Team D2: Superhans

~~Engineering Professional Studies~~ Engineering Professional Studies
~~–~~University of Surrey

Supervisor: Dr. Kevin Wells

Abstract (JB)

This report outlines the initial design, prototyping and testing of the SuperHans game controller device. SuperHans is an open source game controller that aims to be cross compatible with all gaming platforms. The design is completely open to the public via a step by step building guide hosted on www.instructables.com. All code used in the device is available online: the .NET GUI framework on www.github.com and the embedded C++ on mbed.org. This openness aims to encourage global collaboration for future development. In the main body of the report itself role allocation is discussed for each member of the 6 man development team as well as an analysis of the initial Gantt chart and any revisions that might have been made. The challenges related to development of each part of the product are then outlined. The decisions that led to the eventual completion of working prototypes are justified. Some of the less tangible concepts of the device are explored including ethical issues associated with the device, sustainability aspects of the development ecosystem and results from beta testing events. A business overview of the production is then outlined including quality and manufacturing aspects. A final section will discuss some recommended business practices that will be suit our group to ensure survival of the product. To conclude, overall strength and limitations of the device are examined.

Contents

1. Introduction (All)

Superhans is gesture recognition interface for gaming platforms. This device enhances gameplay experience with the lowest projected cost compared to commercial competitors. An initial design overview is covered in the design brief in Appendix A.

No company has yet monopolised the market. Peregrine Wearable Interface [1], Ion Wireless Air Mouse Glove [2] and P5 Gaming Glove [3] are the three main competitors. Each product suffers physical constraints. Each glove is compatible as either a mouse, keyboard or finger gesture interface, with restrictions on game and operating system (OS) compatibility. Superhans is the first product to integrate all these solutions. In addition to this, competitors have a price overhead compared to generic gamepads. Existing interfaces also lack an online and developing community compared to Superhans open source development and customer/developer support.

The team consists of 6 members, whom were given tasks accordingly to their engineering traits and specialisations. Jason Brewer (JB) was responsible for haptic feedback, button integration, and acted as operational director. Luke Testa (LT) was responsible for the graphical user development (GUI) development, front-end technology and acted as the quality control manager. Hans Fernando (HF) was responsible for the glove fabrication, sensor development, hardware integration, and took the role of the procurement manager. Anurag Dhutti (AD) was responsible for the embedded programming and acted as the financial officer of the group. Rashid Al-Mohanadi (RM) was responsible for the sustainability, assisted in hardware integration and business consultancy. Tristan Hughes (TH) was responsible for embedded architecture, software engineering and acted as the manufacturing manager. Accordingly all the physical tasks of the project were evenly distributed and carried out according to a well-planned Gantt chart.

This document outlines a system overview, system architecture, design principle and project management section. The former provides a high-level overview of the product functionality and operations. A user manual for product installation/setup and frequently asked questions (FAQ) document is found in Appendix B and C accordingly. The second discusses hardware architecture and gesture recognition. The third principle illustrates marketing and manufacturing. The latter is responsible for front-end technology - user interface and technical customer support. Each of the peripherals of the device (buttons, flex sensors and haptic feedback) is discussed by first introducing their physical hardware implementation followed by their software implementation. Example C++ code can be found in Appendix D.

Each section of this report will contain the initials of the author for marking purposes. Team sections are marked by 'all'.

2. System Overview (All)

Superhans is hand fitted multi-platform gaming controller. Superhans replaces the traditional analogue sticks/mouse interface of gamepads. It utilises a combination of accelerometer and sensor technology triggered by hand gestures. The details of this proposal are found in the design brief contained within the Appendix A of this document.

The device is designed to be a gaming device to combine the most common controls of a mouse and keyboard combination in one. In summary the product will replace universal serial bus (USB™) peripheral devices. The mouse and key presses are controlled simultaneously by tilting the hand, bending the index, middle and ring fingers and touching the thumb to various buttons placed on the fingers of the glove. This allows a wide variety of keys to be held with simultaneous presses ensuring compatibility with most games.

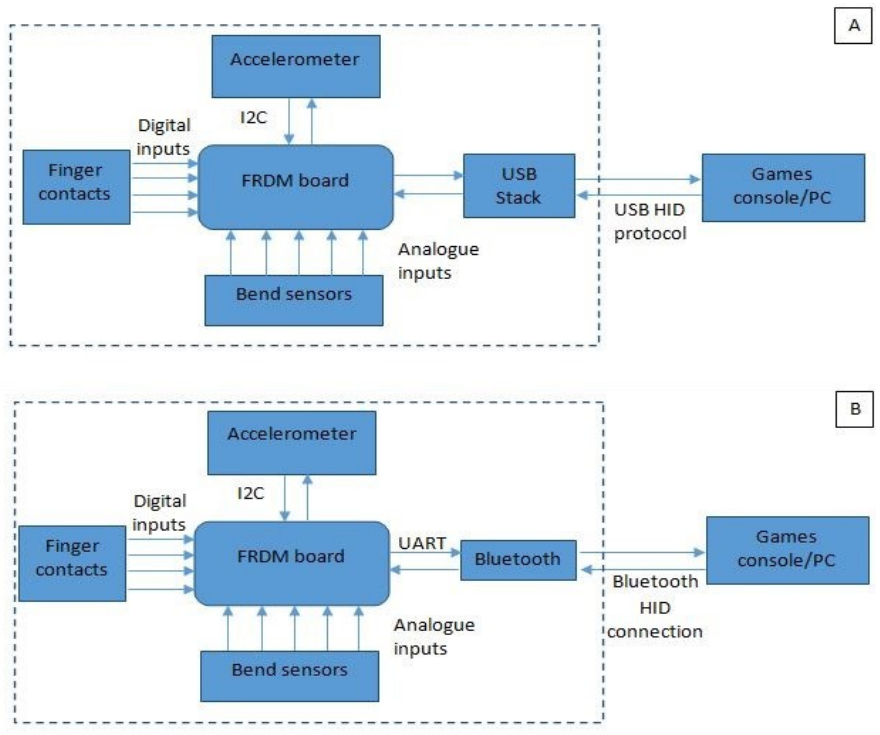


Figure 1: Block diagram of the USB (A) and wireless (B) based system. The diagram shows the types of connections between the different hardware components and an external device.

Compatibility

Superhans does not require any extra third party devices to work as a keyboard and mouse replacement for Windows, Mac and Linux platforms. At present it does not act as a replacement to the gaming controllers commonly used on the Sony Playstation and Microsoft Xbox consoles. There are a variety of products available that convert the keyboard and mouse input into signals recognisable by these platforms.

3. System Architecture

3.1 Hardware Architecture (TH)

In recent years microcontrollers have started to significantly reduce in cost whilst still increasing in functionality. There are now many different types of microcontrollers available on the market. The three main types currently available are the Microchip PIC, ATMEL AVR and the ARM® Cortex™.

These provide an easy to use program interface to simplify the control of the microcontrollers. One of the most common devices is the Arduinos. A similar product called mbed exists for ARM® based microcontrollers.

Arduino is open-source hardware available in several different designs [4]. Each design features the same standard headers to allow extra peripherals known as ‘shields’ to be attached, in order to increase functionality. The code is developed through an Integrated Development Environment (IDE) that simplifies the setup of the microcontroller. The majority of Arduinos are based around an 8-bit microcontroller and contain a 16MHz crystal oscillator and a 5V voltage regulator, making it easy to start the product development.

The mbed is similar to Arduino except it is based around the 32-bit ARM® Cortex™ processors [5]. The mbed platform uses an online compiler, which provides the user with a binary file for drag and drop flash programming, in order to allow for rapid prototyping. mbed offers a range of different devices depending on the application. The lowest power option is the freedom (FRDM) board, designed by Freescale. The FRDM board is based on the Cortex-M0+ and features a large selection of peripherals including USB™, 16-bit analogue to digital converter (ADC) and a capacitive touch sensing input.

3.1.1 Use of mbed

For this application a low cost platform with an easy access to USB™ connections, high accuracy analogue inputs and serial universal asynchronous receiver/transmitter (UART) connections were needed. It would be beneficial to have a fast processor to reduce delay. After analysing the different platforms available the FRDM board was decided as the most appropriate. The product can be brought to the market quickly, due to its low cost of £8.25 and its ease of use.

3.1.1.1 Working with the FRDM-KL25Z Development Board

The FRDM-KL25Z board provided an excellent base to work from. The board breaks out all the relevant interfaces into a set of Arduino compatible headers. It also has an accelerometer mounted onto the board which is connected directly to the microcontroller; one of the key components required for this product. The board also breaks out the microcontroller's on board USB™ module to a micro USB™ connector, allowing the board to operate as a native USB™ device. For the debug and programming of the board, a second microcontroller is used. This microcontroller supports drag and drop programming of binary files and the ARM® Cortex™ Microcontroller Software Interface Standard (CMSIS-DAP) programming and debug standard used by many professional tool chains.

3.1.1.2 Use of Accelerometer

To determine the orientation of the user's hand the FRDM board's accelerometer was used. The accelerometer is the low cost MMA8451Q from Freescale, which provided us with high accuracy readings in a small package. The sensors provided data from 3 axes with 14 bit resolution.

The microcontroller was connected to the accelerometer using an inter-integrated circuit (I^2C) bus. The bus allows for bidirectional communication link between several devices at speeds of up to 2.25MHz. I^2C works on a pair of digital pins known as serial data line (SDA) and serial clock (SCL). Both the lines are pulled up to Vdd with a resistor. The microcontroller acts as the master on the bus, while the sensor is configured as a slave device. Each slave is addressed using a 7-bit I^2C address. Multiple slaves can be connected to the bus if their 7-bit addresses do not overlap.

The accelerometer was set up to continuously update the x, y and z registers at 400 Hz, thus allowing application code to read the registers at 100 Hz thus reducing latency.

The values were read into the microcontroller by sending a data request over the I^2C bus. The data was then returned in the form of two bytes. These bytes were then combined together into a standard integer value. Finally, the integer was converted into a signed float from a value ranging between 1 and -1.

3.1.2 Why USB™ or Bluetooth®?

To connect Superhans to a host device two options were considered: USB™ and Bluetooth®. The USB™ option requires a physical wire connecting Superhans to a host system and the Bluetooth® option, will allow a wireless connection. Both models use the human interface device (HID) protocol to communicate.

3.1.2.1 USB™

The USB™ connection operates in a similar way to a standard keyboard and mouse. When connected to Superhans, the device is powered up and emulates a USB™ Keyboard and mouse combination device, thus allowing the use of standard built in driver software that is available on nearly all operating systems.

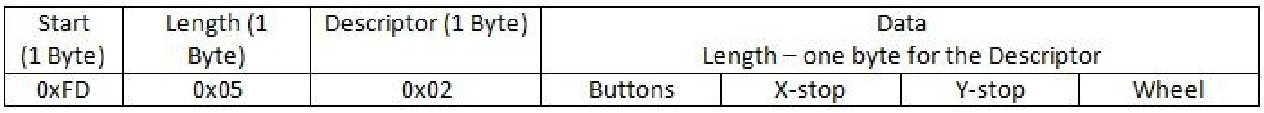
3.1.2.2 HID

HID is a bi-directional protocol which allows the communication with devices that humans can operate. The HID protocol is a standard designed by the USB™ committee. There is a lightweight version implemented for use over Bluetooth® communications.

When a HID device first connects to the host system, the device descriptor data is passed to the host system. The device descriptor details many things such as device type, USB™ version and manufacturer. The host then requests the HID descriptor data, which details exactly how the device is going to form the HID reports.

After the initialisation, the device is able to start sending HID reports to the host. A HID report is only sent when interaction between the human and interface is detected. A HID report formed is as shown below:

3.1.2.3 Mouse

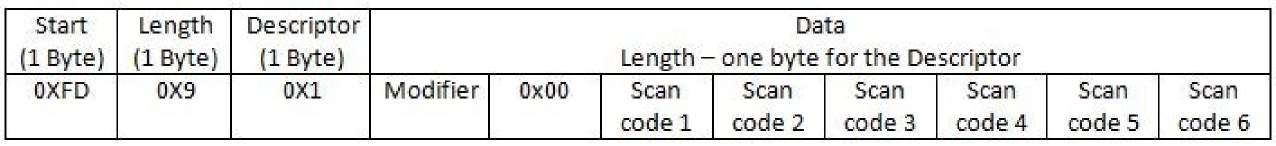


Start (1 Byte)	Length (1 Byte)	Descriptor (1 Byte)	Data Length – one byte for the Descriptor			
0xFD	0x05	0x02	Buttons	X-stop	Y-stop	Wheel

Figure 2: The composition of a mouse HID frame [6].

The start byte identifies the frame as a HID report with the descriptor defining the type of HID report, in this case a mouse input. The data then contains the information from the human input. The X and Y values for the mouse can take values from -127 to 127.

3.1.2.4 Keyboard



Start (1 Byte)	Length (1 Byte)	Descriptor (1 Byte)	Data Length – one byte for the Descriptor							
0XFD	0X9	0X1	Modifier	0x00	Scan code 1	Scan code 2	Scan code 3	Scan code 4	Scan code 5	Scan code 6

Figure 3: The composition of a keyboard HID frame [6].

The keyboard HID report is formed as shown above. The descriptor indicates to the host system that the HID report is for a keyboard. The modifier is used to send the ALT, shift, GUI and CTRL keys. The scan codes that follow indicate what letter is being sent. The report produced allows multiple keys to be sent at once. However, if only one key is sent the rest of the scan codes are set to 0.

3.1.2.5 Bluetooth®

The Bluetooth® connection is made using a Bluetooth® compliant wireless module. The module connects to a host like a HID device. The HID protocol that was implemented over the Bluetooth® link was a lightweight version of the one used for USB™. This allowed transmission of the same HID reports used in the USB™ code to the Bluetooth® module.

3.1.3 RN42 Bluetooth® Module

The Bluetooth® module used in Superhans was the RN42 from Microchip. It is a fully certified Class 2 Bluetooth® device which is designed to allow wireless communication between devices. The small form factor and low power consumption of the device made it ideal for our application. Two different Bluetooth® connection types are supported; serial port profile (SPP) and HID.

The connection to the RN42 was made through serial UART. This connection allowed passing data over the Bluetooth® link, as well as access to the command mode on the RN42. The command mode allows configuration settings to be changed. In this application, the RN42 was configured as a ‘combi’ HID device, meaning that both keyboard and mouse commands could be sent over the connection. The auto reconnect option was also set to allow the RN42 to attempt to reconnect to the last connected device.

To send data over the Bluetooth® connect, a HID data frame was formed on the microcontroller before being sent over the serial connection to the Bluetooth® module. The Bluetooth® module reads the HID frame and phrases it for transmission over the Bluetooth® link. Although the format of the frame is identical to USB™ some of the character codes are different.

To use the Bluetooth® module with the FRDM board, a driver library was written that allowed access to all the features needed. The driver library made sending mouse and keyboard commands into a single function call and ensured the correct HID frame was formed for each input.

The driver also simplified the configuration of the RN42 by providing a set of functions that carried out the configuration. This allowed the RN42 to be easily switched between connection types, reset and initialising.

3.1.4 Use of Real Time Operating System (RTOS)

To allow the microcontroller to handle the input from the accelerometer and bend sensors, a real time operating system (RTOS) was used. A RTOS is an operating system that is designed to handle real time tasks such as reading sensor data. The RTOS used in this application was RTX [7], which is a deterministic RTOS designed for the Cortex™ processors. It gives low latency task switching, mutual exclusion operation and flexible scheduling. It also has a small system footprint which made it ideal for use on microcontrollers with limited resources.

The code for this application used two main tasks, one to read data from the accelerometer and the other to handle the input from the bend sensors. Each task also sends HID commands over the USB™ or Bluetooth® connection. To prevent access violations on the USB™ connection, a mutual exclusion operation (MUTEX) was set up. This allowed each task to 'lock' control of the USB™ connection while it sends data, before 'unlocking' it to allow the other task to use it. This was also used on the wireless version to protect the serial link to the Bluetooth® module. The use of the RTOS enabled the system to carry out more complex tasks as well as handle several inputs at once.

3.2 Glove Design (HF)

The glove used for the development phase of the project was a 661's MTX glove. This glove was deemed sturdy enough to handle the demands of hot-swapping components and extended between multiple developers/testers during the prototyping phase. The glove is rigid enough to mount. It should also be taken into consideration that the glove's rigidness is minimised to prevent restricting movement. Comfort was also factored to ensure the user has no difficulty with prolonged usage at any one time. In addition the glove has a simple assembly. The assembly enables developers to mount components to customize the glove to suit functionality requirements and house components. Thus making sure the glove is presentable and ensuring setup does not hamper with its functionality.

3.2.1 Flex Sensor Integration to the Glove

Initially there were many techniques discussed on how to incorporate the sensors to the glove. Appendix E refers to one of the first designs of integration. This was based on the idea of using a Velcro strap to hold the sensor in place, which was stitched along the finger. The sketch in Appendix E reveals how the sensors are going to be designed as well. But this will be elaborated in a later section. Later it was noticed that the sensor would not be kept fixed between the Velcro, which would make the sensor to slip out when the finger is flexed moving the knuckle forward. Another minor factor for not opting for the Velcro design is having adhesive straps on the glove's fingers. This increases the difficulty of replacing defective sensors. Unstrapping the Velcro might loosen its adhesive grip from the glove, requiring a replacement Velcro.

It was later decided that it would be better to have sensor pockets stitched along the fingers. This also helps the user replace the sensors with minimal hassle. If a light fabric is used no extra weight is added to the glove, which is needed to be at a minimum. The next task addresses an efficient method of stitching additional fabrics onto the glove. This improves finger's manoeuvrability. The best proposal unstitched the glove in the finger areas shown in Figure 4 [8]. Once this was done the glove was turned inside out, so when stitched and turn back out the stitches will be hidden under the fabric.

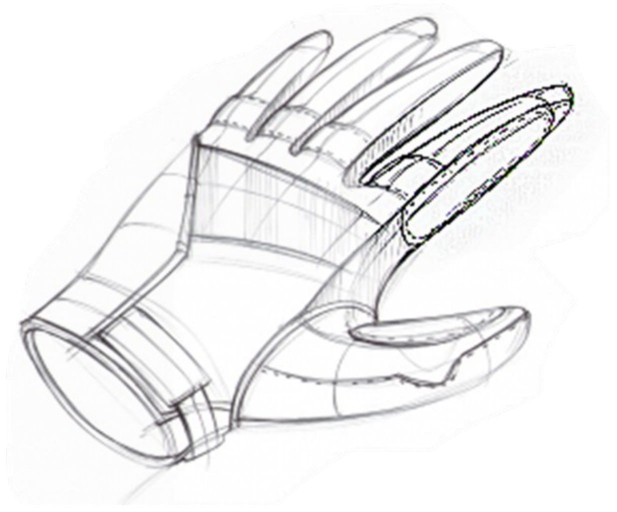


Figure 4: A sketch of the glove used, indicating the stich lines and different fabric composition of the glove

Once completed the fabric that corresponds to the dimensions of the finger area is stitched in between the Velcro. The fabric is then turned right side out concealing the stitch marks making the glove more presentable.

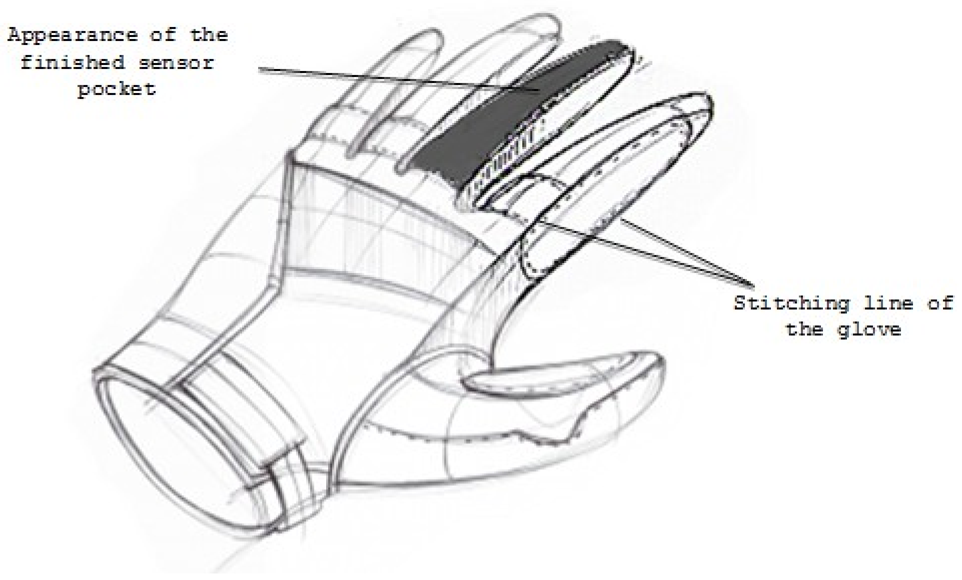

Appearance of the finished sensor pocket
Stitching line of the glove

Figure 5: How the sensor pockets would look once it was stitched in place [8].

The objective of the task is to prove the glove's concept. Three fingers were prioritized as shown in Figure 6.

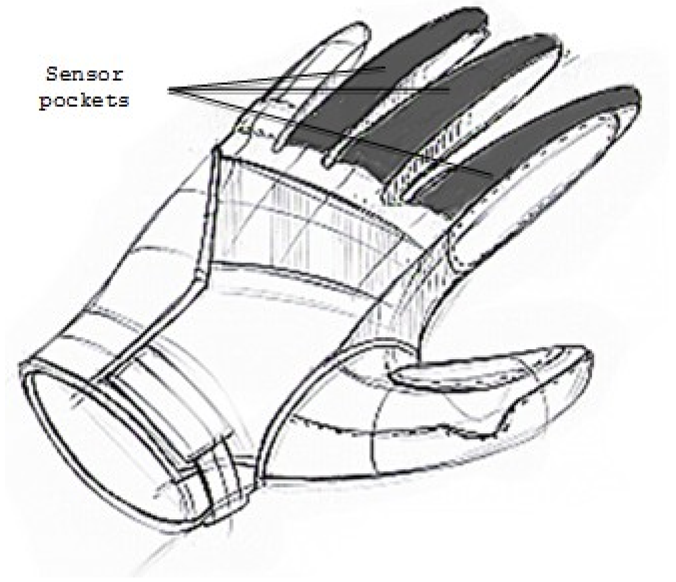


Figure 6: The sketch of the current design implemented on the glove, with three sensor pockets [8].

