gesture attribute points (2D) with an income gesture	47
Figure 49. Peak and valley of a signal	48
Figure 50. Temporal centroids of two signals	48
Figure 51. CNN data reduction	50
Figure 52. First communication setup	51

Figure 53. Interface structure of the program	54
Figure 54. Main menu	55
Figure 55. Configuration of the serial port	56
Figure 56. Particle system	57
Figure 57. Labyrinth Game	58
Figure 58. Position storing and loading	59
Figure 59. Gesture recording, storing and loading	60
Figure 60. MIDI data selection	61
Figure 61. Upper bar (left)	62
Figure 62. Upper bar (right)	62

APPENDIX C - Data Compact Disc

One compact disc is included in this thesis. It contains this paper in a pdf format, the interface software and the Teensy source codes.

Arduino

The codes for the two Teensy microcontrollers used in the presentation are included.

Final App

Contains the debugged application. It is the user-end application.

mySketch

This folder contains all the source codes and the Code::Blocks project.

BIBLIOGRAPHY

- Abolfathi, Peter P., Interpreting sign language is just the beginning for the AcceleGlove open source dataglove, *Gizmag website*, July 23, 2009, Retrieved 2013-06-14
<www.gizmag.com/acceleglove-open-source-dataglove/12252/>
- AnthroTronix Inc., AnthroTronix YouTube Channel, *YouTube*, Retrieved 2013-06-14
<www.youtube.com/user/AnthroTronix>
- Anderson, D., Bailey, C. and Skubic, M. (2004) Hidden Markov Model symbol recognition for sketch-based interfaces. *AAAI Fall Symposium*. Menlo Park, CA: AAAI Press, 15-21.
- ABC editor - "Backwards Compatible - The Power Glove". ABC website - Good Game. *Australian Broadcasting Corporation (ABC)*. 19 May 2008. Retrieved 2009-06-06.
<www.abc.net.au/tv/goodgame/stories/s2248843.htm>
- Banzi, M., Cuartielles, D., Zambetti, N., Arduino, 2012, Retrieved 2013-04-05
<www.arduino.cc/>
- Biotecmexico, 'P5 Glove', video, *YouTube*, 7th of Nov 2007, Retrieved 2013-05-08
<www.youtube.com/watch?v=hoKD-R1zlpw>

- Ben Heck Show, The, Episode Power Glove for Xbox, video, *Revision 3 Internet Television* March 6, 2012, Retrieved 2013-04-18
<tv.revision3.com/tbhs/kinect-glove>
- Benbasat, A.Y., Paradiso, J.A., An inertial measurement framework for gesture recognition and applications. *Gesture and Sign Language in Human-Computer Interaction, International Gesture Workshop, 2001.*
- Bruning, Lynne, eTextile Lounge, Youtube Channel, *YouTube*, Retrieved 2013-04-14
<www.youtube.com/user/LynneBruning>
- Buechley, L., SparkFun Electronics, LilyPad Arduino, Retrieved 2013-04-12,
<arduino.cc/en/Main/arduinoBoardLilyPad>
- Cemeteck, Mitchell, C., Clove 2 (Cemeteck Bluetooth Dataglove), *Cemeteck*, July 8, 2008, Retrieved 2013-04-18
<www.cemeteck.net/projects/item.php?id=16#s4>
- Chen, S., Levine, E., Mister Gloves - A Wireless USB Gesture Input System, *Cornell University*, 2010, Retrieved 2013-04-18
<courses.cit.cornell.edu/ee476/FinalProjects/s2010/ssc88_egl27/>
- Coon, R., Stoffregen, P., Teensy, *PJRC Electronic Projects Components Available Worldwide*, 2012 - Retrieved 2013-04-05
<www.pjrc.com/teensy/>

- CyberGlove Systems, CyberGlove II Wireless Glove, 2010, Retrieved 2013-04-05
<cyberglovesystems.com/?q=products/cyberglove-ii/overview>
- CyberGlove Systems, CyberGrasp, 2010, Retrieved 2013-04-05
<cyberglovesystems.com/products/cybergasp/overview>
- CyberGlove Systems, CyberGrasp, 2010, Retrieved 2013-04-05
<cyberglovesystems.com/products/cybergasp/overview>
- David R. Butenhof, Programming with POSIX Threads, *Addison-Wesley Professional*, 1997
- Digi International Inc, XBee, 2012, Retrieved 2013-04-05
<<http://www.digi.com/xbee/>>
- Digi International Inc, X-CTU Software, 2012, Retrieved 2013-04-21
<www.digi.com/support/productdetail?pid=3352&osvid=57&type=utilities>
- DubSpot, 'Ableton Live + Nintendo Power Glove: Meet Controllerist Yeuda Ben-Atar aka Side Brain @ Dubspot', video, *YouTube*, Oct 23 2012, Retrieved 2013-05-08
<www.youtube.com/watch?v=J1hiacj1R2Y>
- Essential Reality, P5 Gaming Glove, 2009, Retrieved 2013-05-08
<www.gizmag.com/go/1148/>