3.2.2 Hardware Integration

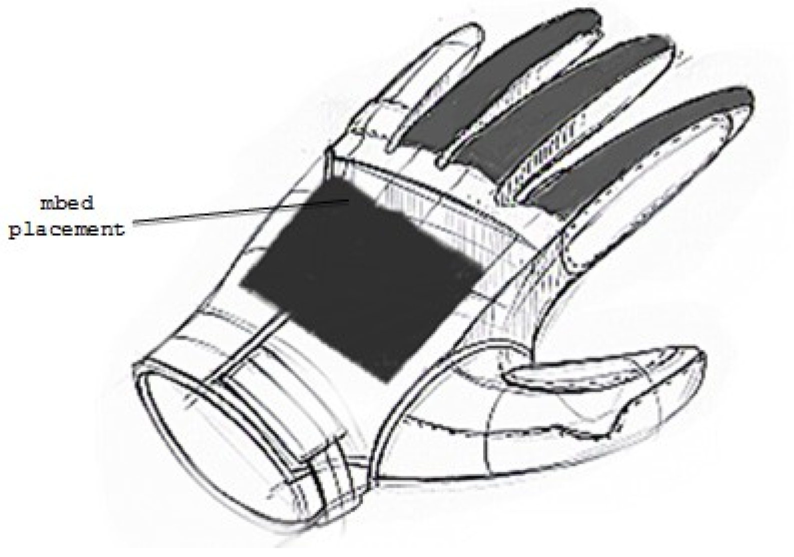


Figure 7: Sketch of the sensor and mbed placement on the glove [8].

Another part presented on the glove is an ARM® processor. It was decided an adhesive Velcro strap should be selected. Unlike the sensors, the processor should be fixed on the glove. One strap was glued to the bottom of the board and the other to the glove. This resulted in a sturdy grip between the board and the glove. Due to the small board dimensions, component mounting was simplified. This enhances comfort.

3.3 Sensor Design (HF)

Alternative flex sensors were designed to improve system sustainability. Flex sensors were designed to resemble the off the shelf counterparts.

3.3.1 What is a Flex Sensor?

A flex sensor is a type of sensor that changes resistance when bent. They measure finger flex as an electrical value. Resistance is proportional to the amount of finger flex applied.

3.3.2 Operation and Purpose of the Flex Sensor

There were do-it yourself (DIY) flex sensor designs available online but none of them gave promising outcomes. Once the concept of the DIY sensors was understood, a different approach was taken to achieve the same result. Starting from the same raw materials, the flex sensor design in Figure 8 is proposed based on experimental development trails.

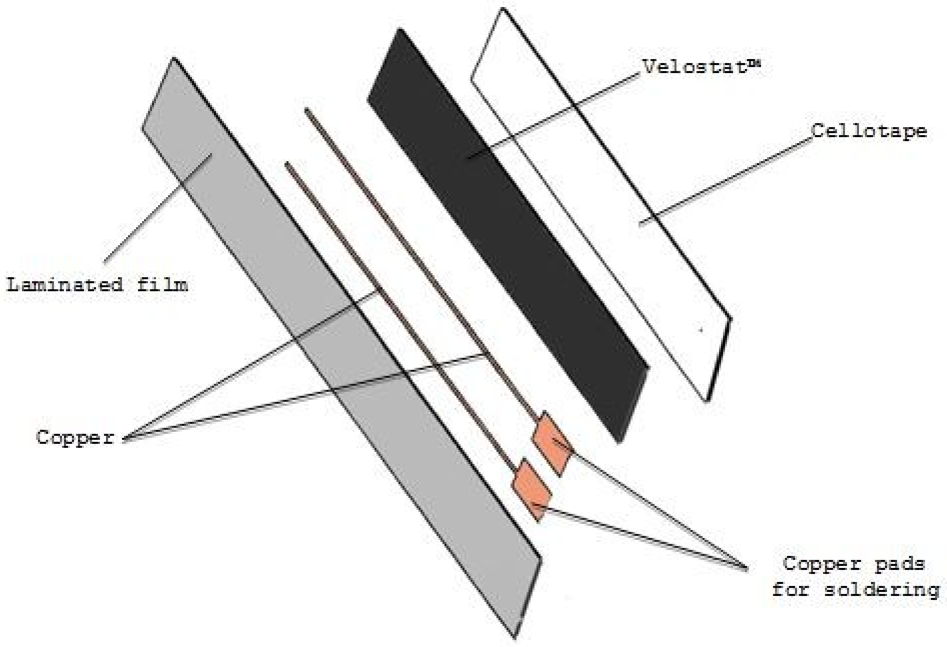
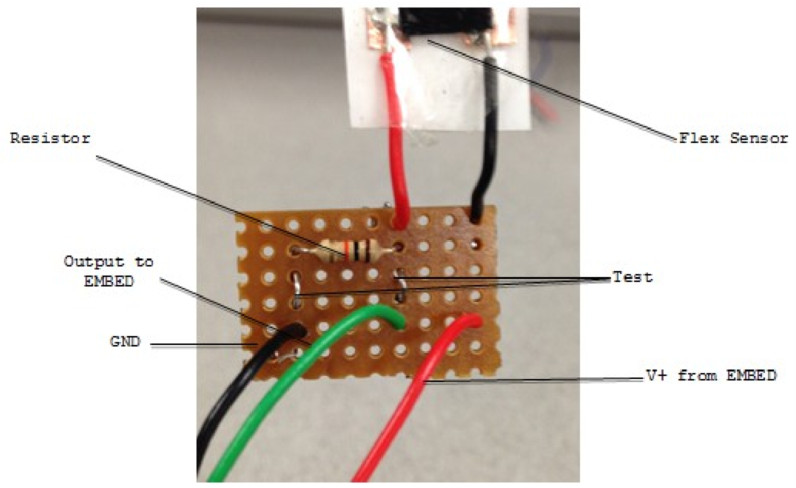
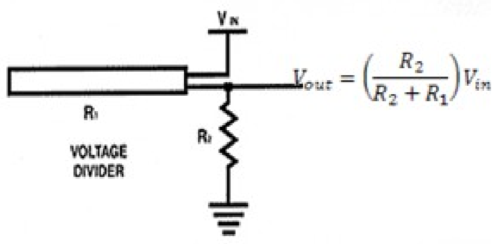


Figure 8: Composition and the materials used for the designing of the sensor.

The main component is a Velostat^{TM} bag, which is made from single layer of carbon-loaded polyethylene and its conductivity does not depend on humidity and acts as a conductive resistor. This is securely fastened between the copper strips. The Velostat conducts for an applied voltage. When bend or any degree of physical deformation is experienced the impedance of the Velostat^{TM} changes causing a change in voltage as explained below. This principle is applied to commercial flex sensors.

Flex sensors operate as analogue resistors. The flex sensors are configured into a potential divider network. The peak output voltage is proportional to the degree of bend applied. This is the technique used to obtain readings from the glove. There are complex circuits that also do more advanced functions by incorporating these sensors. But the main purpose was to acquire a voltage corresponding to the degree of bend and to keep external circuits to a minimum on the glove. So the circuit shown below did this.





[a] [b]

Figure 9: Photographic circuit diagram of the setup used to obtain the varying voltage relative to the resistance of the flex sensor.

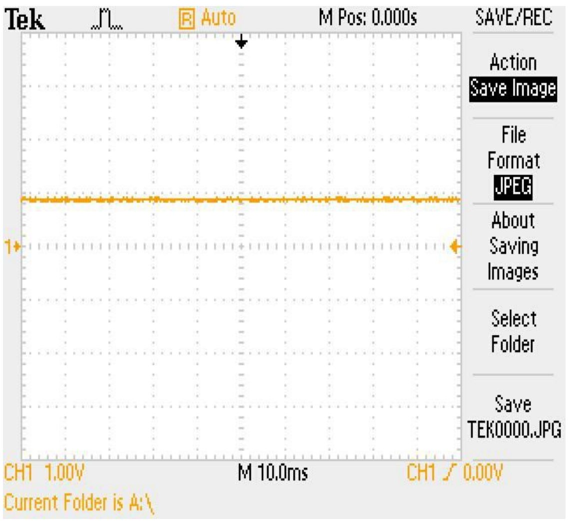
The sensor voltage will serve the purpose of a button/click while playing games. In future work this can be used to detect finger motions, which can be used to read sign language as mentioned in the design brief in Appendix A.

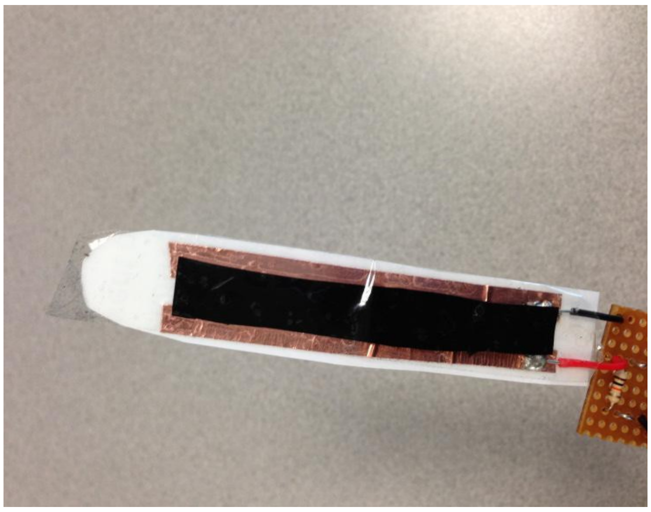
3.3.3 Determination of R2 Resistor

The resistor mentioned should be chosen carefully, so that the supplied voltage is equally split across the network. The impedance of the flex sensor should be measured at rest. When bent, the impedance of the flex sensor increases and hence the output voltage decreases. The voltage at half point (if equally split) will give a clear drop that would be beneficial for the software integration. In other words easily distinguishable between bent and rest states.

It should also be noted that the total network impedance may vary between sensors. Ideally the spacing between copper and Velostat^{TM} layers is constant. The objective of commercial sensors is to maintain these parameters. However, the objective of this task was to retrieve a voltage from a gesture. In addition sensor requirements vary between finger dimensions. Therefore priority for layer alignment was minimised. To compensate for alignment, the resistor R2 in Figure 9.a may vary between sensors.

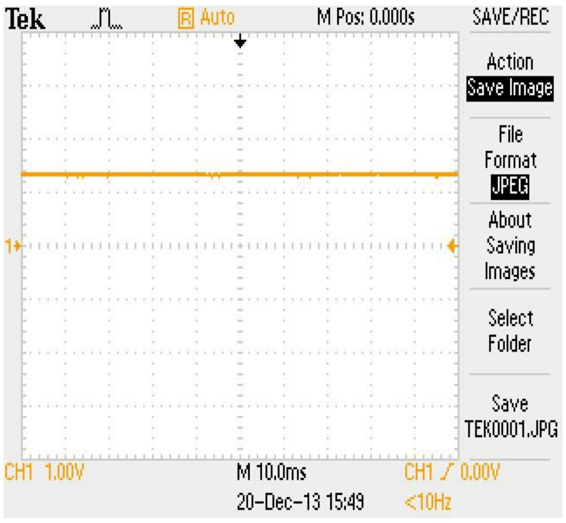
3.3.4 Readings of Flex Sensors

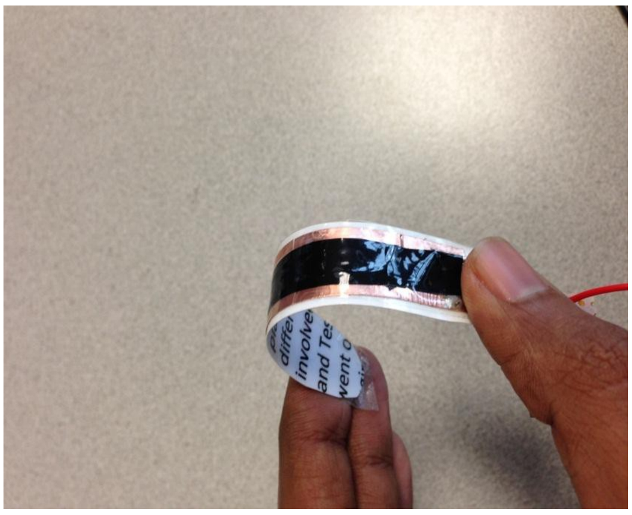




[a] [b]

Figure 10: shows the voltage reading when the sensor is not bending, which is 0.9mV when an input of ~3V is applied. 10.b shows the flex sensor at rest





[a] [b]

Figure 11: 11.a shows the reading when the sensor is bend to a degree shown in 11.b. This gives a voltage of 1.4V. This change is more than enough for the mbed to detect a change.

3.3.5 Key Presses (JB)

This section outlines the physical design of Superhans' buttons as well as discussing how the software handles key press.

Superhans features 16 separate touch buttons placed on the fingers of the glove. Figure 12 shows the final copper strip configuration used in the final prototype:

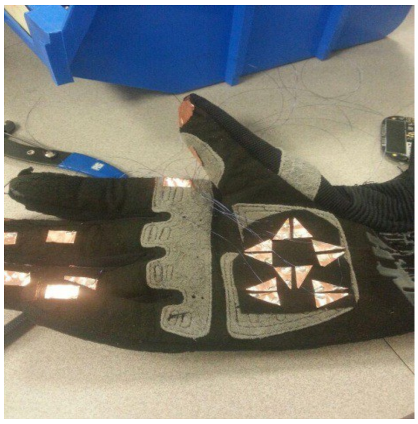


Figure 12:Right hand glove palm showing copper pad placement.

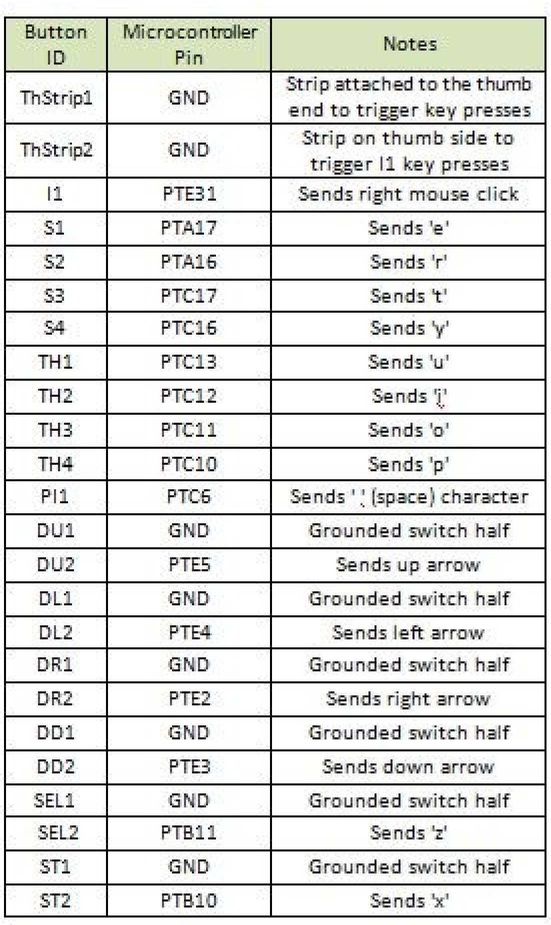
3.3.5.1 Design Choice

During the initial design process 3 design proposals were considered.

- Piezoelectric pressure sensitive pads: these would allow the user to touch the button with any part of their body to trigger a key press. It was determined that the cost of these devices (approximately £14 each with 16 units required by the target device) would make them unsuitable for the final prototype [9].
- Simple mechanical push buttons: a traditional approach would simply hardware integration but complicate component mounting. They would need to be attached with glue which has the risk of becoming undesirably loose during game play.
- A system of self-adhesive copper tape pads: this implementation would consist of attaching a strip of copper tape to the thumb of the glove and several strips placed on the fingers and palm. The adhesive backing of the copper tape used was found to be suitable for keeping the buttons in place during extended gameplay and transport of the device. This design was ultimately chosen.

3.3.5.2 Button Assembly and Wiring

A naming convention was introduced for software variables. Object members contain button, the American Standard Code for Information Interchange (ASCII) character they are assign to and their corresponding connection to the FRDM-KL25Z is shown in the Figure 13.



Button ID	Microcontroller Pin	Notes
ThStrip1	GND	Strip attached to the thumb end to trigger key presses
ThStrip2	GND	Strip on thumb side to trigger I1 key presses
I1	PTE31	Sends right mouse click
S1	PTA17	Sends 'e'
S2	PTA16	Sends 'r'
S3	PTC17	Sends 't'
S4	PTC16	Sends 'y'
TH1	PTC13	Sends 'u'
TH2	PTC12	Sends 'i'
TH3	PTC11	Sends 'o'
TH4	PTC10	Sends 'p'
PI1	PTC6	Sends ' ' (space) character
DU1	GND	Grounded switch half
DU2	PTE5	Sends up arrow
DL1	GND	Grounded switch half
DL2	PTE4	Sends left arrow
DR1	GND	Grounded switch half
DR2	PTE2	Sends right arrow
DD1	GND	Grounded switch half
DD2	PTE3	Sends down arrow
SEL1	GND	Grounded switch half
SEL2	PTB11	Sends 'z'
ST1	GND	Grounded switch half
ST2	PTB10	Sends 'x'

Figure 13: - table showing button connections to FRDM-KL25Z

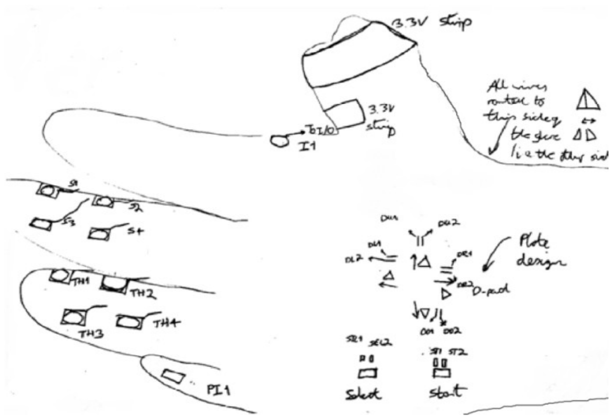


Figure 14: Preliminary naming conventions.

Superhans features two button variants. The rectangular copper pads on the fingers in Figure 12 measure 13mm x 7mm. DUn, DRn, DLn, DDn, SEL and ST in the table above are the split arrow shaped pads found on the palm. These are formed by cutting a standard rectangular pad into right angle triangles. One half of the pad is grounded and the other connected to the relevant digital input on the FRDM-KL25Z. To close the switch, both halves must be connected with a conductive material for a digital 0 to be sent to the corresponding freedom board pin. These buttons are assigned to the lesser used keys in a game e.g. keys corresponding to pause functions and special hotkey controls. A conductive material sheath can be placed on the thumb/index finger of the left hand to operate these buttons.

In the working prototype shown in Figure 15 and Figure 16 each strip is connected to a single track of veroboard on which DIL pins are placed. This is then connected to the FRDM-KL25Z via a crimped plug.

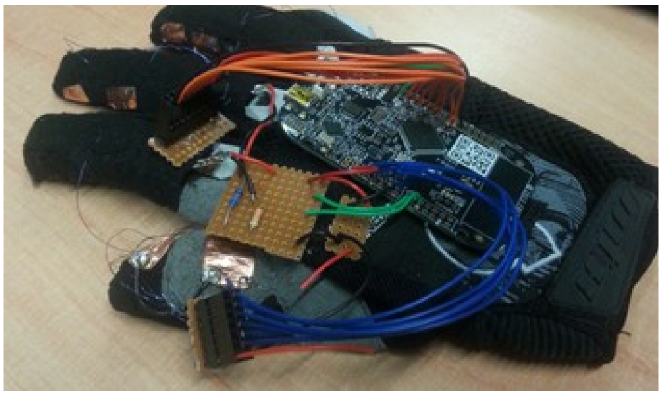


Figure 15: Working prototype wiring

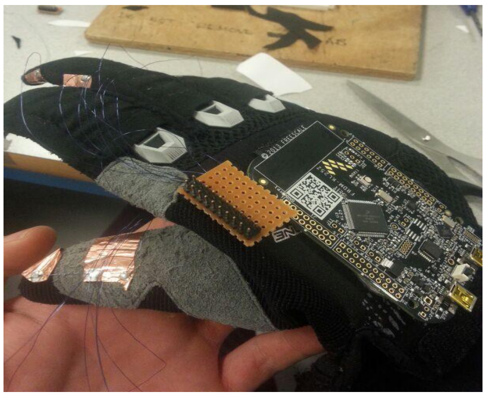


Figure 16: Prototype showing DIL pins

There are two copper strips attached to the thumb of the glove that measure 13mm x 40mm for the strip nearest the tip (ThStrip1 in Figure 15) and 13mm x 25mm for the strip used to make contact with pad I1 (ThStrip2 in Figure 16). Both of these strips are also connected to the GND pin of the FRDM-KL25Z.

The wires used to connect the copper pads to the header are secured using fabric loops to minimise accidental breakages.

3.3.5.3 FRDM-KL25Z Physical Connection and Key Press Software

All buttons except those directly connected to GND are connected to one of the digital general purpose input/out (GPIO) on the FRDM-KL25Z via an internal pull up resistor valued at 4.7kΩ [10]. They are therefore in active LOW configuration:

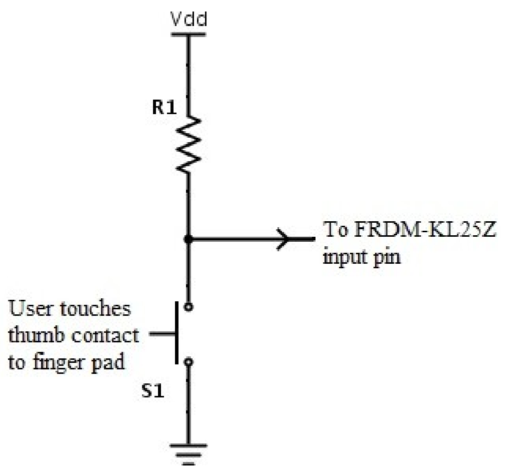


Figure 17: Internal pull-up resistor configuration. R1 = 4.7kΩ

The state of all buttons is monitored in the embedded software in the keyboard thread of the RTOS mechanism. This code block reads the state of the button and sends a HID report via the Bluetooth® or USB™ interface to correspond to a standard key press or key release. Initial code revisions simulated a repeated press of the relevant key every 17 milliseconds but during development it was discovered that the majority of games respond to a separate 'press' or 'release' command to initiate control; repeating the key press yielded unpredictable results.

The final code features a modified version of the keyCode C++ language function found in the USBMouseKeyboard mbed library. The original function does not allow multiple keys to be pressed at once. The modified function features the concept of a key buffer to allow up to two buttons to be pressed at once, utilising the HID report introduced in the Hardware Architecture section of this document. This allows up to three controls to be pressed and released with predictable results; up two keyboard key presses and one mouse click. Following user research this was deemed satisfactory for the majority of gaming titles. The mouse clicks are sent via a completely separate HID report and so have no influence over the characters that are sent to the host. A representation of the algorithm is show below in Figure 18. The full code keyboard thread can be found in Appendix D.

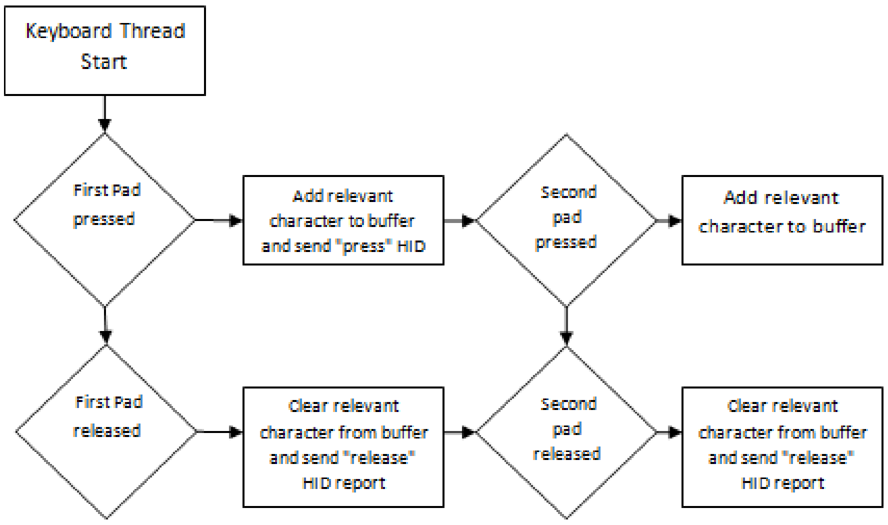
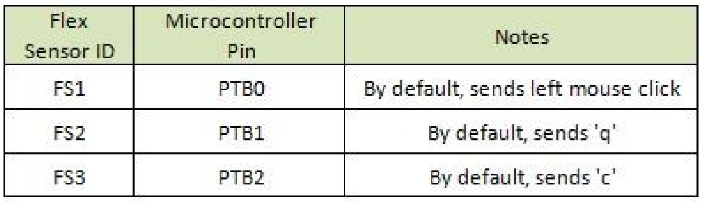

Keyboard Thread Start
First Pad pressed
Add relevant character to buffer and send "press" HID
Second pad pressed
Add relevant character to buffer
First Pad released
Clear relevant character from buffer and send "release" HID report
Second pad released
Clear relevant character from buffer and send "release" HID report

Figure 18: - Key press Flow Diagram

3.3.5.4 Flex Sensor Interaction

Depending on the type of game played, the three flex sensor/potential divider circuits on the device can be used to trigger either key presses or mouse clicks. Flex measurements are fed into 3 of the separate 16-bit ADCs available on the FRDM-KL25Z. The digital value is compared with a threshold voltage. If the corresponding glove finger surpasses the threshold, a key event is invoked on the host device. When the glove finger straightens and the corresponding digital value falls below the threshold a release command is sent. The flex sensor input follows the same flow as shown in Figure 18.

The default commands sent by the final prototype are listed below:



Flex Sensor ID	Microcontroller Pin	Notes
FS1	PTB0	By default, sends left mouse click
FS2	PTB1	By default, sends 'q'
FS3	PTB2	By default, sends 'c'

Figure 19: Default flex sensor

The button interface could be vastly improved if a custom HID profile was designed for the controller. This would remove the need for the key buffer construct since the HID report would have room for 16 different button presses. The keyboard HID profile used presently limits the button pressed down to 6 and so each button cannot be dealt with via separate HID channel. In the final prototype, if the flex sensors are mapped to key presses they can be configured to use fixed HID data channels which allows up to four keyboard keys to be pressed down at once but a custom HID profile of type game controller would give the code increased modularity and elegance.

The drawback of the custom game controller HID report is that it may not be recognised by devices like the Raspberry Pi and Arduino; the plug and play library features may not be present.

All in all the current solution represents the best compatibility of the device.

3.4 Haptic Feedback (JB)

3.4.1 Overview

Haptic Feedback is a widely researched field that covers many diverse areas of engineering. Haptic feedback enhances Superhans by adding a layer of realism that allows the user to interact with augmented sensation.

Superhans features 4 direct current (DC) vibration motors. The vibration motor chosen is a small self-contained disc type found in many modern applications including mobile phones, as shown in figure 20. The motor's aim is to provide simple haptic feedback to the user:

- One on each of the fingers that feature a flex sensor to alert the user if commands have been sent when a finger is bent.
- One motor on the palm of the device that acts as a training feature. In the absence of a functional calibration suite this motor, deactivated by default and activated by a push button toggle switch, vibrates once whenever the user reaches the rest position when no mouse event is triggered. This aims to bypass the need for a calibration function by constantly reinforcing the rest position needed for predictable control.

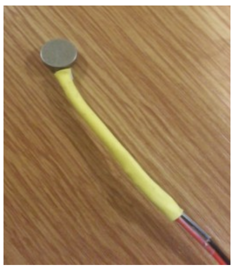
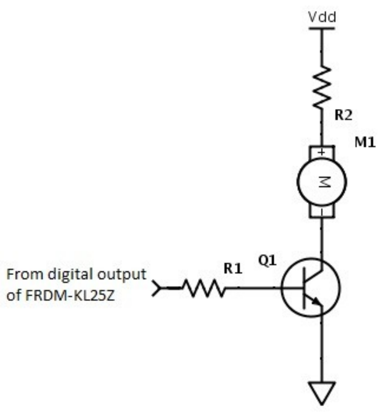


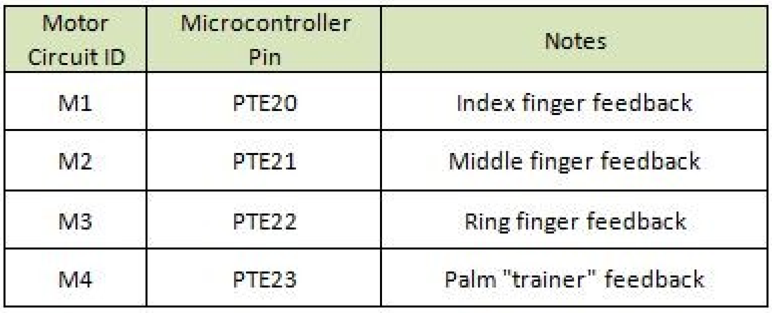
Figure 20: Disc vibration motor. The inline resistor is hidden under heat shrink tubing

Through experimental research it was found that the average resistance of each vibration motor was ~80Ω. Furthermore it was determined that only a subtle vibration was required to provide adequate haptic feedback. A full 3.3V input on the motors (corresponding to a digital HIGH from the FRDM-KL25Z) was deemed too strong and so a 39Ω resistor was to provide it with ~2V.

In practice it was found that the maximum current draw of each motor was around 200mA. The maximum current draw available from each of the digital outputs of the FRDM-KL25Z is ~80mA. These findings necessitated the simple bipolar junction transistor circuit featured overleaf in figure 21. The vibration motor is denoted by M1.



The FRDM-KL25Z connections are shown in the table below:



Motor Circuit ID	Microcontroller Pin	Notes
M1	PTE20	Index finger feedback
M2	PTE21	Middle finger feedback
M3	PTE22	Ring finger feedback
M4	PTE23	Palm "trainer" feedback

Figure 21: Table of haptic motor connections

3.4.2 What Could be Done to Improve Haptic Feedback?

Haptic feedback could also be used for error handling. For example, if a malfunction occurs in one of the flex sensors then the vibration motor corresponding to that flex sensor could vibrate once every half a second to alert the user of the fault. Other types of actuators could be used to simulate a constant force to simulate the feeling of touch on each finger.

3.5 Gesture Recognition (AD)

Gesture recognition has recently gained attention in the field of human-computer interaction because of its speed and intuitive nature to the user. It is a complex subject and may involve aspects such as motion modelling, pattern recognition and machine learning. Majority of the previous work is based on computer vision techniques that involve aspects of motion modelling, pattern recognition and machine learning. For an example, the Wii remote has an IR light sensor inside the remote and detects motion by tracking the relative movement of the infra red (IR) transmitter mounted on the display. These techniques are usually hindered by their hardware requirements (camera and transmitter), difficulty in hand localization and complexity of hand gesture modelling. These problems are addressed in Superhans, which uses an on-board accelerometer for real time gesture recognition. Recent work such as Artificial Neural Networks (ANN) [11] and uWave [12] demonstrated gesture recognition using accelerometer data. The presented gesture recognition algorithms although efficient, require acceleration quantization; dynamic time warping (DTW) (matching two time series) and gesture libraries. These gesture libraries comprise of time series of acceleration data and the input series is matched with templates. A typical algorithm is shown in Figure 23.

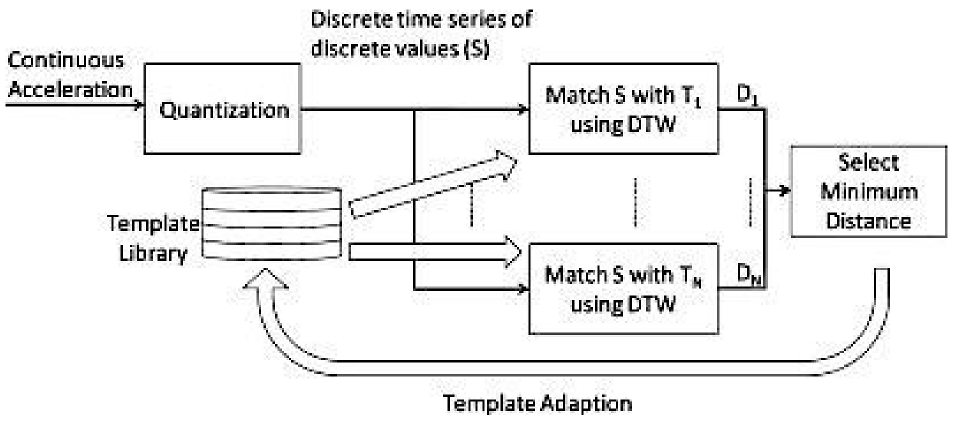


Figure 22: A typical algorithms based on acceleration quantization, template matching with DTW and template adaptation [12]

These complex algorithms are computationally expensive and can take up to 300ms on a 16-bit microcontroller in recognizing a gesture from an eight-gesture vocabulary[12]. This latency can result in undesirable gaming experience and therefore a real-time gesture recognition using static acceleration was proposed.

Static and Dynamic Acceleration

An accelerometer is an electromechanical device that measures acceleration forces, which can be static caused by constant forces like gravitation or dynamic caused by time varying movements. The initial plan with Superhans was to analyse arm movement, which would have required processing dynamic acceleration data using complex algorithms and hence latency as described earlier. To address this issue the idea of using hand tilt movements was proposed and is described.

3.5.1 Static Acceleration Data Analysis

3.5.1.1 Rapid prototype and Python graphical accelerometer data logger

A simple application was made using the Python Matplotlib API to plot the raw acceleration data in real-time [13]. Three axis accelerometer data was sent via a PC serial interface. Before the development of Superhans began, a rapid prototype was built using Arduino Uno equipped with an accelerometer. The Arduino was programmed to send raw accelerometer data over serial port. This rapid prototype was used to test the Python application and accelerated the development process.

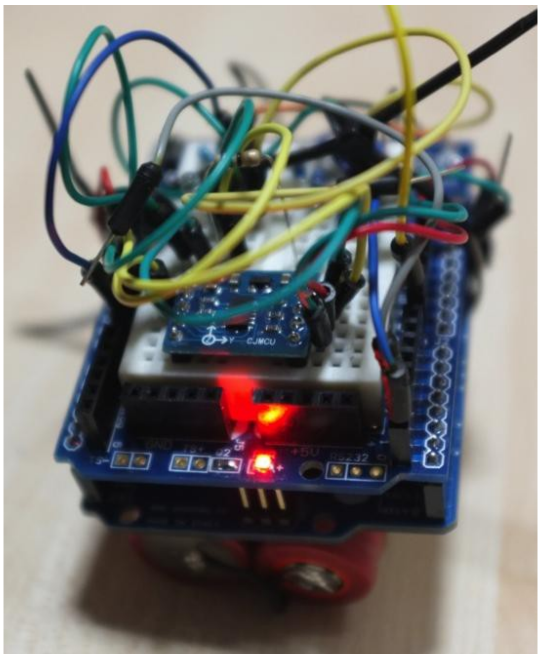


Figure 23: Prototype for accelerometer data analysis

3.5.1.2 Accelerometer Calibration for Superhans

Superhans is built on the Freescale FRDM-KL25Z Freedom Development Platform which is equipped with an Xtrinsic MMA8451Q three-axis, 14-bit low power Digital Accelerometer shown Figure 25 [13].

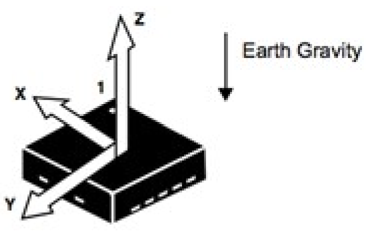


Figure 24: MMA8451Q and directions of the detectable accelerations [13]

Superhans raw acceleration data was monitored at rest and a high frequency noise of ±0.1g was seen as shown in Figure 26. This noise factor was reduced by carefully setting threshold values to avoid any errors.

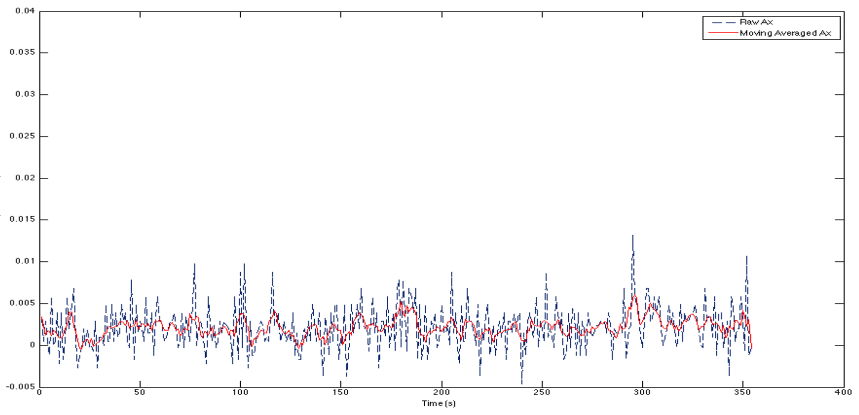


Figure 25: High frequency noise from accelerometer at rest

The raw data harvested from the accelerometer was categorised into the following gestures:

Vertical wrist movements : Radial and ulnar deviation as shown in Figure 28 was set to trigger up and down movement respectively.

Pronation and supination as shown in Figure 29 were identified as left and right movement respectively. These hand orientations were ideal because of the naturalness of movement.

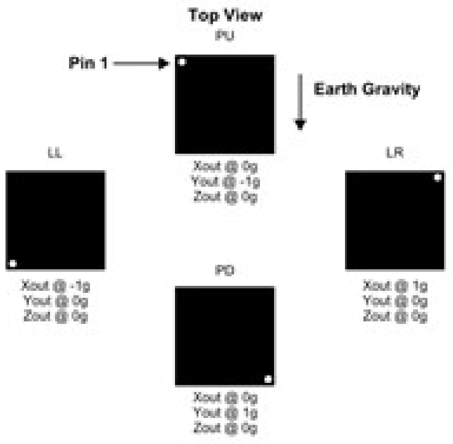


Figure 26: High frequency noise from accelerometer at rest

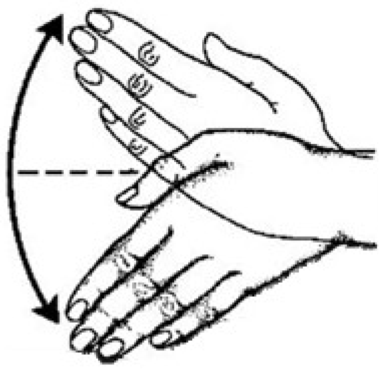


Figure 27: [a] Hand ulnar - down and [b] radial -up movement [14]

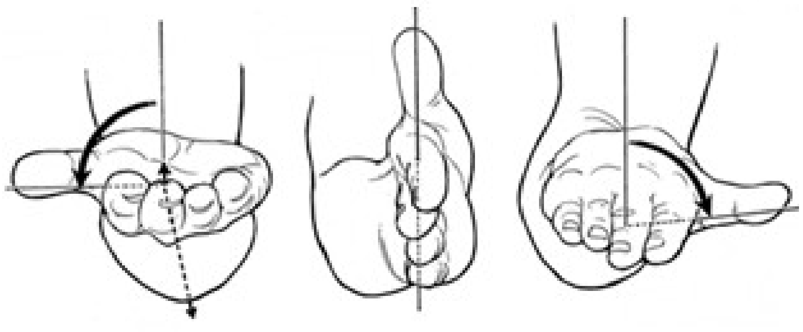


Figure 28: [a] Hand Supination - right, [b] rest position and [c] pronation - left movement [14]

The direction for acceleration was chosen depending on the placement of the freedom board on the hand and was used to calibrate the sensor to output left or right logics.

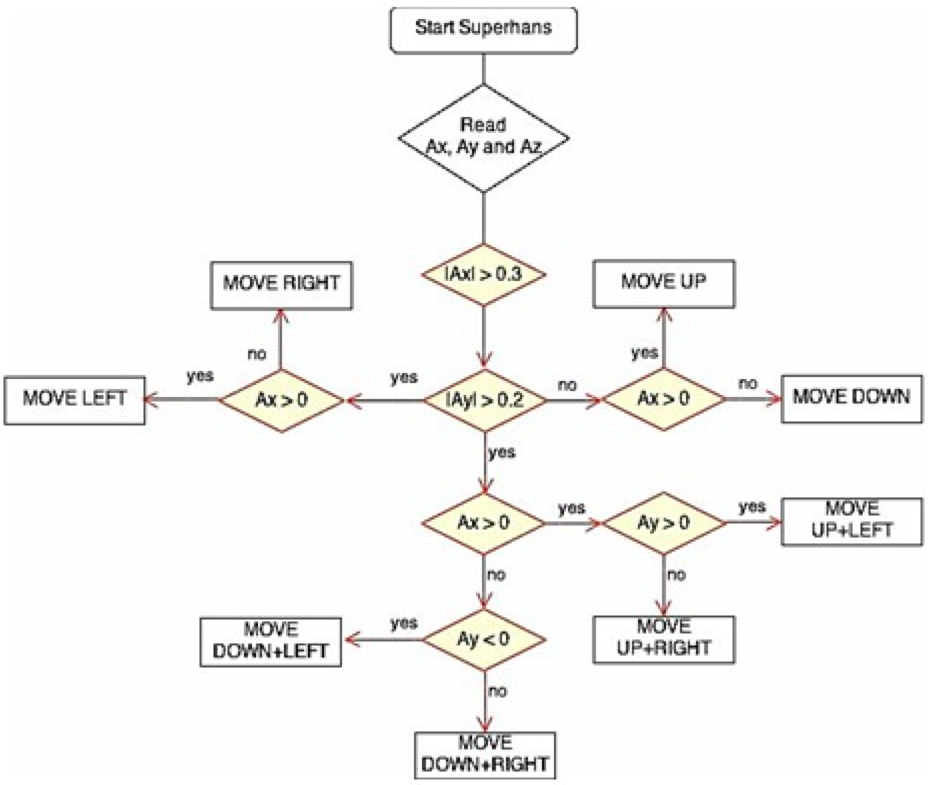


Figure 29: Algorithm for hand orientation using static acceleration data

Below is an example showing the x direction acceleration data corresponding to three consecutive hand pronation movements. The threshold was set to 0.4g and the corresponding left logic can be seen in Figure 31.