- Fakh Hamad, O., Analog, Digital and Multimedia Telecommunications: Basic and Classic Principles, *Xlibris Corporation*, 2011
- Fifth Dimension Technologies, The 5th Glove, Retrieved 2013-04-05
<www.5dt.com/hardware.html#glove>
- Hernandez-Rebollar, Jose L., Kyriakopoulos, N., Lindeman, Robert W. (2002) The AcceleGlove: a whole-hand input device for virtual reality, *George Washington University*
- Inition Co. 2001, Retrieved 2013-04-28
<inition.co.uk/3D-Technologies/productsection/43>
- Inition Co., Fakespace Labs PINCH Glove, Retrieved 2013-04-28
<inition.co.uk/3D-Technologies/fakespace-labs-pinch-gloves>
- Inition Co., Didjiglove, Retrieved 2013-04-28
<inition.co.uk/3D-Technologies/didjiglove>
- Inition Co., DGTech Vhand, Retrieved 2013-04-28
<inition.co.uk/3D-Technologies/dgtech-vhand>
- Inition Co., X-IST Data Glove, Retrieved 2013-04-28
<inition.co.uk/3D-Technologies/x-ist-data-glove>

- Iron Will Innovations, The Peregrine, 2012, Retrieved 2013-05-08
<theperegrine.com>
- Kessler G.D, Hodges L.F, Walker N., Evaluation of the CyberGlove as a Whole Hand Input Device, *ACM Transactions on Computer-Human Interaction*. 2(4), 1995, pp. 263-283.
- Lieberman, Z., Watson T., Castro, A., openFrameworks, *MIT License*, 2013, Retrieved 2013-04-25
<www.openframeworks.cc/>
- Little Bird Electronics , Triple Axis Accelerometer Breakout - ADXL335, 2012, Retrieved 2013-04-05
<littlebirdelectronics.com/products/breakout-3-axis-analog-accelerometer-adxl335>
- MIDI Manufacturers Association Inc, MIDI Messages, Retrieved 2013-05-06
<www.midi.org/techspecs/midimessages.php>
- Minority Report, 2002. Film. Directed by Steven Spielberg. USA: *Twenty Century Fox Film*
- Mattel Co., The PowerGlove, *Nintendo Entertainment System*, Retrieved 2013-04-01
<www.ebay.com/bhp/nintendo-power-glove>
- Measurand Inc, ShapeHand, 2009, Retrieved 2013-05-08
<www.finger-motion-capture.com/shapehand.html>

- O. Wobbrock, J., D. Wilson, A., Li, Yang (2007) Gestures without Libraries, Toolkits or Training: A \$1 Recognizer for User Interface Prototypes, *Columbia University*
- PCVR Magazine, How to build an instrumented glove based on the Powerglove flex sensors. *PCVR 16 pp 10–14. Stoughton, WI, USA: PCVR Magazine, 1994*
- Peeters, G., Rodet, X. (2002) Automatically selecting signal descriptors for Sound Classification, *Pennsylvania State University*
- Peeters, G., Giordano, B., Susini P. and Misdariis N., McAdams, S. (2010) The Timbre Toolbox: Extracting audio descriptors from musical signals, *Acoustical Society of America*
- Pinouts.ru, RS-232 and other serial ports and interfaces pinouts, 2009, Retrieved 2013-05-05
<pinouts.ru/SerialPorts>
- Puckette, M., Pd-Extended, *Pure Data*, Retrieved 2013-02-02
<puredata.info/downloads/pd-extended>
- Reas, C., Benjamin, F., Processing, *MIT Media Lab*, 2013, Retrieved 2013-02-02
<www.processing.org/>
- Rowberg, J., Keyglove, freedom in the palm of your hand, Keyglove website, Retrieved 2013-05-09
<www.keyglove.net>

- Rowberg, J., Keyglove Promo #01, video, *Vimeo*, 2013 Retrieved 2013-04-05
<vimeo.com/59319446>
- Rubine, D. (1991) Specifying gestures by example. Proc. SIGGRAPH '91. New York: ACM Press, 329-337
- Sparkfun, XBee 1mW Chip Antenna - Series 1 (802.15.4), 2012, Retrieved 2013-04-05,
<www.sparkfun.com/products/8664>
- Sparkfun, XBee Explorer Regulated WRL -09132, 2012, Rerieved 2013-04-05
<www.sparkfun.com/products/9132>
- Sparkfun, Polymer Lithium Ion Battery - 2000mAh PRT-08483, 2012, Rerieved 2013-04-05
<www.sparkfun.com/products/8483>
- Sparkfun, Conductive Thread (Thin) - 50' DEV-10118, 2012 - Retrieved 2013-04-05
<www.sparkfun.com/products/10118>
- Sparkfun, Conductive Thread 117/17 2ply DEV-08544, 2012 - Retrieved 2013-04-05
<www.sparkfun.com/products/8544>
- SparkFun, Hook-up Wire, 2012 - Retrieved 2013-06-05
<<https://www.sparkfun.com/products/8022>>

- Sturman, D.J., Zeltzer, D. (January 1994). A survey of glove-based input. *IEEE Computer Graphics and Applications*
- Tham. M.T., Dealing with measurement noise (A gentle introduction to noise filtering), *School of chemical Engineering and Advanced Materials, Newcastle University*
- University of Waikato, Weka 3: Data Mining Software, 2013, Retrieved 2013-05-15
<www.cs.waikato.ac.nz/ml/weka/index.html>
- Zimmerman T.G and Lanier J, Computer data entry and manipulation apparatus method, Patent Application 5,026,930, *VPL Research Inc*, 1992.
- Zoltan Prekopcsák (2008), Accelerometer Based Real-Time Gesture Recognition, *Budapest University of Technology and Economics*