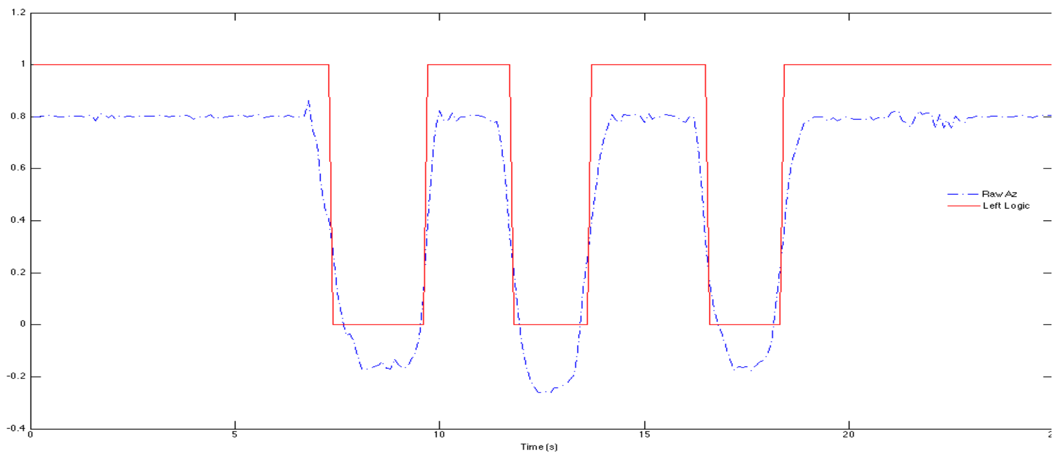
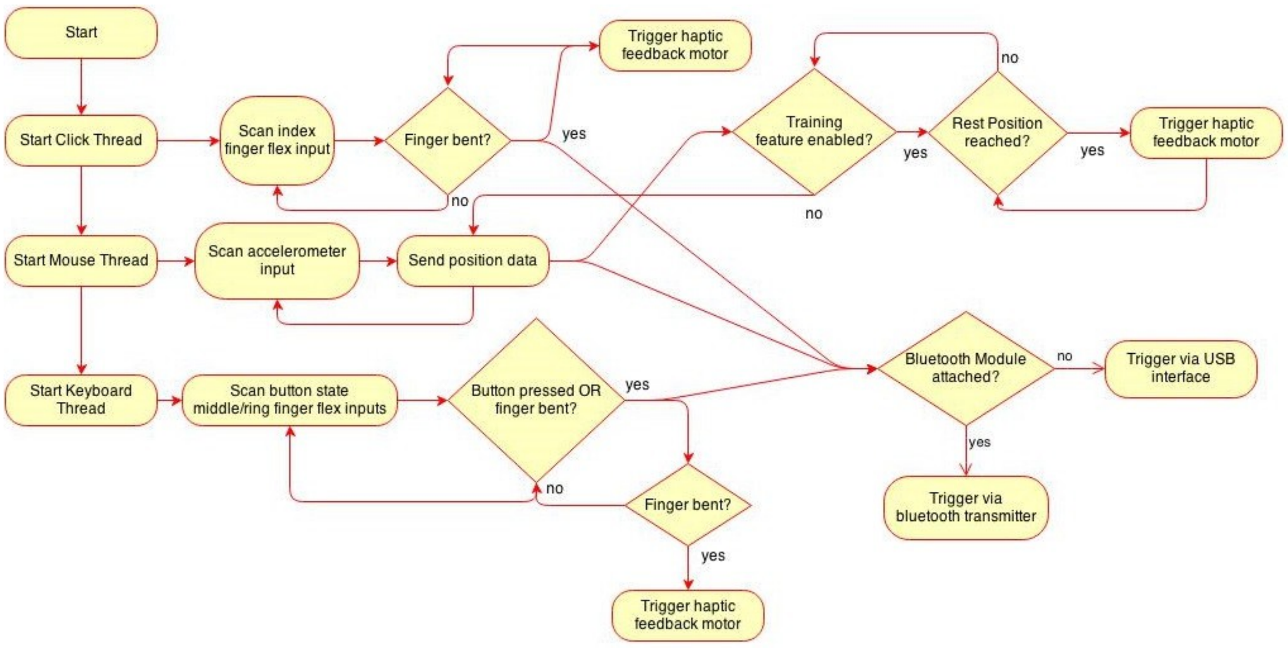


Figure 30: Raw acceleration data in X direction and logic representing three consecutive left tilts

3.6 Embedded Software Review (JB)

3.6.1 Overview

Below is a flow diagram representation of the final embedded code loaded onto the FRDM-KL25Z. The final code is an all in one solution featuring a pre-processor directive that must be properly defined before compilation to choose which transmission protocol Superhans uses either USB™ or Bluetooth®.



Start
Start Click Thread
Scan index finger flex input
Finger bent?
yes
no
Trigger haptic feedback motor
Training feature enabled?
yes
no
Rest Position reached?
no
yes
Trigger haptic feedback motor
Start Mouse Thread
Scan accelerometer input
Send position data
Start Keyboard Thread
Scan button state middle/ring finger flex inputs
Button pressed OR finger bent?
yes
no
Finger bent?
yes
Trigger haptic feedback motor
Bluetooth Module attached?
no
yes
Trigger via USB interface
Trigger via bluetooth transmitter

Figure 31: Embedded Code flow diagram

Following the execution path from "start" in the top left hand code of figure 32, we see that the three *RTOS* threads begin immediately, running concurrently from this point on. The three forms of sensor are then continuously scanned for input. If valid input is read, e.g. a finger is bend, a touch pad is pressed or the hand is tilted the data is sent to the host via either the Bluetooth or USB interface.

3.6.2 mbed Libraries Utilised

The full code for the wired version of the glove can be found in appendix D. Please note this the example code does not feature the haptic feedback displayed above or the invert toggle switch control mentioned in section 4.3.1. The code inherits functions from the open source mbed libraries listed below:

- *mbed* - Allows access to all functions to set up the GPIO of the FRDM-KL25Z [15]
- *RTOS* - Contains all code necessary for running the RTOS, used to simulate multithreading [16]
- *MMA8451Q* - Allows access to the accelerometer on board the FRDM-KL25Z [17]
- *USBMouseKeyboard* - Contains all functions that allow the transmission of mouse and key press data via the USB™ or Bluetooth® bus [18]
- *RN42* - Contains all functions concerning the Microchip RN 42 Bluetooth® module implemented in the wireless version of Superhans. This library is written by (TH) and can be found in appendix [letter

3.7 Graphical User Interface (LT)

The GUI allows a user to interact with product functionality using graphical icons Superhans does not require the GUI to be installed before operating. The GUI is designed to assist the user with the initial connection. The GUI is compatible on OS X and Windows devices.

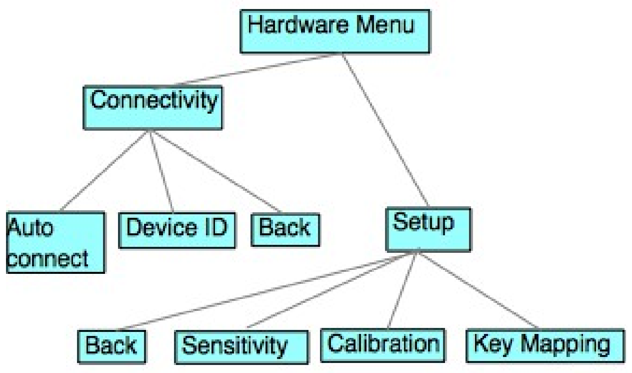
Superhans GUI is written with the C-sharp object-oriented programming language. C-sharp's .NET libraries have been selected for their productivity toolset. C-sharp is predominantly compatible with Windows operating systems. A cross-compiler was required to enable .NET applications to operate on Unix-based operating systems. Xamarin, an open-source Unix cross-compiler was utilized for this functionality [19].

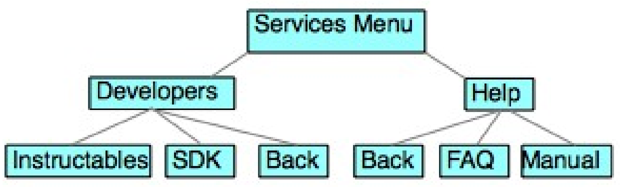
3.7.1 GUI Objectives

The GUI should provide a visual representation of adjustable system parameters. The GUI offers an interface for interacting with functionalities and product services independent of the internal program architecture. From the hardware architect's perspective, there are two major operational functionalities: Wireless connectivity and device calibration. Customer support is decomposed into product support, documentation and online communities. All GUI services are presented on a main menu layout on a Windows Form.

3.7.2 Menu Architecture

For visual lucidity, the main menu is constructed into two menu hierarchies: Hardware and Services Menu as illustrated in Figure 33. The Windows Form displays hardware and customer service menu options in parallel. This enables the user to navigate within both sub-menus simultaneously, with each hierarchy working independently of each other. When the user selects a menu icon, only the new layout related to the icon's sub-menus is rendered..





[a] [b]

Figure 32: Parallel architecture of main menus on the GUI Windows Form: [a] Hardware menu hierarchy and [b] Service menu hierarchy

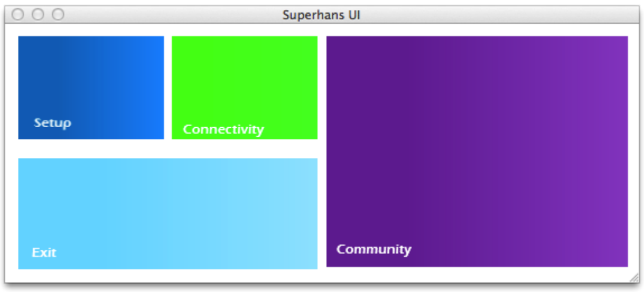
The main window form is constructed using two parallel dynamic trees displayed in Figure 34. Figure 34.a represents the hardware architecture parameters and 34.b represents user and online services. Parent nodes dictate the menu navigation using icons. Menus contain a list of sub-menus to render when selected. Sub-menus are represented as the parent's child nodes in Figure 34. Each child node holds a texture map and an icon as object member. When the child node is requested for rendering, the texture is read and rendered accordingly. The child-parent node link allows any sub-menu to retrieve and render the parent menu irrespective of the child's tree location. The back child node decrements the tree iterator. The leaf nodes execute hardware parameter or customer services tasks.

The two menu hierarchies are further decomposed into sub-menus containing options for controlling the system parameters and requirements. Dynamic trees are selected to provide bidirectional menu navigation between child and parent menus. In addition, there is no limitation to the number of possible child nodes. Two trees provide menu modularity. These algorithms have a run-time computation of O (log n). The dynamic tree is constructed of parent and child nodes during compilation.

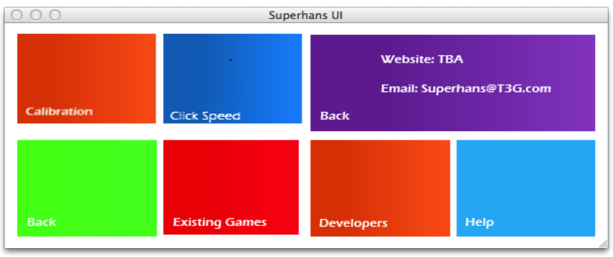
An iterator is assigned to each tree. The iterator returns the pointer or child node data. When the iterator's status changes, the texture contents of the child node are obtained and rendered. Rendering includes navigating to the directory containing the texture's image file, loading the image onto an icon object and constructing any classes that contain operations for the current node.

3.7.3 Rendering

Textures are created as .png image files using Microsoft Office. Textures are contained in a directory within the GUI project folder. A directory hierarchy identical to the dynamic tree architecture contains node textures. Each menu's rendering order is contained in a text file. The text file contains a list of reference keys associated with the icon ID requesting the texture and the location and name of the texture binding.



[a]



[b]

Figure 33: Visual representation of the main Window Form: [a] Main form menu displaying the root nodes of both hierarchies and [b] Main form displaying a tree depth of two

The leftmost tiles on 34.a contain the hardware parameter menu. The community tile represents the customer service menu. The remaining three icons for the services sub-group are initially hidden. The parent nodes contain child icon's show/hide properties. When clicked, the menu resources through tree levels. 35.b shows a new menu rendered for a tree depth of two.

Icons are stored as .NET PictureBox objects. There are eight PictureBox objects contained on the main form with an equal x-axis and y-axis separation between objects. Texture images are created with size protocol to relieve the rendering class of unnecessary computation. The menu nodes contain information regarding icon visibility, location of the texture images and the rendering order.

When the tree iterator undergoes a unary operation, the parent node is passed as an input argument to a texture class' constructor. The texture class navigates to the current node's texture directory and reads the menu's text file. The texture is located and loaded into C-sharp. The texture is binded into a bitmap object for rendering. Once bonded, the icon is rendered. This is repeated for each child.

Menu icons undertake animation during rendering to enhance the GUI's appearance. The animation class contains a range of animation styles. A PictureBox contains sizing and positioning information. Translation and rotation vectors describe the motion of PictureBoxes. These vectors are manipulated to simulate animation.

Typically animations occur at the monitor's refresh rate. For this reason, all animation vectors have a sampling interval of 17ms corresponding to 60 frames per second (FPS).

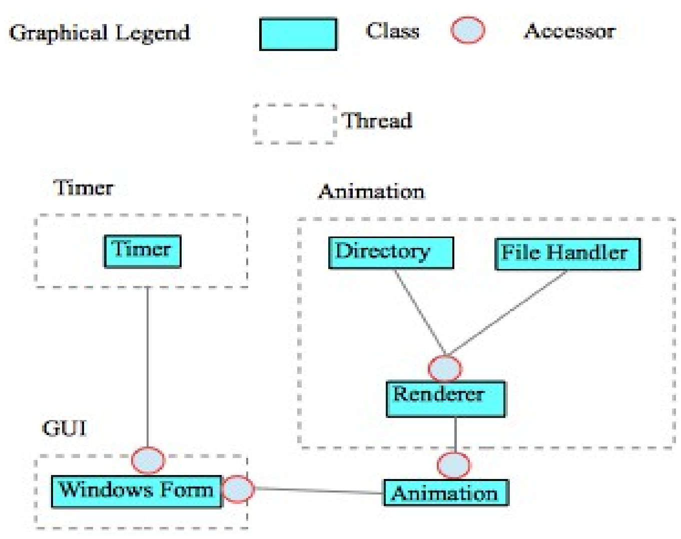


Figure 34: GUI class architecture

Classes are grouped into parallel threads in Figure 35 to relieve the Windows Form from memory intensive tasks. Classes access objects within the Windows Form class using function pointers (delegates). The main form is handled independently of the sub-classes. The file handler class is responsible for directory navigation and file reading/outputting using .NET streams. The texture class handles the image container and texture binding. The animation class manipulates the position and size of the objects on the form class using a reference key

A timer class logs the animation time. An event handler is a function call triggered by a system or key event. An event handler is instantiated and linked to a timer class in the Windows Form. The event is called when the program's execution time increases by 17ms interval. Each event call increments the animation timer.

The animation class encapsulates object translation and animation operations. This class inherits the texture mapping class. When the tree iterator changes its state, an instance of the animation class is instantiated. The class handles texture reading, binding, rendering and manipulates PictureBox dimension and location properties for animation purposes. The direction of rotation during menu rendering is dictated by a random number generator. Market research suggest an optimum animation time of 2 seconds for menus and 0.5 seconds for external forms.

3.7.4 Multi-Threading

Procedural GUI designs suffer from latency. Memory intensive operations can freeze the GUI. During this state key events are no longer processed. As a result, the user may misinterpret the state and force quit the application.

The multi-threading architecture as illustrated in Figure 35, handles memory intensive operations in parallel to relieve the Windows Form from intensive tasks. File handling, directory navigation, rendering and timer are handled on external threads. Threads are automatically instantiated in memory by C Sharp. During instantiation, the threads are explicitly linked to class operations. To completely isolate data and thread operations, shared data is accessed by invoking function pointers. This removes direct access between threads.

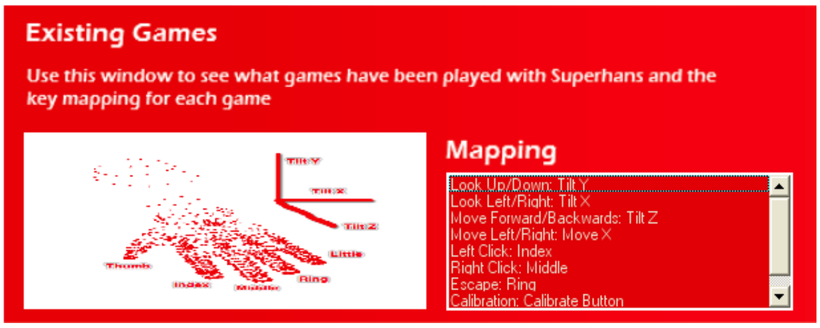
3.7.5 Functionality Operations

Leaf nodes in the dynamic tree architecture are responsible for executing user requested operations. These operations include: handling of external programs such as .doc or web documents, toggling hardware states such as automatic Bluetooth® connection and opening other Windows Forms containing more operations. A list for hardware and customer service operations is listed below respectively.

Auto-Connect:	Toggle automatic Bluetooth® detection
Device ID:	Display device Bluetooth® ID
Key Mapping:	Access device key mapping database
Sensitivity:	Adjust mouse movement sensitivity
Calibration:	Change device reference position
PDF:	Open product manual using Adobe
FAQ:	Open help document using Adobe
SDK:	Web link to the source code Github
Instructables:	Web link to the hardware instructions

Some hardware functionality requires multiple system parameters to be modified individually. These parameters are contained on an external Windows Form. These forms manipulate the device performance characteristics such as movement sensitivity and calibration. The services menu provides access to private development web pages, online support and product documentation. Services are opened with .NET process objects via OS system panels. The external application automatically links the user to online material through their default web browsers or opens manual/help documentation using Adobe.

A process is a .NET object that executes and monitors an external program's executable. The GUI handles three types of application-related operations. Document readers such as Adobe and Microsoft Office automatically open help files related to Superhans products. The .NET process object automatically links users to online services via web browser. Mouse/Keyboard system panels provide access to key and mouse event sensitivity. This process is platform dependent. Windows stores keyboard settings as an executable. OS X has a panel shortcut to a system panel contained in the system path.



Existing Games

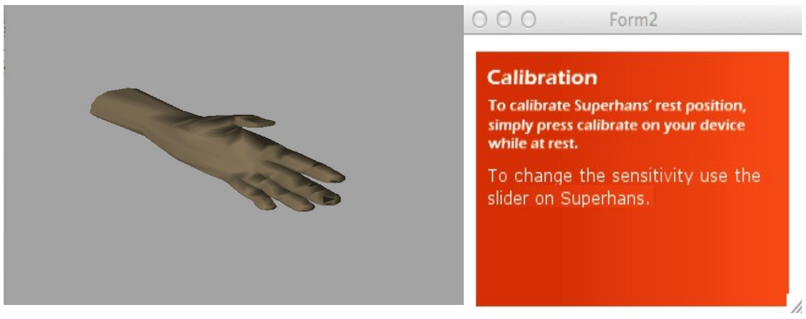
Use this window to see what games have been played with Superhans and the key mapping for each game

Mapping

Look Up/Down: Tilt Y
Look Left/Right: Tilt X
Move Forward/Backwards: Tilt Z
Move Left/Right: Move X
Left Click: Index
Right Click: Middle
Escape: Ring
Calibration: Calibrate Button

Figure 35: External Windows Form containing options for mapping hand gestures to games.

The Windows Form in Figure 37 displays options for mapping hand gestures to key events. An interactive list initially displays the default key presses for games with a graphical representation of gesture names. Key event handlers are implemented in C-sharp to enable list item manipulation through key presses.



Calibration

To calibrate Superhans' rest position, simply press calibrate on your device while at rest.

To change the sensitivity use the slider on Superhans.

The calibration form provides the user with calibration instructions. Calibration influences the sensitivity of accelerometer and trigger readings. A real-time interactive hand model created in open graphic library (OpenGL) with C++ moves with the connected device. This provides a real-time visual representation of device calibration.

3.7.6 OpenGL Rendering

An open-source blender model is used to generate a 3D model mesh of a human hand. The 3D model is exported with a *.obj* file extension. An object loader has been written to read the contents of the 3D model and generate an OpenGL mesh. The object loader locates and opens the *.obj* file. Each object vertex is stored as a 3D vector. Colour and material properties are stored as normalized vectors. The object loader reads vector information into a frame buffer. The program then tessellates the point cloud into triangles to enhance lighting and material property influence. Tessellated object faces are contained in a vertex buffer for rendering. The model depth is culled based on the pinpoint camera position. To improve rendering speed, the frame buffer only renders the closest polygons to the camera.

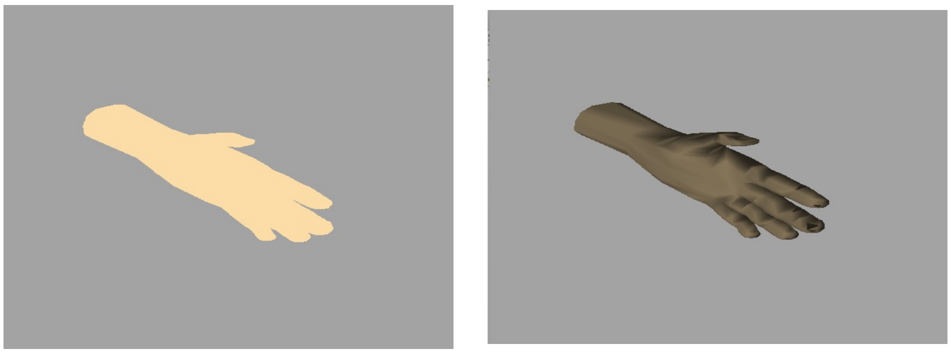
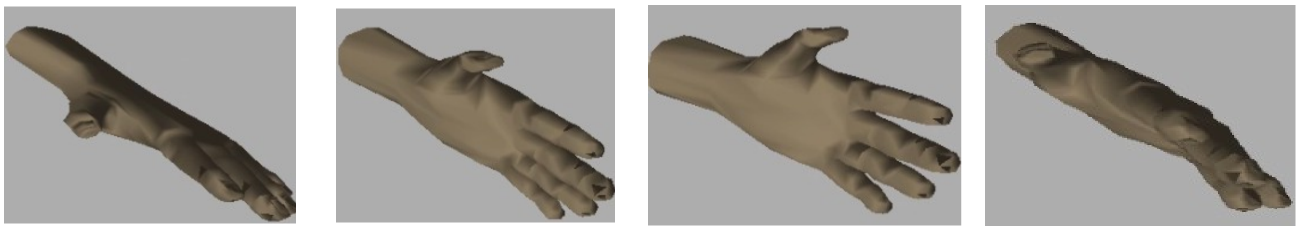

[a] [b]

Figure 36: Influence of lighting and shading models on the appearance of 3D models. [a] 3D model with lighting disabled [b] 3D model with flat shading disabled

For each polygon face, the Modelview matrix is translated to polygon vertices. After translation, vertices are rendered as points. A convex is created by interpolating the parameter space between vertices. Light classes simulate the effect of specular, incident and ambient illumination on 3D models. The reflection properties of lighting on objects are dictated by the material properties. A flat shading model is implemented. Flat shading computes the colour information of polygons based on the colour of the polygon and the consequent lighting colour at each vector. The colour of the face is interpolated across the polygon between vertices before rendering.



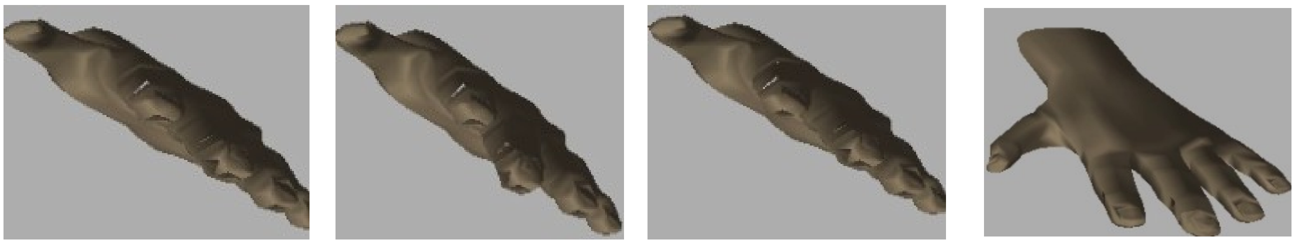


Figure 37: 3D model undertaking real-time interaction

When a key event is triggered, OpenGL uses call back function pointers to catch the integer representation of the ASCII keys triggered by gestures. The model matrix containing the model undergoes an affine transformation to translate and rotate the 3D model.

4 Design Principles

4.1 Innovation (LT)

The multimedia industry is a successful participant in consumer electronics. Gaming has been a growing sector within this industry. Popularity within this sector has significantly risen over the last decade in comparison to other entertainment sectors.

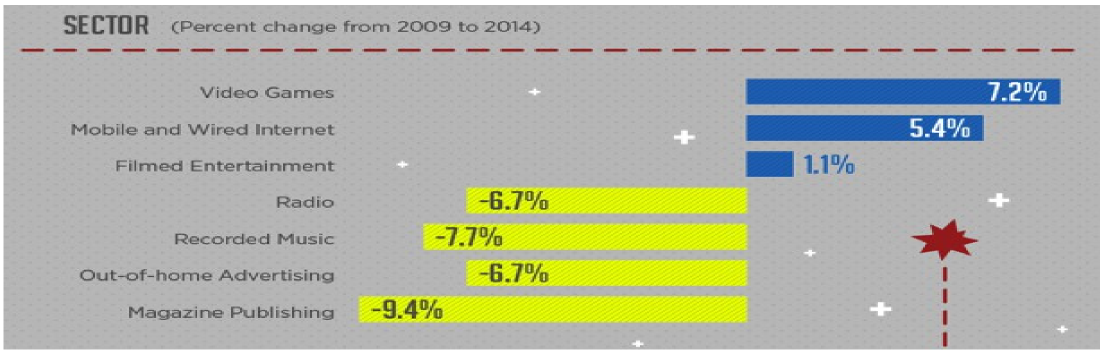
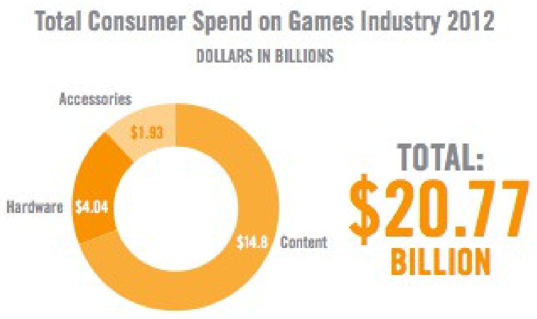


Figure 41 shows the North American gaming sector generated 20 billion dollars in revenue. Accessories and hardware such as game controllers and connectivity was responsible for 30% of total income within this sector. There was a significant growth in 2012 for a time window that contained no next-generation console releases or game-related franchise. The average consumer spending for a four member family reached $255. In conclusion, additional hard accessories for gaming are high in customer interest.



Game controllers dictate how the user interacts and play games. However game controllers are still operated using gamepads similar to platform predecessors.

Popularity and revenue is proportional in consumer electronics. Crowd-funded products undertake product development in the public domain as a market strategy. The Oculus Rift is a virtual reality headset based on physical gestures. Public development resulted in an initial investment of $2.5M [22]. The 'Oculus Rift' holds a recommended-retail price higher than the average household expenditure in this sector. Similarly, the Wii console contains an accelerometer-based interface. The Wii interface aided Nintendo in claiming 85% of console sales in 2012 [20]. However, the Wii is only compatible with Nintendo platforms.

An affordable, multi-platform compatible gaming interface is required that integrates real-time body gestures into an interface to gaming platforms.

4.2 Design Feasibility (AD)

4.2.1 Design Feasibility and Market Research

To ensure successful iterations, a design process was implemented throughout Superhans development phase and is illustrated in the Gantt chart in Appendix F. While waiting for the components to arrive, a rapid prototype was made. The first Superhans prototype had two degrees of motion controlling the mouse movement and one bend sensor triggering the mouse click. This initial prototype was connected to the PC via USB and was successfully tested on online flash games. For the next prototypes, priority was given to make Superhans wireless and was battery powered and equipped with Bluetooth® connectivity. Further iterations were carried out to make add programmable buttons to improve the user centricity of Superhans to allow gamers to play a wide variety of games.

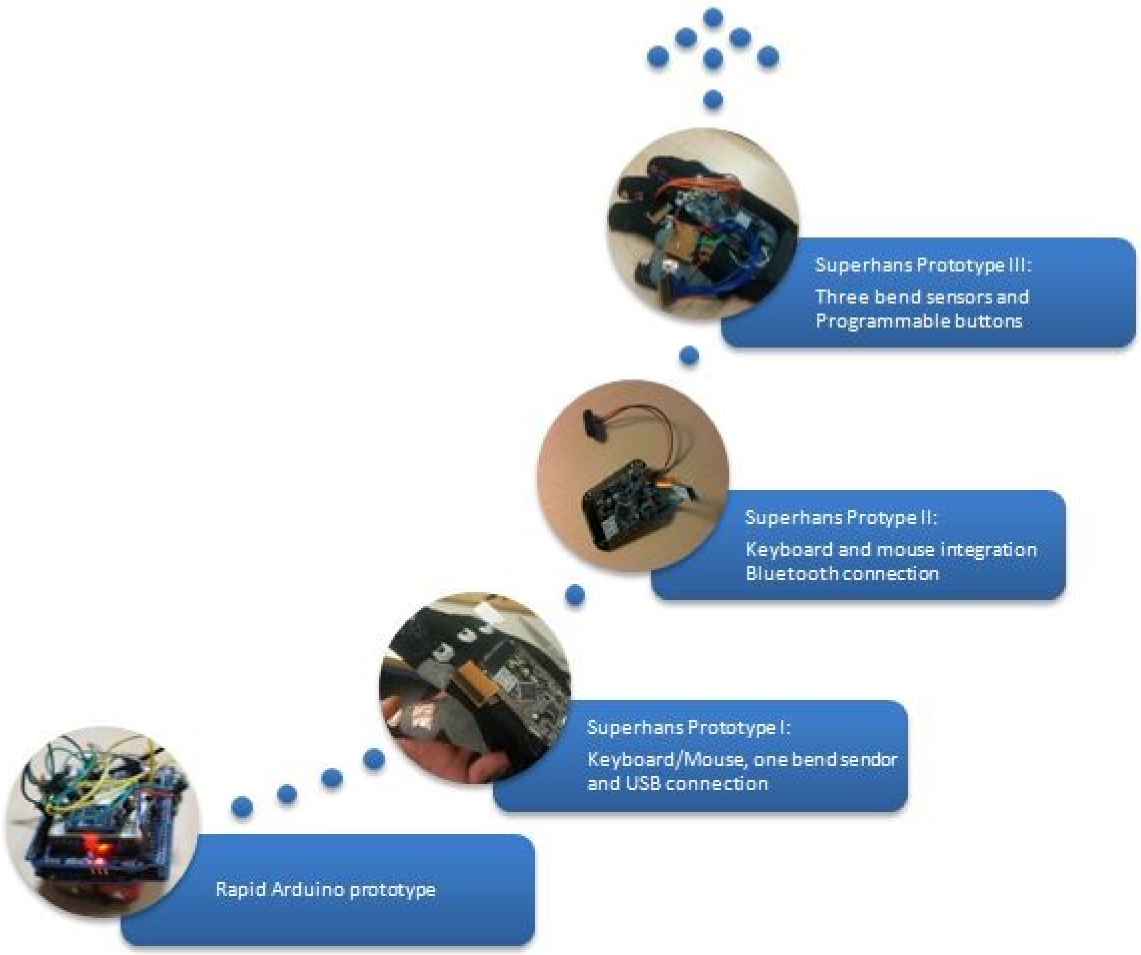

Superhans Prototype III:
Three bend sensors and Programmable buttons
Superhans Protype II:
Keyboard and mouse integration Bluetooth connection
Superhans Prototype I:
Keyboard/Mouse, one bend sendor and USB connection
Rapid Arduino prototype

Figure 38: Superhans prototypes and development phases

4.2.2 User Feasibility

Superhans will weigh around 50-75g, which is comparable to the weight of an average wristwatch. Superhans is designed to feel comfortable allowing several hours of gaming without causing strain; which are common after extended use of traditional hard button controllers. Although Superhans will use state of the art components, some complex algorithms or wireless data transfer might lead to some undesirable lag in the response time.

4.2.3 Market Research

There are several competitor devices presently available in the glove based peripheral control market. These include the Peregrine Wearable Interface, Ion Wireless Air Mouse Glove and P5 Gaming Glove[1][2][3]. These are not open source and are fairly expensive. Peregrine glove only has programmable buttons, which replaces the keyboard but lacks the natural feel, which Superhans offers through its wrist movement. Ion wireless glove on the other hand only control the mouse, hence limited to a few games. Although P5's design is inspiring the 6 degree of freedom and the receiver requirement makes the device quite latent.

4.3 Usability and User Centric Design (RM/JB/RM respectively)

When designing a product of any type, its usability must be taken into regard. Usability is defined as how efficient and fast a user can perform a task with the product [23]. Two usability evaluation methods are presented, one is the questionnaire-based and the other being performance-based

[24] .In the questionnaire based method the user preference is taken into regard and is implemented in the beta testing strategy. Secondly the performance-based method is based on specified performance criteria applied by the designers of the product, both methods have their limitations, thus a hybrid method was used when designing Superhans. A questionnaire was taken before the initial design phase and since the group members played video games extensively it was logical to include their feedback as well. Usability can be classified in three dimensions; the product, user and the task, from these basic three dimensions, other dimensions of usability can be derived and a table of these usability criteria is shown in Figure 43 from [24]. In the design and production processes of Superhans these usability dimensions were taken into concern where dry runs of the commands were taken place during the design meetings to ensure the intuitiveness of the device. The intuitiveness of the device corresponds to the simplicity, consistency, feedback, forgiveness, familiarity and efficiency of the device depicted in Figure 44.

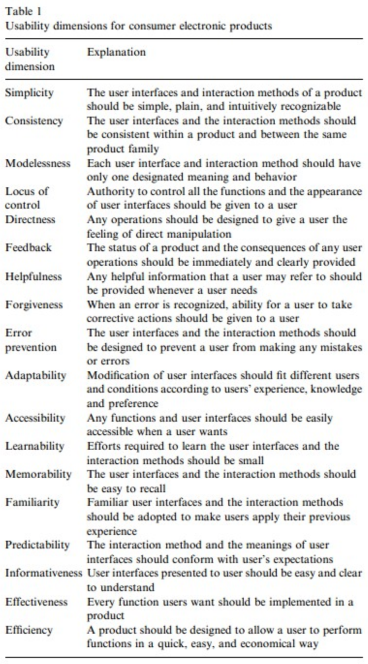


Table 1
Usability dimensions for consumer electronic products

Usability dimension	Explanation
Simplicity	The user interfaces and interaction methods of a product should be simple, plain, and intuitively recognizable
Consistency	The user interfaces and the interaction methods should be consistent within a product and between the same product family
Modelessness	Each user interface and interaction method should have only one designated meaning and behavior
Locus of control	Authority to control all the functions and the appearance of user interfaces should be given to a user
Directness	Any operations should be designed to give a user the feeling of direct manipulation
Feedback	The status of a product and the consequences of any user operations should be immediately and clearly provided
Helpfulness	Any helpful information that a user may refer to should be provided whenever a user needs
Forgiveness	When an error is recognized, ability for a user to take corrective actions should be given to a user
Error prevention	The user interfaces and the interaction methods should be designed to prevent a user from making any mistakes or errors
Adaptability	Modification of user interfaces should fit different users and conditions according to users' experience, knowledge and preference
Accessibility	Any functions and user interfaces should be easily accessible when a user wants
Learnability	Efforts required to learn the user interfaces and the interaction methods should be small
Memorability	The user interfaces and the interaction methods should be easy to recall
Familiarity	Familiar user interfaces and the interaction methods should be adopted to make users apply their previous experience
Predictability	The interaction method and the meanings of user interfaces should conform with user's expectations
Informativeness	User interfaces presented to user should be easy and clear to understand
Effectiveness	Every function users want should be implemented in a product
Efficiency	A product should be designed to allow a user to perform functions in a quick, easy, and economical way

Furthermore, it was ensured that the design process was flexible during the manufacturing process of Superhans. Most of the design was realized but with minor alterations added to the device for a number of reasons. The most significant of which was for usability to make Superhans more intuitive to use. When the final product was realized the product showed simplicity of use, it was used on simple games which required 2D movement and simulators where the intuitive movement of the hand can be used to perform actions in the game. Superhans was tested by volunteers to ensure that the usability of the device was acceptable.

To ensure that Superhans has high usability a model [23] was used. Where the human perception of how a product works is taken into regard.

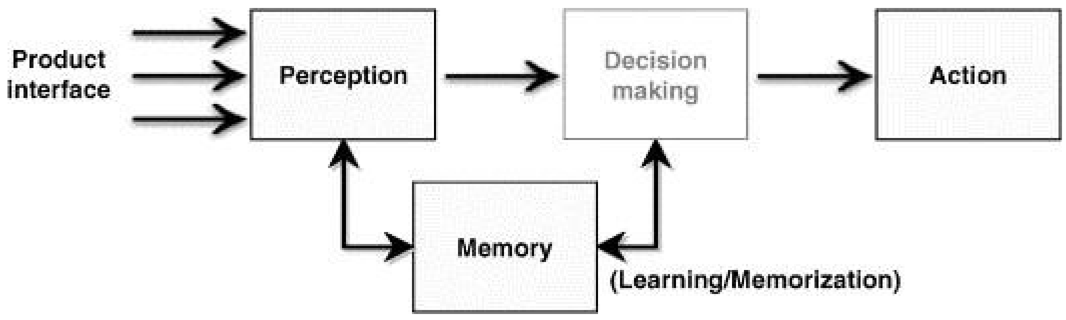


Figure 39: Human information process

Since Superhans is a real-time hand gesture interface the product must be designed in a way to allow the memory process to jump-start the user to perform the action required in [23]. After some trials it was found to be less intuitive than generic game controller, but it had a more gradual learning curve compared to gesture recognition technology used in games nowadays. In order to keep in touch with popular opinion, a simple user centred design process was followed

4.3.1 User Centred Design Implementation (JB)

Before development of the prototype began a survey was conducted involving various people without an electronics background in order to gauge interest for the product and to help us make some final decisions regarding button placement and compatibility issues. A copy of the survey and its results can be found in Appendix F. The results of the survey directly influenced the placement of the buttons found on the glove. The results also revealed that most consumers would like to see the device compatible with all three of the major gaming platforms: PC/Mac, Sony Playstation 3 and Microsoft Xbox 360. At the time of development Xbox One and Playstation 4 (PS4) were not yet released. Due to a lack of material regarding developing for these systems it was decided to remove any thought of Xbox One and PS4 compatibility from the development plan.

This led to the decision of the placement of the I1(see figures 13 and 14) button. This is a button to be used in a situation where a quick action is needed with minimal effort.

The final question of the survey asked if the participant would be happy to be included in the beta testing group. All that responded were added to a dedicated Asana (see section 5 - Project Management) account that included links to video clips of development.

In practice a working prototype was only developed for PC and Mac and so only four respondents were used for beta testing. To ease the beta tester a beta testing strategy consisting of a short playthrough of each of the following games:

- Armour Games' online game Sushi Cat: uses mouse and left click controls.

- Armour Games' online game Bubble Tanks: uses mouse and keyboard controls.
- 505 Games' demonstration copy of Sniper Elite V2: uses advanced Mouse and keyboard controls.

Different glove sizes are needed for reliable reading of the middle finger flex sensor. A particular beta tester had particularly small hands and as such their finger did not bend far enough to trigger a key press.

Solution: Offer different sizes of glove or adjust the bend sensor so that smaller fingers still trigger a key press even though the finger of the glove itself is too large. In practice the sensor position was adjusted and key presses were triggered accurately.

Different users naturally rest their hands in different positions when they do not want to trigger a movement. It was found in some cases that the rest position of certain beta testers triggered the UP movement .The control would drift in an undesired manner. Solution 1: Place a haptic feedback motor on the palm of each glove that would vibrate once whenever the user reaches the predefined (and hardcoded) rest position.

Certain beta testers mentioned that it would be useful if there was a toggle button on the glove to invert the up and down control on both the right and left glove for gamers that preferred to have reverse controls. This feature was implemented on both gloves.

4.3.2 What Can be Done to Improve Usability? (RM)

Since the product was designed and implemented in short time duration other more critical tasks took precedence and so unfortunately not all of the aforementioned usability aspects could be realized. In the future, the electronic circuitry could be hidden for aesthetics. Also more tests could be conducted to adjust the sensitivity of the button presses induced by the flex sensors in the gloves. Furthermore, a bigger test group can be used to provide accurate feedback on the usability and could be used to add the required adjustments to the product for a better user experience.

Windows 7 was considered the reference platform for beta testing. Two games were selected as a point of reference. The former is Bossa Studio Ltd's Surgeon Simulator 2013. Proper Gameplay requires the use of a minimum of three buttons to control the grasp of the thumb, index and middle fingers of an on-screen left hand whilst holding down either the left or right mouse buttons to move the hand down or rotate the hand respectively.

4.4 Quality Assurance (LT)

Quality assurance addresses the procedures for preventing product defects, manufacturing issues and delivery problems. Below are proposals to ensure organised commercial manufacture of Superhans.

Administrative and procedural activities handle product requirements, goals and services. Teams will be managed efficiently using Asana's project management toolset. Perforce a software configuration management tool will be implemented to monitor software revisions and run-time/compilation error history. Management procedures will comply with the International Organization for Standardization (ISO) 9004 protocol [25].

A systematic measurement procedure for verification and testing is to be written characterizing the noise sensitivity of sensors and the power consumption of hardware modules. A Monte Carlo diagram will summarise device's performance distribution due to manufacturing defects and component tolerances. In the manufacturing process defect headroom will allow minor faults to occur within 1% of devices.

Product inspection will be visible to the public. Online services will be provided for customers and clients to provide feedback on the inspection procedure and log any product defects. Products will be accompanied by one year warranty. Additional cover for defects will also be provided.

Hardware labs oversee the integration of ARM® processor chips. Labs will comply with the level two electrostatic discharge regulations (ESD) stated in the UK ISO ESD specifications [26]. It will be made compulsory for employees to wear static bands and antistatic garments to de-sensitive clothing. Devices under test (DUT) must be kept at a distance of 30cm from static materials on lab benches. Polyethylene terephthalate (PET) antistatic bags will enclose devices for relocating hardware. The ARM® processors will be contained on a custom printed circuit board (PCB) enclosed in an anti-static material before shipping to customers. Factory calibration is to be undertaken in the test environment. Verification and testing is to comply with the 17025 ISO protocols [27]. The wired version will contain a removable magnetic connection to the user's platform that will house the firmware specific to the USB™ device. A separate magnetic Bluetooth® enabled module will be made available, giving the user the freedom to upgrade their Superhans ARM® devices. The ARM® device will be enclosed in a plastic vacuum. The enclosure will prevent the interaction of magnetic fields with the ARM® processors (Hall's effect). The wireless device will not contain magnetic material. Magnetic material will interfere with the resonance characteristics of the Bluetooth® antenna due to the effects of mutual inductance.

Sensors will be attached to the PCB with stranded twisted pair cables. This will increase the cable's resistance to mistreatment. Sensors will be encapsulated with a plastic material and glued onto the glove's fingers. Sensors and cabling will be isolated from the user. They will be placed between stitching layers on the glove. Sensor technology will also be encapsulated within anti-static material before integration.

4.5 Manufacturing (TH)

For this product to be commercialised it would need to redesigned for mass production. To convert the prototype design into a product that can be manufactured would require a transition from the FRDM development boards to a dedicated PCB. This miniaturises the circuit and removes the components that are not used on the development board, thus giving the product a more professional look. To reduce the tooling cost of PCB fabrication, only one PCB design is manufactured for both the USB and Bluetooth® version of the device. All the components would be surface mount devices and would be available on tape and reel, for use with pick and place machines. This would consequently reduce the labour cost associated with hand soldering of the components. Surface mount test points would also be added to the system to allow flying probe meters, in order to complete quality control on each PCB produced. During population of the PCB’s, only the components required for each model would be populated to allow both models to be produced on a single production line.

To reduce the complexity of the production line, the microcontrollers would need to be purchased with the software pre-flashed. There would also be a single software build for both models. The software would detect the connected hardware when the device is powered for the first time. The device will then operate in the correct way for that model. There would also be a self-test program run on the first power up to ensure that all the components have been attached correctly and operating as expected.

The glove would be based on an existing design, but with some slight modifications to allow the sensors to be attached. Where possible, the wires would be sewn into the glove using conductive thread, in order to improve the aesthetics of the design. The bend sensors would also be stitched into the correct place to prevent them from moving or becoming dislodged and with the metal squares for detecting finger presses being made of conductive fabric, to allow them to be stitched into place.

The final product would need to undergo testing to ensure it is safe for use. These tests would include electromagnetic radiation test, static discharge tests and drop tests. After each test the design may need to be re-worked to ensure that it complied with the testing standard.

4.6 Sustainability (JB/RM)

4.6.1 Product Developmental Sustainability (JB)

4.6.1.1 Concept

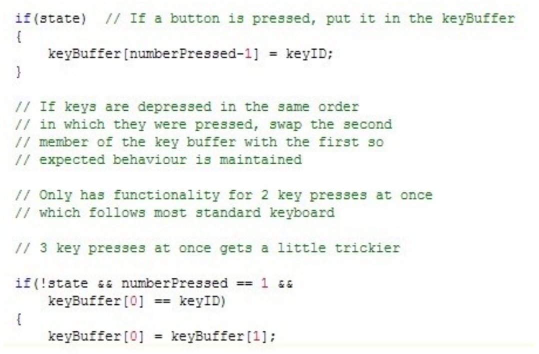
Sustainability is the capacity of a product to be endure an extended period of time. Superhans fits into the wearable gaming sector of the PC peripheral market, as mentioned in the introduction. Products in this market are traditionally expensive and there are very few open source DIY game controllers that allow user to add new features.

To complement the sale of readymade versions, Superhans aims to be one of the first open source gaming controllers with DIY manufacture methodology. The source code will be available in the public domain. This will allow future development of the product through use of the global developer community like Arduino and Raspberry Pi platforms. Indeed, it is designed to complement these two. There are libraries available for the Raspberry Pi and Arduino that support input from external USB™ keyboard and mice. Since Superhans uses standard plug and play keyboard and mouse libraries can be used to control applications in any way the end user desires.

4.6.1.2 Implementation

As previously mentioned, the device uses Freescale's FRDM-KL25Z development board and with the ARM's online library for the mbed platform. Mbed couples an online only C++ language compiler with a range of libraries that anyone can modify to suit their given application. The Superhans code exploits these features. The mbed online community also acts as a version controlled source code repository that's freely available to any user that registers on the mbed website.

The code base and all libraries used in the project will be saved in the mbed project Superhans. This will be open source and can then be imported into any user account. A readme file included with the source code will explain how to set up the project and how to upload the compiled binary file to the user's FRDM-KL25Z. The code comments will indicate where the code can be extended to include a greater functionality as well as how the existing code could be made better, as shown overleaf in figure 45.



```
if(state)  // If a button is pressed, put it in the keyBuffer
{
    keyBuffer[numberPressed-1] = keyID;
}

// If keys are depressed in the same order
// in which they were pressed, swap the second
// member of the key buffer with the first so
// expected behaviour is maintained

// Only has functionality for 2 key presses at once
// which follows most standard keyboard

// 3 key presses at once gets a little trickier

if(!state && numberPressed == 1 &&
    keyBuffer[0] == keyID)
{
    keyBuffer[0] = keyBuffer[1];
```

Figure 40: Example of commented code

However, just submitting the source code only opens the project up to the most intrepid of hobby electronics enthusiasts. To open up the project to a wider audience a step-by-step pictorial guide of how to build a Superhans device will be hosted on http://www.instructables.com/id/SuperHans. This makes the design process readily available, and much more importantly, changeable to meet the requirement of the user. An itemised parts list along with the details of the lowest cost suppliers will also be included.

4.6.1.3 **Target Audiences of the SDK**

The theory behind the open source aspect is that most individuals interested in hobby electronics have played at least one computer game in their past. A DIY game controller gives the hobbyist control of an application that the user will be familiar with that will give instant feedback. It is also implemented in the hope that this will be a good first project that actually features a lot of advanced concepts. The steps on the how to go are easy to follow but such features as RTOS multithreading and USB HID reports encompass sophisticated data handling techniques. A Further Reading section in the instructables.com guide will link the reader to extra resources that, if they are interested, which will uncover the mysteries of the lower level workings of the device.

To those more experienced with electronics and embedded programming the source code repository and step-by-step construction guide acts as a first step towards the enhancement of the product.

4.7 Material Sustainability (RM)

It is vital to provide realistic recycling plans for electronic devices to reuse as much of the material as possible.

The main issue with recycling electronics is that most of material used in electronics are not biodegradable and might contain many hazardous material thus many electronic companies are trying to apply as might biodegradable and reusable material in their products as possible to reduce the impact of their devices. The European Union has issued the WEEE which is considered one of the strictest initiatives to reduce waste caused by electronic and electrical devices [28].

In figure 46 the carbon footprint of the electronics sector is analysed, scope 1 is the carbon footprint under the company or manufacturer's control, scope 2 is the energy purchased from external sources, scope 3 is the emission coming from upstream processes [28]. For practical reasons scope 1 will be our main focus, since it is under our direct control, for the first part iron and steel were not used in the manufacture of Superhans so it is not considered, other services must be considered since some parts were ordered and the process of manufacturing and delivering the parts cause greenhouse emissions, both system design and programming services emissions can be discarded since all of the programming was done internally semiconductors, PCBs and computers would be the area where most of our CO2 emissions might come from because of the presence of the Freescale board.

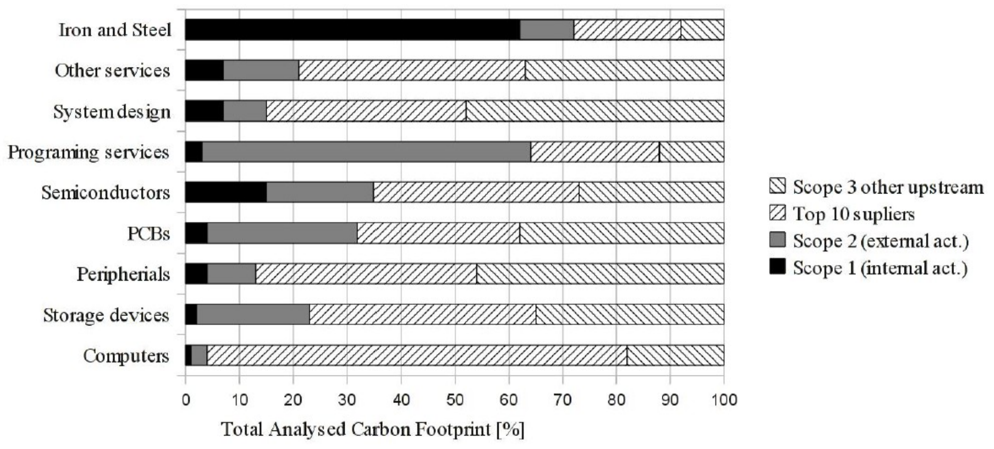


Figure 41: Total analysed carbon footprint of the electronics sector [28]

When designing Superhans sustainability was one of the main issues discussed, where it was concluded that the product would be designed from as much reusable material as possible so that when the product is obsolete waste management centres would easily be able to salvage as much reusable material from our product. First of all when it came to choosing the gloves for Superhans, the gloves chosen had as much fabric as possible since fabric can be easily reused. The discrete electronic components of Superhans can be easily detached and taken to be reused in other projects, and the design of the product allows for the easy attaching and detaching of the brains of the device, the Freescale freedom board.

4.8 Engineering Ethics (HF)

Generally it is the moral obligation of an individual towards his/her organization and working colleagues involved in any aspect of engineering. With regard to Superhans development this relates to the ethical standards that each member of the group exhibits.

There may be individuals who tend to be more persuasive and demanding towards other members of the group. Others may lack the initiative to move outside their comfort zone. These portrayals of different aspects are known as subliminal ethical standards within a group. These viewpoints may change throughout the course of the project as the rapport and individual strategies are established. There are three core fundamental ethical values that every engineer should adhere to [29]:

- Using one's expertise and abilities for the betterment of human welfare.
- Being truthful, unbiased and aiding with loyalty to the masses and to the employer’s hierarchy.
- Endeavouring to maximize the knowledge of the relevant engineering profession by aiding relevant societies.

Discussing these three values further with regards to Superhans, it should be made clear the designing and the manufacture of the device strives to bring benefit to human society. According to the time and budget constraints the device is mainly. The primary focus is to enhance the user gaming experience. But in the long run the benefit to society is endless, for example the device can be used to read sign languages thus reducing the learning curve immensely and also bring the society a step closer to augmented reality.

Apart from the core values mentioned above there are further values which act as guidelines on engineering ethics [30]:

- Safety of the public is of utmost importance

- Public statements issued should be truthful without any unforeseen loopholes.
- There should be no conflict of interest when engineers are acting as agents for clients and they should proceed in a professional manner, revealing truthful answers to all asked queries.
- Healthy competition is always advised but engineers should build their career status purely on their accomplishments and not in unfair competition.
- Always work in such a way that each individual will honor the profession.
- Through an engineer’s professional development they should provide prospects for other engineers under their supervision to develop professionally.

When working in a group, it is each member’s responsibility to make sure that the atmosphere of the group is welcoming and open for suggestions and has minimum discrimination if a group is to work well as a unit and achieve the intended goal. Within the D2 group this was brought upon by how each member reacted to each person’s idea and how it was discussed giving reasons clearly why it might be suitable or not depending on the circumstances. These brainstorming sessions not only gave the objective for our design but also set the tone how the team would work together. When a team works well together a lot of characteristics that build up a very good ethical profile can be seen.

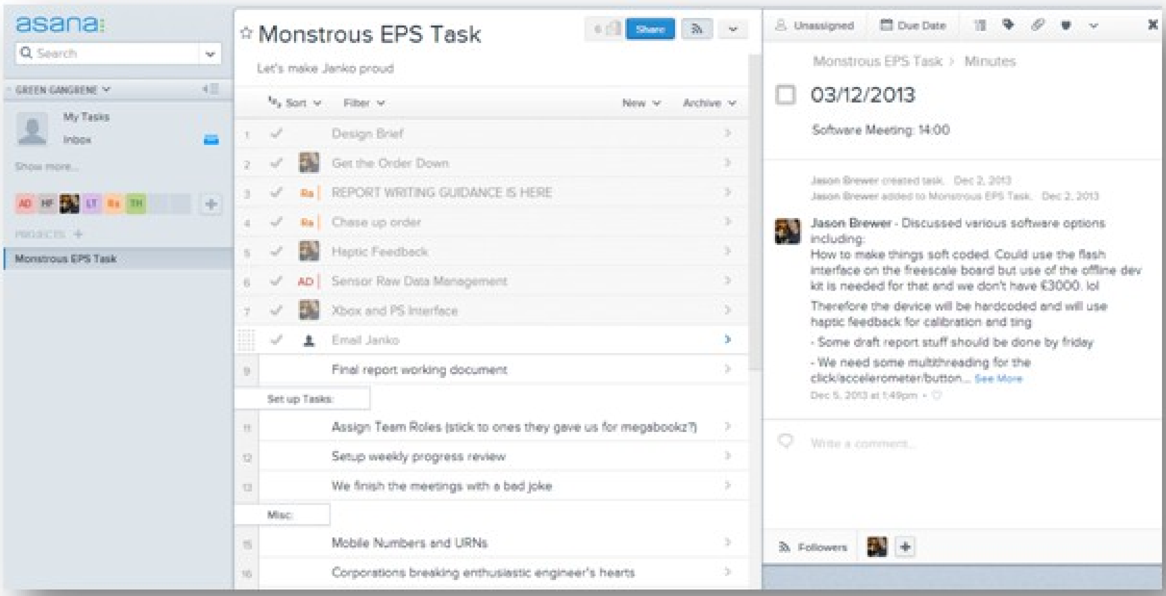
Therefore as a professional engineer it is one's responsibility to adhere to ethical practices and to respect moral obligations towards your colleagues, intendant markets and yourself to reveal any debatable methods used that will eventually lead to a harmful device.

5. Project Management

Project management is a vital part of any product design process. Without sufficient project management the final result would not be realized, progress would not be attained and completion of the project would be on the line. Thus from the beginning of the project an agreement was made that a project management software or website was going to be used.

A project management toolset was integrated into the design process. Team members are assigned tasks and objectives. An archive of task completion, contact details and meeting minutes was also encapsulated onto the project page. In addition, this tool was used to hold discussions about the project design outside of the meetings; also tasks were assigned to members using the website to organize the work flow in the group.

Figure 48 shows a screen shot of the asana.com website showing some of the discussions done during the duration with tasks being assigned.



Monstrous EPS Task

Let's make Janko proud

1 Design Brief
2 Get the Order Down
3 REPORT WRITING GUIDANCE IS HERE
4 Chase up order
5 Haptic Feedback
6 Sensor Raw Data Management
7 Xbox and PS Interface
Email Janko
9 Final report working document

Set up Tasks:

11 Assign Team Roles (stick to ones they gave us for megabookz?)
12 Setup weekly progress review
13 We finish the meetings with a bad joke

Misc:

15 Mobile Numbers and URNs
16 Corporations breaking enthusiastic engineer's hearts

Monstrous EPS Task > Minutes

03/12/2013

Software Meeting: 14:00

Jason Brewer created task. Dec 2, 2013
Jason Brewer added to Monstrous EPS Task. Dec 2, 2013

Jason Brewer - Discussed various software options including:
How to make things soft coded. Could use the flash interface on the freescale board but use of the offline dev kit is needed for that and we don't have €3000. lol
Therefore the device will be hardcoded and will use haptic feedback for calibration and ting
- Some draft report stuff should be done by friday
- We need some multithreading for the click/accelerometer/button... See More
Dec 5, 2013 at 1:49pm

Write a comment...

Figure 48 - Screenshot of the Asana website

Meeting minutes were archived throughout the year. Weekly progress reviews were held to ensure members were on task. Furthermore additional design related tasks required the respective team members to plan proposals prior to lab sessions. All design suggestions and decisions were archived for future reference. An additional thread was held on WhatsApp app as a point of communication between team members.

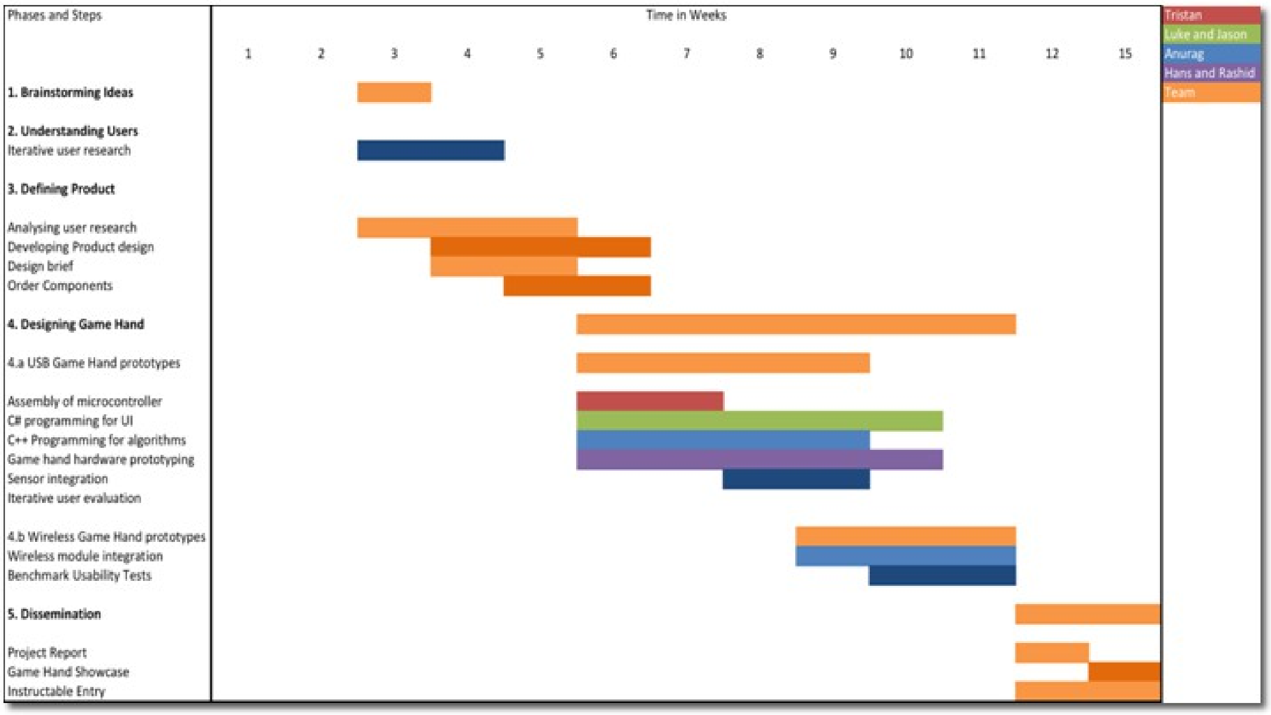


Figure 49 – Diagrammatic work plan of project tasks.

Figure 49 details weekly project tasks. Three teams were designated: Embedded, Hardware integration and Front-end software. The former addresses device calibration, gesture recognition and connectivity. This was the priority task. A timescale of 6 weeks was designated. Hardware integration's objective is to propose a sustainable alternative to commercial flex sensors. This task was designated 3 weeks. In reality, component delivery was delayed resulting in an initial time extension. An unstable design was proposed. Fixing this proposal took an additional 2 weeks. The latter team's objective is to integrate a graphical user interface for compatibility with OS X, Windows and Linux. The objective was completed in the 3 week timescale.

6. Conclusion (All)

The main objective was to design a device that would replace the analogue sticks and push buttons of traditional gaming controllers. State of the art gesture recognition techniques were explored and their feasibility for the Superhans was evaluated. In addition to these gesture recognitions techniques Superhans includes 16 buttons and three flex sensors to send predefined key presses and mouse clicks via a wired/wireless interface. Two separate button designs were implemented in the form of copper strips and placed in easily accessible locations. Flex sensors, also sending key presses and a left mouse click to complement the natural movement. The purpose of building flex sensors was to add another sustainable aspect to the project. These were complemented by haptic feedback in the form of four vibration motors.

The hardware choices enabled efficient development. The mbed platform has allowed the use of the advanced features available on the microcontroller with ease. The FRDM board has been a successful hardware prototyping tool and allowed us to prototype the functionality of Superhans without having to spend time developing our own PCBs. The USB™ HID connection to a computer has proved a worthy choice. This allows Superhans to be used with all common operating systems without the need for any specific drivers or software.

An iterative design process was implemented throughout the product development phase. User feedback helped to shape the product towards the user from the outset of the initial design. A GUI provides a user interface between product functionality and system parameters/online support. A multi-threading architecture is proposed to relieve the Windows Form from computational heavy tasks, and freezing the window handling classes form processing key threads. The GUI utilizes graphics pipelines to render menu icons and 3D models to assist the user during navigation and device operation.

A developmental sustainability framework was defined consisting of a construction guide on www.instructables.com and the hosting of source code in the public forum. This aimed to lay the foundation to allow the global developer community to grasp the core concepts of Superhans and provide sustained evolution of the device.

Quality assurance addresses the procedures used for preventing product defects, manufacturing issues and delivery problems. Software configuration management and project management tools are proposed to fulfil administrative and procedural requirements based on approved ESD environments. Currently the sustainability of electronics is of great importance considering the staggering amount of products being manufactured. This was an important factor in the manufacturing of Superhans and was made sure that the materials used to manufacture the product have minimal environmental impact. Certain sustainability criteria were presented and abided to during the duration of the project.

Superhans does however have certain limitations that were highlighted in the beta testing procedure. The device is not compatible with gaming platforms apart from Windows, Mac and Linux systems. The embedded code could be made more streamlined through the design of a custom gamepad HID report. This would also allow greater future expandability.

The GUI has not yet been compiled on a Linux system and, using the .NET framework and OpenGL, it must be recompiled to work on different operating systems. The key mapping feature of the GUI is also simple a point of reference to hard coded key mapping as oppose to a system to modify the key presses sent. The calibration suite is also non-functional.

The lack of sensitivity to gradual change is the main limitation of the DIY flex sensor. This can lead to inaccurate readings.

Superhans successfully provides a novel and intuitive method for computer game control and beyond.

References

[1] “The Peregrine” Internet: http://theperegrine.com/, [Jan. 12, 2014].

[2] “ION Air Mouse” Internet: http://www.ionwirelessairmouse.com/, [Jan. 12, 2014]

[3] “Essential Reality P5 Glove” Internet: http://inition.co.uk/3D-Technologies/essential-reality-p5-glove, [Jan. 12, 2014]

[4] Arduino, "Arduino" Internet: http://arduino.cc/, [Jan. 9, 2014]

[5] mbed, "Explore mbed" Internet: http://mbed.org/explore/, [Jan. 9, 2014]

[6] USB™ implementers' Forum, "Device Class Definition," 2001. Internet: http://www.usb.org/developers/devclass_docs/HID1_11.pdf, [Jan. 9, 2014]

[7] ARM, “KEIL™ Tools by ARM” Internet: http://www.keil.com/rl-arm/kernel.asp, [Jan. 9, 2014]

[8] JCoHarris, “Sketch Feed” Internet: http://jcoharris.com/sketch-feed/, [Jan.9, 2014]

[9] Spark Fun Electronics Inc, “Flexiforce Pressure Sensor - SEN-08712” Internet: https://www.sparkfun.com/products/8712, [Jan. 9, 2014]

[10] FRDM-KL25Z User's Manual 2013-10-24 Rev 2.0

[11] B. M. Lee-Cosio, C. Delgado-Mata, and J. Ibanez, “ANN for Gesture Recognition using Accelerometer Data,” Procedia Technol., vol. 3, pp. 109–120, Jan. 2012.

[12] J. Liu, Z. Wang, L. Zhong, J. Wickramasuriya, and V. Vasudevan, “uWave: Accelerometer-based personalized gesture recognition and its applications,” 2009 IEEE Int. Conf. Pervasive Comput. Commun., pp. 1–9, Mar. 2009.

[13] “Freescale Semiconductor, ‘Xtrinsic MMA8451Q 3-Axis, 14-bit/8-bit Digital Accelerometer’ MMA8451Q datasheet [Revised Oct. 2013].”.

[14] “Hand Anatomy, American scoeity for surgery of the hand. Available at http://www.assh.org/public/handanatomy/Pages/default.aspx.” .

[15] - mbed, “mbed library internals”, Internet: http://mbed.org/handbook/mbed-library-internals, [Jan. 9, 2014]

[16] - mbed, “RTOS - Handbook”, Internet: http://mbed.org/handbook/RTOS, [Jan. 9, 2014]

[17] - Johan Kritzinger, “MMA8451Q Library” Internet: http://mbed.org/users/JoKer/code/MMA8451Q/, [Jan. 9, 2014]

[18] - mbed, “USBMouseKeyboard - Handbook”, Internet: http://mbed.org/handbook/USBMouseKeyboard, [Jan. 10, 2014]

[19] Xamarin, “Xamarin - Build apps with C#”, Internet: http://xamarin.com/, [Jan. 10, 2014]

[20] Sebastian Anthony, “ PC gaming vs. consoles, the infographic”, Internet: http://www.extremetech.com/gaming/97705-pc-gaming-vs-consoles-the-infographic, [Jan. 10, 2014]

[21] Entertainment Software Association, “2013 Sales, Demographic and Usage Data - Essential facts about the computer and video game industry”, Internet: http://www.theesa.com/facts/pdfs/esa_ef_2013.pdf [Jan. 10, 2014]

[22] Oculus, “Oculus Rift: Step into the game”, Internet: http://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game [Jan. 10, 2014]

[23] J. Kim and S. H. Han, “A methodology for developing a usability index of consumer electronic products,” Int. J. Ind. Ergon., vol. 38, no. 3–4, pp. 333–345, Mar. 2008.

[24] S. H. Han, M. H. Yun, J. Kwahk, and S. W. Hong, “Usability of consumer electronic products,” vol. 28, no. January 2000, pp. 143–151, 2001.

[25] Managing for the sustained success of an organization -- A quality management approach, British Standard BS EN ISO 9004:2011, November 2009.

[26] ElectrostaticsProtection of electronic devices from electrostatic phenomena. General requirements, British Standard BS EN 61340-5-1:2007, May 2008.

[27] General requirements for the competence of testing and calibration laboratories, British Standard BS EN ISO 17025:2005, December 2013.

[28] M. Sloma, “Carbon footprint of electronic devices,” vol. 8902, p. 890225, Jul. 2013.

[29]Martin, Mike & Schinzinger, Ronald: Ethics in Engineering, 3rd Ed. McGraw Hill

[30]W. Frey, "Ethics of Teamwork," Connexions, p.6, 2011.

Appendices

Appendix A - Design Brief (All)

Design Brief: Hand Gesture Interface for Real Time Applications

The design brief describes the "Superhans" gaming controller. The objective of this document is to outline the key elements of the product design.

Innovation

The gaming industry is a prominent player in consumer electronics. Popularity within this sector has significantly risen over the last decade in comparison to other entertainment sectors.

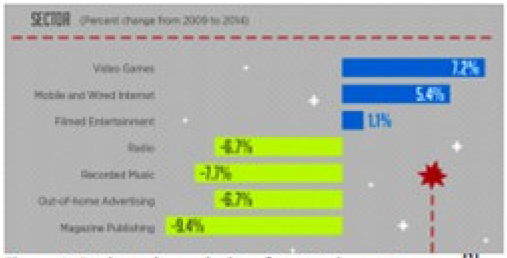


Figure 1: Projected popularity of entertainment sectors [1]

Our proposal is "Superhans", a hand-fitted multi-platform gaming controller. "Superhans" replaces traditional analogue sticks/mouse interface with a combination of sensors triggered by hand gestures.

In the late 20th century, Mattel and Reality Quest released hand gestured controllers. However, these wired modules lacked a 'slick' design and were released when game platforms were expensive and lacked popularity. Both controllers lacked multi-platform support. In 2012, the "Leap Motion" attempted to integrate hand gestures with image processing. Although popular this device is sluggish, reducing the frames per second of gameplay for the handful of Mac and PC compatible games. Driver installation further complicated hardware setup and mobility. Similarly the Kinect's pricing & installation process deters consumers.

The software development kit (SDK) will promote versatile development for other applications requiring hand localisation such as fencing simulators, virtual object interaction, sign language and drawing tools.

"Superhans" will be compatible with all Bluetooth and Universal Serial Bus (USB) enabled platforms. It will automatically connect to smart phones, game consoles via an interface and PC/Mac devices. "Superhans" also automates hardware connectivity and installation without sacrificing weight, mobility and performance.

Technical Communications

The technical aspects of the "Superhans" system can be spit up into several main blocks as shown below.

Each team member has been assigned a specific role in the production of the systems. Tristan is writing low-level drivers for the microcontroller. Hans is in charge of the design and ascetics of the glove with Rashid. Anurag is designing the sensor data decoding algorithms. Luke and Jason will look at the calibration and key event mapping Graphical User Interface (GUI).

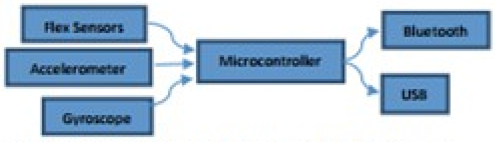


Figure 2: A block diagram of the "Superhans" controller

Design Feasibility

Since we are limited by time constraints, it will be challenging to achieve a fully functional cross-platform controller. Therefore an iterative design process will be implemented. In the first iteration, "Superhans" with be equipped with an accelerometer and one bend sensor. This initial prototype will only be functional for Windows games. The next iteration will integrate a Bluetooth module for wireless connectivity. The final iteration will increase accuracy by adding another three degrees of freedom. If time permits, "Superhans" will be cross platform compatible accompanied by a GUI with calibration and key event mapping.

Although gaming industry is a good target market for generating revenue, it can be beneficial for teaching sign language hand gesture.

The software will be optimised to reduce computation time and minimise latency giving the user an undesirable gaming experience.

Legal Issues

"Superhans" uses open source software. Additionally there are no patent clashes with releasing an ARM chipset within our product as long as it's not marketed under ARM's branding. The same is true for the Bluetooth chipset.

Pricing

Accessories such as game controllers own 10% of the market share, comprising a $2bn industry in the United States alone.

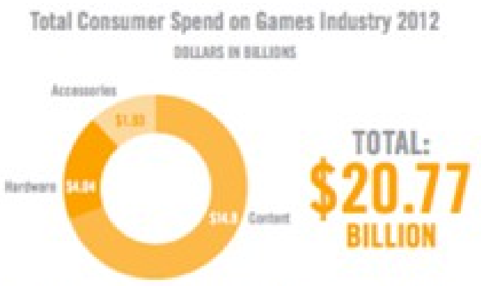


Figure 1: Distribution of US consumer spending within the gaming industry [2].

"Superhans"' prototype manufacturing costs are estimated at £30 for the wired module and £50 for the wireless. Considering bulk production costs, "Superhans" will be competitively priced. For example, the recommended retail price of the Xbox One controller is £54.99.

Sustainability

To promote productivity in the developer ecosystem, "Superhans" will offer open source libraries in a SDK. Furthermore, all code used in the design as well as a step-by-step guide to constructing "Superhans" will also be made readily available online. The aim of this strategy is to ensure an organic growth of features and expansion of user applications through iterative global collaboration. The sensors used in our design are all hand made from plastic and copper wires found in abundance. Secondly, it was made sure that the glove used was made of materials, which can be recycled or removed and used elsewhere when the product estimated life span is over. The glove that will be used is the Storm "SixSixOne" which uses 50% leather which can be either recycled or reused elsewhere, 40% polyester and 10% cotton, both could easily be recycled using known recycling means.

Where possible circuits will be designed to be easily salvageable or repairable. Finally, the power source which will power our product will be a rechargeable 9 volt battery to avoid throwing away used batteries which contain heavy metals which cause great harm to the ecosystem.

This will be compatible with low end hardware specification to deter platform updates.

Usability

"Superhans" targets the average gamer – A 30-year-old male [2]. The prototype glove will be medium sized to serve the majority audience.

"Superhans" will weigh around 50g to 75g, which is comparable to the weight of an average wristwatch. "Superhans" will be incorporated with a padded glove for a comfortable user experience. This will allow several hours of gaming without causing strain injuries, blisters and carpal tunnel syndrome; which are common problems faced after extended use of traditional hard button controllers. The product will be accompanied by a GUI and a help page for general assistance.

User-centric design

User feedback will be harvested and acted upon at each stage of the design process in order to accurately build up a complete picture of the user environment and their goals in using "Superhans". A pre-prototype survey has already been conducted on our target audience and a group of beta testers has been identified. The praise and criticism of this group will be constantly reassessed in order to maximise customer satisfaction before the initial product release.

Engineering Ethics

It should be made clear that the design of "Superhans" is carried out with the intention of bringing benefit to human society.

Within the time and budget constraints the device is mainly focused on enhancing the gaming experience. But in the long run the benefits to society are endless: for example the device can be used to read sign language gestures.

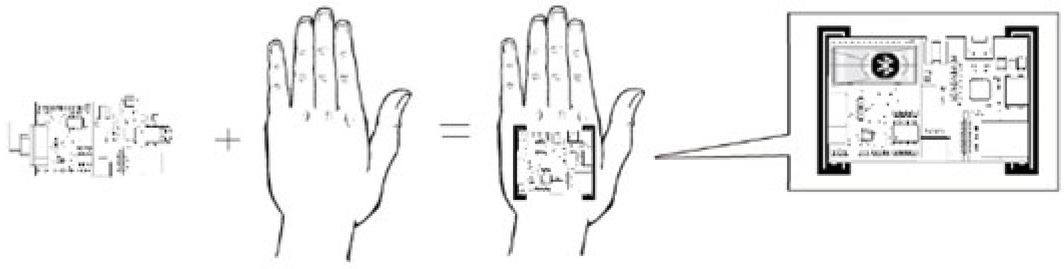
It is also vital that the students involved are truthful and impartial to the ideas brought up by other team members and be loyal by adhering to the criteria imposed by academia; for instance keeping the budget under £100 and not purchasing components without any consent.

References

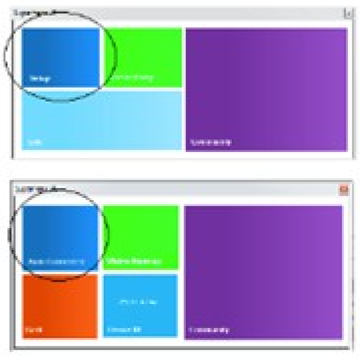
[1] PC gaming vs. consoles, the infographic, Sebastian Anthony, ExtremeTech.com [Web-Document, 2011].
[2] 2013 Sales, Demographic and Usage Data, National Purchase Diary (NPD) Group/Games Market Dynamics, US. [Web Document, 2013]

Appendix B - Installation User Manual (LT)

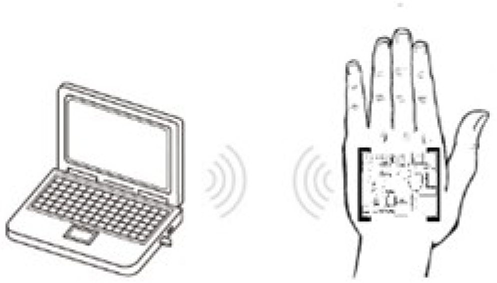
①



②

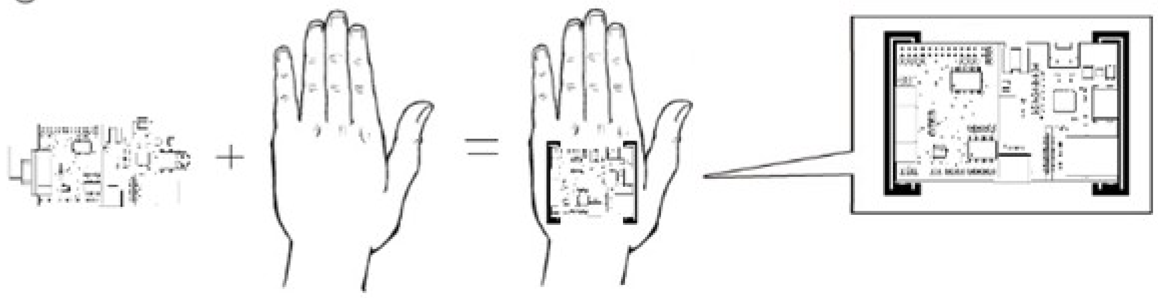


③

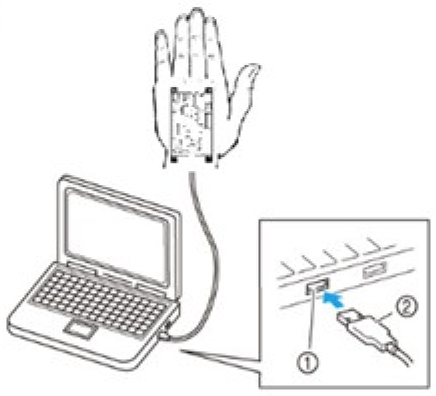


④

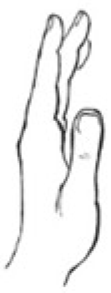
①



②



③



Appendix C - Frequently Asked Questions (LT)

How much does Superhans cost?

You can purchase a manufactured Superhans model from any of our recommended retailers. The GUI and Developer's tools are free, however do require Superhans to operate. Keen hobbyists are encouraged to have a go at making their own. You can refer to our instructables page from the Superhans UI to find out instructions on how to do this.

How do I install the GUI?

Go to the following "sourceforge" link to download the installer for your respective operating system. Open the installer and follow the instructions.

The UI is only available on Mac, Windows and Linux. Android, BBM and iOS applications are in working progress.

The PS4 and Xbox One do not require a GUI. The Superhans will auto-detect the console it is connected to.

Do I need to run the GUI to make Superhans work?

For Windows, Mac and Linux, the GUI initially needs to be installed for the one-time connectivity setup. However once installed, you can enable or disable auto-connection to the Superhans.

How do I setup Superhans' BlueTooth connection?

To auto-detect the Superhans, simply open the UI, navigate to the Connectivity menu and select auto-detect. Finally authenticate the BlueTooth connection on your machine the usual way. The BlueTooth device ID will be sniffed from the device and viewed under the auto-detect menu on the UI.

Once enabled, Superhans can act as a wireless mouse but doesn't override your standard mouse and keyboard features.

Auto-detect can be manually enabled or disabled.

I think the finger or tilt sensors are broken. How can I check?

Open the calibration tool under Setup on the UI. This will display a graphical hand that responds to Superhans' functionality. Use this to verify the sensors are not working.

If the Superhans has a fault, contact our help team.

I'd like to build one. How do I go about this?

Take a look at our Instructables page. You can navigate to the page via the UI. Select Community->Developers->Instructables.

The glove feels unnatural. Can I change the hand orientation?

To calibrate the Superhans open the Calibration menu under setup on the UI. Once opened a graphical hand will display moving to Superhans' features. On your Superhans press the calibration button to save the current position as your hand's natural position. Superhans takes care of the rest.

This sounds like a glorified Wii Remote. How does it differ from Wii?

Superhans does not need to be in the line-of-site with the Wii console - And it plays in pitch black! It is multiple-platform compatible and relies on being operated through all natural hand gestures. USers can also control the calibration of Superhans. Not to mention our online community and developing base!

How can I setup Superhans with the PS4, Xbox One, Windows, Mac or Linux

operating system?

For Windows, Mac, Linux, Android and iOS see how do I install the GUI.

For PS4 and Xbox One, just plug-in and play via USB. Wireless connectivity is currently working progress.

Can Superhans remember my game mapping?

Unfortunately not. We are not licensed to work directly with game publishers. This is something we are interested in the future. We are happy to co-operate with developers interested in this feature. Please contact us via the Developer's page on our website.

Superhans is setup to automatically calibrate gestures to the standard controls of console and computer games. If this is setup not to your taste, you can change the game controls the usual way. We do offer a service to remember what manual mapping you have undertaken for games. This can be found on the UI under Setup->Key Mapping.

My question is not on here. How do I contact customer support?

For Developers: superhans@communitySupport.com For Users: superhans@customerSupport.com

Appendix D – Embedded Source Code (JB/TH/AD)

```
/* SuperHansRightGlove.cpp


Code for WIRED RIGHT GLOVE

Sends mouse movement dependent on tilt
of the glove, index finger left mouse click,
middle finger 'q' press and various
finger/palm keys corresponding to various
keys on a standard keyboard

For use with:
FRDM KL25Z

Authors:
Tristan Hughes
Anurag Dhutti
Jason Brewer

*/



#include "mbed.h"
#include "rtos.h"
#define MMA8451_I2C_ADDRESS (0x1d<<1)
#include "MMA8451Q.h"
#include "USBMouseKeyboard.h"
#include "USBHID.h"

// Misc Init

DigitalOut led(LED1);
MMA8451Q acc(PTE25, PTE24, MMA8451_I2C_ADDRESS);
AnalogIn ain(PTB1);    // Forefinger sensor
AnalogIn ain1(PTB0);   // Second finger Sensor
USBMouseKeyboard key_mouse;
Mutex USB_mutex;

// Button Init
// D-Pad Palm Switches
DigitalIn _up(PTE5);
DigitalIn _left(PTE4);
DigitalIn _down(PTE3);
DigitalIn _right(PTE2);
DigitalIn _sel(PTB11);
DigitalIn _start(PTB10);

// Finger Switches
DigitalIn _i1(PTE31);
DigitalIn _s1(PTA17);
DigitalIn _s2(PTA16);
DigitalIn _s3(PTC17);
DigitalIn _s4(PTC16);
DigitalIn _th1(PTC13);
DigitalIn _th2(PTC12);
DigitalIn _th3(PTC11);
DigitalIn _th4(PTC10);
DigitalIn _pi1(PTC6);

void click_thread(void const *args)
{
  bool press[2];
  bool lPress = 0;
  //bool rPress = 0;
  // PLEASE NOTE:
  // The following code does not allow both mouse
  // buttons to be pressed down at the same time.
  // Your system will register the press of each button
  // but as soon as one is released the other will also
  // be released. Review your application to see if this is
  // applicable.

  while(true) {
    lPress = ain.read()<0.5;
   //key_mouse.printf("Fore: %f Sec: %f\n",ain.read(), ain1.read());

     if( lPress && (press[0] == 0) ) {
        press[0] = 1;
        led != led;
```

```
          //key_mouse.printf("L: %i\n", lPress);
          USB_mutex.lock();
          key_mouse.press(MOUSE_LEFT);
          USB_mutex.unlock();
      } else if ( !lPress && (press[0] == 1) ) {
          press[0] = 0;
          //key_mouse.printf("L: %i\n", lPress);
          USB_mutex.lock();
          key_mouse.release(MOUSE_LEFT);
          USB_mutex.unlock();
      }
      if(!_i1 && press[1] == 0) {
      //if( rPress && (press[1] == 0) ) {
           //key_mouse.printf("R: %i\n", rPress);
           press[1] = 1;
           if(!lPress) USB_mutex.lock();
           key_mouse.press(MOUSE_RIGHT);
           if(!lPress) USB_mutex.unlock();     // As Right mouse click
       }else if(_i1 && press[1] == 1) {
      //}else if( !rPress && (press[1] == 1) ) {
           //key_mouse.printf("R: %i\n", rPress);
           press[1] = 0;
           if(!lPress) USB_mutex.lock();
           key_mouse.release(MOUSE_RIGHT);
           if(!lPress) USB_mutex.unlock();
       }

      Thread::wait(100);
  }
}


void mouse_thread(void const *args)
{
  while(true) {
      USB_mutex.lock();
      key_mouse.move(-acc.getAccZ()*10, 0);
      key_mouse.move(0, -acc.getAccX()*10);
```

```
        USB_mutex.unlock();
        Thread::wait(1);
    }
}

void SendKeyPress (uint8_t keyID, bool state, uint8_t numberPressed)
{
    // Send a simulated keyboard keypress. Returns true if successful.
    static HID_REPORT report;
    static uint8_t last_pressed;
    static uint8_t keyBuffer[3];
    report.data[0] = REPORT_ID_KEYBOARD;
    report.data[1] = 0;
    report.data[2] = 0;
    report.data[6] = 0;
    report.data[7] = 0;
    report.data[8] = 0;

    report.length = 9;

    // Multiple button press functionality

    if(!numberPressed)  // Clear the keyBuffer if no buttons are pressed
{
        keyBuffer[0] = 0;
        keyBuffer[1] = 0;
        keyBuffer[2] = 0;
  }

    if(state)  // If a button is pressed, put it in the keyBuffer
{
        keyBuffer[numberPressed-1] = keyID;
    }

    // If keys are depressed in the same order
    // in which they were pressed, swap the second
    // member of the key buffer with the first so
    // expected behaviour is maintained
    // Only has functionality for 2 key presses at once
    // which follows most standard keyboard
    // 3 key presses at once gets a little trickier
```

```
      if(!state && numberPressed == 1 &&

       keyBuffer[0] == keyID)
{
       keyBuffer[0] = keyBuffer[1];
       keyBuffer[1] = 0;
   }

//key_mouse.printf("keyBuffer: %X, %X, %X\n", keyBuffer[0], //keyBuffer[1], keyBuffer[2]);

   if(numberPressed == 0)
   {
       report.data[3] = 0;
       report.data[4] = 0;
       report.data[5] = 0;
   }else if(numberPressed == 1)
   {
       report.data[3] = keyBuffer[0];
       report.data[4] = 0;
       report.data[5] = 0;
   }else if(numberPressed == 2)
   {
       report.data[3] = report.data[3];
       report.data[4] = keyBuffer[1];
       report.data[5] = 0;
   } else if(numberPressed == 3)  // not actually functional
   {
       report.data[3] = report.data[3];
       report.data[4] = report.data[4];
       report.data[5] = keyBuffer[2];
   }

   key_mouse.send(&report);

}



void keyboard_thread(void const *args)
{
   uint8_t numberPressed = 0;
   bool press[16] = {0};
   while(true) {

       // Uncomment for debug
       //key_mouse.printf("No: %d\n", numberPressed);
       /*key_mouse.printf("pressArr: %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d
%d\n",

       press[0], press[1], press[2], press[3], press[4], press[5], press[6], press[7],

       press[8], press[9], press[10], press[11], press[12], press[13], press[14], press[15]);*/


       // Makes the second bend sensor map to the key "q"
       // Mapping the second bend sensor to right mouse
       // poses some issues to do with how the clicks are sent
       // in the mbed libraries. An indepth analysis
       // of HID reports is required and tbh cba ukno

       if( (ain1.read()<0.25) && (press[0] == 0) ) {
          if(press[0] == 0) numberPressed++;
          press[0] = 1;
          //key_mouse.printf("   qPress: %i\n", press[0]);
          //USB_mutex.lock();
          //key_mouse.press(MOUSE_RIGHT); // Sends right mouse click
          //USB_mutex.unlock();
          SendKeyPress(0x14,1,numberPressed);     // Sends 'q'
//keypress
    } else if ( (ain1.read()>0.25) && (press[0] == 1) ) {
          if(press[0] == 1) numberPressed--;
          press[0] = 0;
          //key_mouse.printf("R: %i\n", press[0]);
          //USB_mutex.lock();
          //key_mouse.release(MOUSE_RIGHT); // releases right mouse
click
          //USB_mutex.unlock();
          SendKeyPress(0x14,0,numberPressed);
       }else
       {
          numberPressed = numberPressed;
       }

       // SECOND FINGER BUTTONS (4 of them)


       if(!_s1 && press[1] == 0) {
          if(press[1] == 0) numberPressed++;
          press[1] = 1;
          SendKeyPress(0x08,1,numberPressed);     // Sends 'e'
keypress (a = 0x04)
          //SendKeyPress(0x04,1,numberPressed);
       } else if(_s1 && press[1] == 1) {
          if(press[1] == 1) numberPressed--;
          press[1] = 0;
          Thread::wait(17);
          SendKeyPress(0x4,0,numberPressed);
       }else
       {
          numberPressed = numberPressed;
       }


       if(!_s2 && press[2] == 0) {
          if(press[2] == 0) numberPressed++;
          press[2] = 1;
          SendKeyPress(0x15,1,numberPressed);     // Sends 'r'
keypress (w = 0x1a)
          //SendKeyPress(0x1a,1,numberPressed);

       } else if(_s2 && press[2] == 1) {
          if(press[2] == 1) numberPressed--;
          press[2] = 0;
          Thread::wait(17);
          SendKeyPress(0x1a,0,numberPressed);
       }else
       {
          numberPressed = numberPressed;

       }
       if(!_s3 && press[3]== 0) {
          if(press[3] == 0) numberPressed++;
          press[3] = 1;
          SendKeyPress(0x17,1,numberPressed);     // Sends 't' keypress (s = 0x16)
          //SendKeyPress(0x16,numberPressed);
       } else if(_s3 && press[3]== 1) {
          if(press[3] == 1) numberPressed--;
          press[3] = 0;
          Thread::wait(17);
          SendKeyPress(0x17,0,numberPressed);
       }else
       {
          numberPressed = numberPressed;
       }

       if(!_s4 && press[4]== 0) {
          if(press[4] == 0) numberPressed++;
          press[4] = 1;
          SendKeyPress(0x1c,1,numberPressed);     // Sends 'y'
keypress (d = 0x07)
          //SendKeyPress(0x07,0);
       } else if(_s4 && press[4]== 1) {
          if(press[4] == 1) numberPressed--;
          press[4] = 0;
          Thread::wait(17);
          SendKeyPress(0x1c,0,numberPressed);
       }else
       {
          numberPressed = numberPressed;
       }

      // THIRD FINGER BUTTONS (4 of them)


       if(!_th1 && press[5]== 0) {
          if(press[5] == 0) numberPressed++;
          press[5]= 1;
          SendKeyPress(0x18,1,numberPressed);     // Sends 'u'
keypress
```

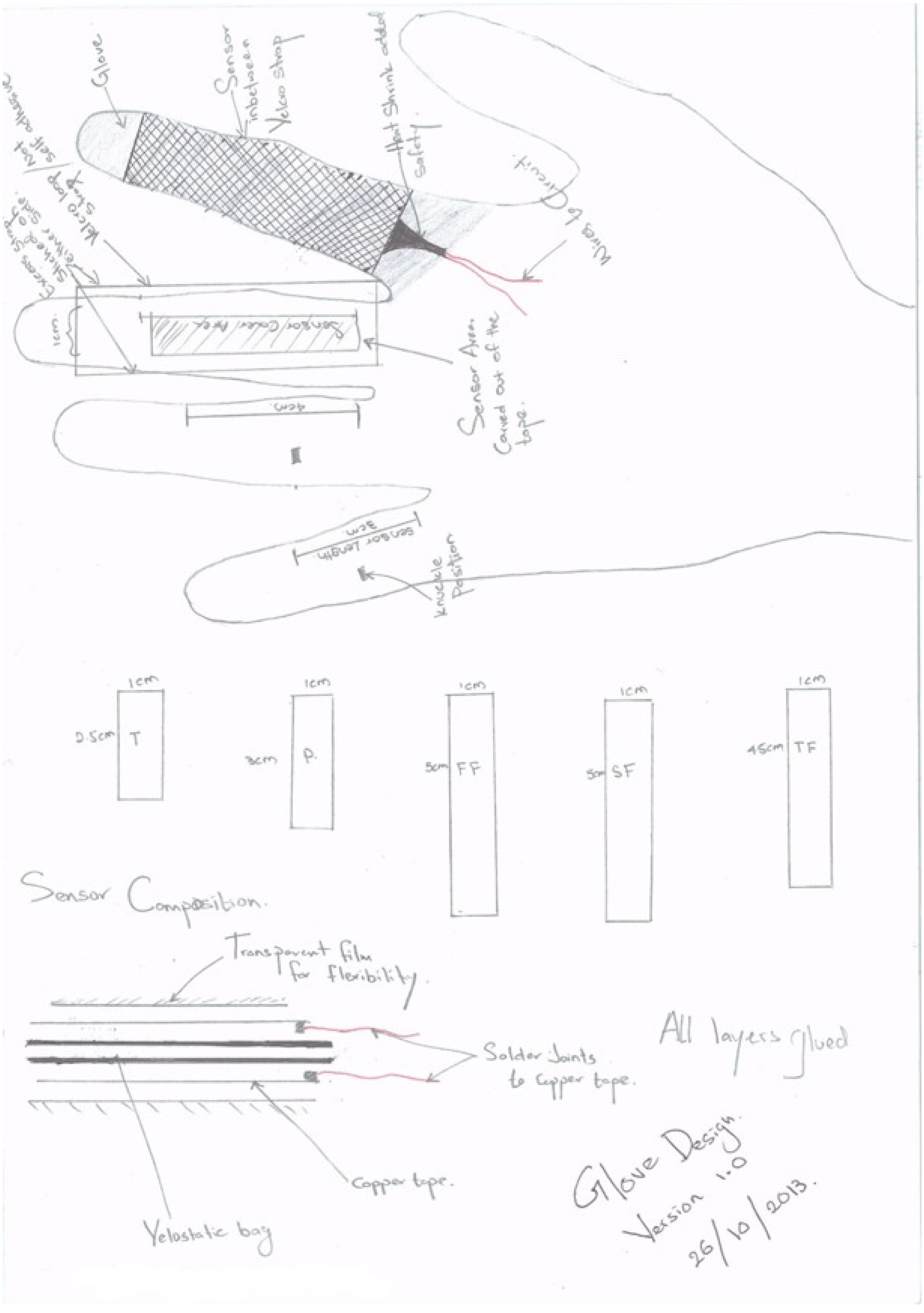
```
    } else if(_th1 && press[5]== 1) {
      if(press[5] == 1) numberPressed--;
      press[5]= 0;
      SendKeyPress(0x18,0,numberPressed);
    }else
    {
      numberPressed = numberPressed;
    }

    if(!_th2 && press[6]== 0) {
      if(press[6] == 0) numberPressed++;
      press[6]= 1;
      SendKeyPress(0x0c,1,numberPressed);     // Sends 'i'
keypress

    } else if(_th2 && press[6]== 1) {
      if(press[6] == 1) numberPressed--;
      press[6]= 0;
      SendKeyPress(0x0c,0,numberPressed);
    }else
    {
      numberPressed = numberPressed;
    }

    if(!_th3 && press[7]== 0) {
      if(press[7] == 0) numberPressed++;
      press[7]= 1;
      SendKeyPress(0x12,1,numberPressed);     // Sends 'o'
keypress

    } else if(_th3 && press[7]== 1) {
      if(press[7] == 1) numberPressed--;
      press[7]= 0;
      SendKeyPress(0x12,0,numberPressed);
    }else
    {
      numberPressed = numberPressed;
    }
    if(!_th4 && press[8]== 0) {
      if(press[8] == 0) numberPressed++;
      press[8]= 1;
      SendKeyPress(0x13,1, numberPressed);     // Sends 'p'
keypress

    } else if(_th4 && press[8]== 1) {
      if(press[8] == 1) numberPressed--;           press[8]= 0;
      SendKeyPress(0x13,0,numberPressed);
    }else
    {
      numberPressed = numberPressed;
    }

    // PINKY BUTTON


    if(!_pi1 && press[9]== 0) {
      if(press[9] == 0) numberPressed++;
      press[9]= 1;
      //SendKeyPress(0x2f,0);     // Sends '[' keypress
      SendKeyPress(0x2c,1,numberPressed);     // Sends 'space'
keypress
    } else if(_pi1 && press[9]== 1) {
      if(press[9] == 1) numberPressed--;
      press[9]= 0;
      Thread::wait(17);
      SendKeyPress(0x2c,0, numberPressed);
    }else
    {
      numberPressed = numberPressed;
    }

    // PALM D-PAD BUTTONS (6 of them)


    if(!_up && press[10]== 0) {
      if(press[10] == 0) numberPressed++;
```

```
         press[10]= 1;
         SendKeyPress(0x52,1,numberPressed);     // Sends up arrow keypress

      } else if(_up && press[10]== 1) {
        if(press[10] == 1) numberPressed--;
         press[10]= 0;
         SendKeyPress(0x52,1,numberPressed);
      }else
      {
         numberPressed = numberPressed;
      }

      if(!_left && press[11]== 0) {
         if(press[11] == 0) numberPressed++;
         press[11]= 1;
         SendKeyPress(0x50,1,numberPressed);     // Sends left arrow
keypress

      } else if(_left && press[11]== 1) {
         if(press[11] == 1) numberPressed--;
         press[11]= 0;
         SendKeyPress(0x50,0,numberPressed);
      }else
      {
         numberPressed = numberPressed;
      }



      if(!_down && press[12]== 0) {
         if(press[12] == 0) numberPressed++;
         press[12]= 1;
         SendKeyPress(0x51,1,numberPressed);     // Sends down arrow keypress

      } else if(_down && press[12]== 1) {
         if(press[12] == 1) numberPressed--;
         press[12]= 0;
         SendKeyPress(0x51,0,numberPressed);
      }else
      {
```

```
    numberPressed = numberPressed;
}

if(!_right && press[13]== 0) {
    if(press[13] == 0) numberPressed++;
    press[13]= 1;
    SendKeyPress(0x4f,1,numberPressed);     // Sends right arrow keypress

} else if(_right && press[13]== 1) {
    if(press[13] == 1) numberPressed--;
    press[13]= 0;
    SendKeyPress(0x4f,0,numberPressed);
}else
{
    numberPressed = numberPressed;
}

if(!_sel && press[14]== 0) {
    if(press[14] == 0) numberPressed++;
    press[14]= 1;
    SendKeyPress(0x1d,1,numberPressed);     // Sends 'z' keypress


} else if(_sel && press[14]== 1) {
    if(press[14] == 1) numberPressed--;
    press[14]= 0;
    SendKeyPress(0x1d,0,numberPressed);
}else
{
    numberPressed = numberPressed;
}

if(!_start && press[15]== 0) {
   if(press[15] == 0) numberPressed++;
    press[15]= 1;
    SendKeyPress(0x1b,1,numberPressed);     // Sends 'x' keypress

} else if(_start && press[15]== 1) {
    if(press[15] == 1) numberPressed--;
    press[15]= 0;
    SendKeyPress(0x1b,0,numberPressed);
}else
```

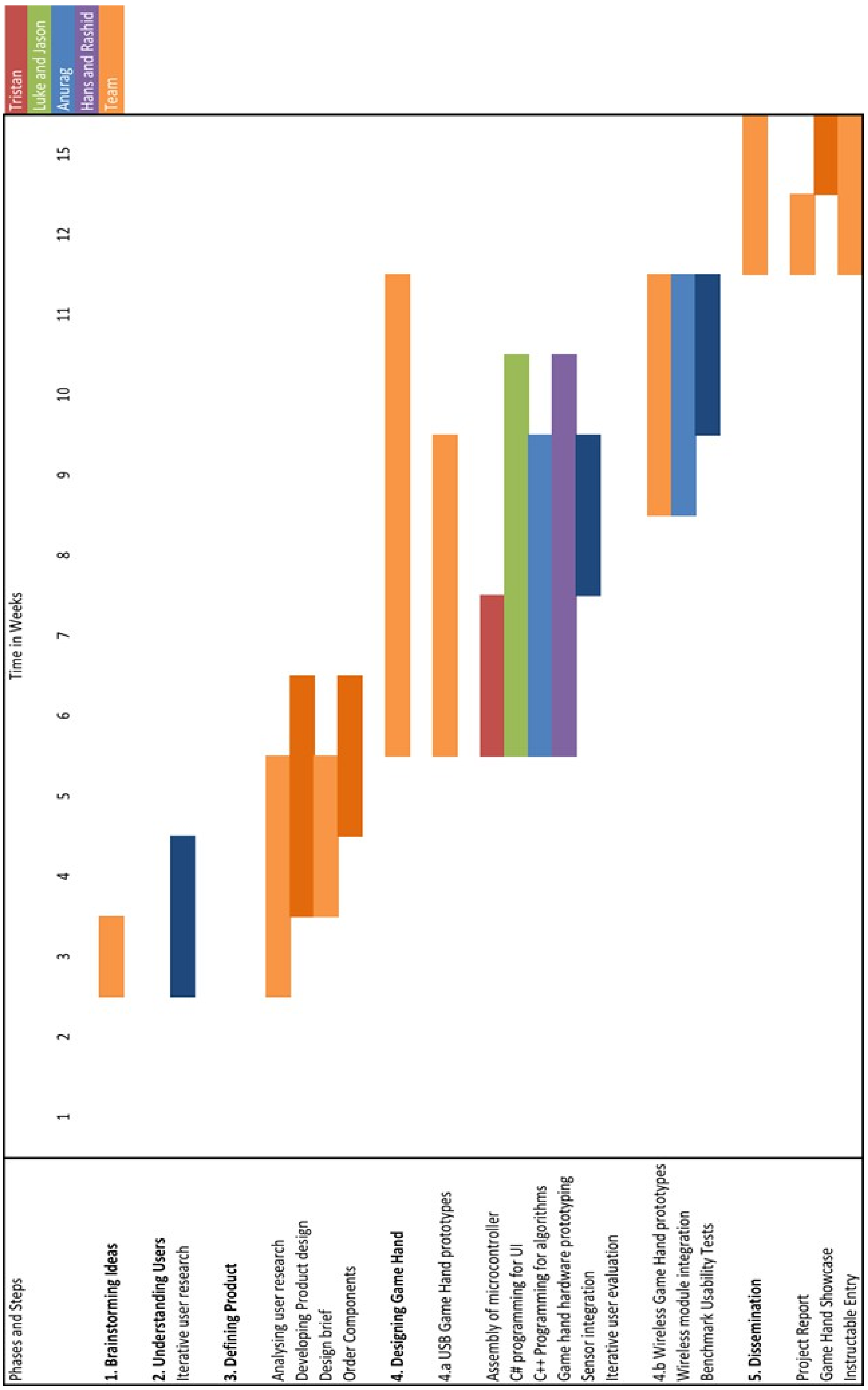
```
      {          numberPressed = numberPressed;
      }

  Thread::wait(100);  // Slows the whole thread down. Without this keys stick!

  }
}


int main()
{
  Thread click(click_thread);
  Thread mouse(mouse_thread);
  Thread keyboard(keyboard_thread);


  led = 1.0;

  while (true) {
  }
}
```

Appendix E – Initial Glove Design (HF)



Glove
Sensor inbetween Velcro Strap
Heat Shrink added Safety
Wires to Circuit
Not adhesive / Self adhesive
Velcro loop Strap
Stiched on either side
Excess Strap
1cm
Sensor Contact Area
Sensor Area Carved out of the tape.
4cm
Sensor Length 3cm
Knuckle Position
1cm
2.5cm
T
1cm
3cm
P.
1cm
5cm
FF
1cm
5cm
SF
1cm
4.5cm
TF
Sensor Composition.
Transparent film for flexibility.
Solder Joints to copper tape.
All layers glued
Velostatic bag
Copper tape.
Glove Design. Version 1.0 26/10/2013.

Appendix F - Diagrammatic time plan of project tasks (AD)



Appendix G - Python Code for graphical accelerometer data logger (AD)

```
import serial
import numpy as np
from matplotlib import pyplot as plt

#setting up connection to serial port
ser = serial.Serial('/dev/tty.usbmodemfd122', 9600)

# set plot to animated
plt.ion()
xdata = [0] * 10
ydata = [0] * 10
zdata = [0] * 10

ax1=plt.axes()

# make plot
t = np.arange(len(ydata))
linex, liney, linez = plt.plot(t, xdata, 'r--', t, ydata, 'b--', t, zdata, 'g--')
plt.ylim([-5,5])                    # set the y-range to -5 to 5

# start data collection
while True:
        data = ser.readline().rstrip() # read data from serial
                    # port and strip line ending

        data_split = data.split(",")
        xdata.append(data_split[0])
        ydata.append(data_split[1])
        zdata.append(data_split[2])

        del xdata[0]
        del ydata[0]
        del zdata[0]
        plt.plot()

        #line.set_xdata(np.arange(len(ydata)))
        linex.set_ydata(xdata)
        liney.set_ydata(ydata)
```

```
    linez.set_ydata(zdata)        # update the data
    plt.draw()                    # update the plot
```

Appendix H - Pre Prototype Survey (JB)

Please tick one of the following

Left Handed	1
Right Handed	9

I am mostly a:

PC Gamer	4
Xbox Gamer	5
Playstation Gamer	2
Other:	1

Favourite Game Genres:

FPS	4
Racing	1
RPG	3
Platformer	2
Retro Gamer	1
I only play Metal Gear Solid 3	
Don't pigeonhole me	1

Please answer the following as truthfully as OJ on Oprah

Given the brief outline of the Super Hans:

	Strongly Disagree	Somewhat Disagree	Neutral	Somewh at Agree	Strongl y Agree
The learning curve seems too steep	1	2	3	3	
My wrist problems would render it useless	6	2	2		
It seems too much like a glorified wii remote		6	3	1	
I would want to see it compatible with Windows, Xbox and PS				3	6
I don't mind having to use my opposing hand to operate a palm d-pad	1	2	3	3	1
It seems like it could be too heavy on my hand	2	4	3	1	
I would not buy a wired version		1	3	2	4
I would like to be a beta tester for the Super Hans		1	2	1	6

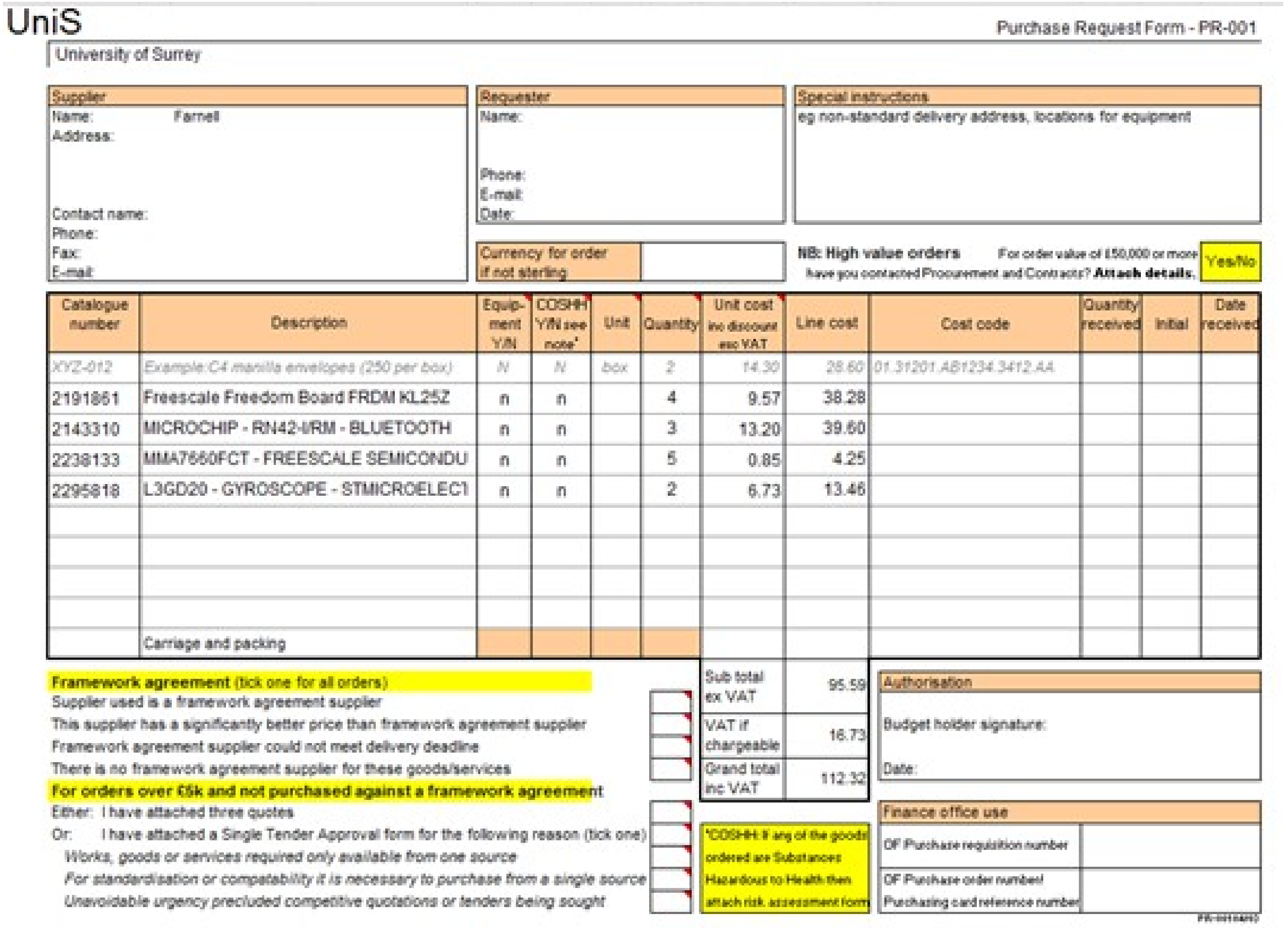
Any other comments?

If you do want to be a beta tester, please provide your
contact details:
Name:
Email:

Survey Notes

- Respondants were first given a verbal overview of how Superhans was to operate
- A total of 10 respondants filled out the survey
- Comments and names/email addresses are not disclosed

Appendix I - Bill of materials (TH)



UniS

University of Surrey

Purchase Request Form - PR-001

Supplier	Requester	Special instructions
Name: Farnell	Name:	eg non-standard delivery address, locations for equipment
Address:	Phone:	
Contact name:	E-mail:	
Phone:	Date:	
Fax:	Currency for order if not sterling	NB: High value orders For order value of £50,000 or more have you contacted Procurement and Contracts? **Attach details.** Yes/No
E-mail:		

Catalogue number	Description	Equip-ment Y/N	COSHH Y/N see note*	Unit	Quantity	Unit cost inc discount exc VAT	Line cost	Cost code	Quantity received	Initial	Date received
XYZ-012	*Example:C4 manilla envelopes (250 per box)*	*N*	*N*	*box*	*2*	*14.30*	*28.60*	*01.31201.AB1234.3412.AA*			
2191861	Freescale Freedom Board FRDM KL25Z	n	n		4	9.57	38.28				
2143310	MICROCHIP - RN42-I/RM - BLUETOOTH	n	n		3	13.20	39.60				
2238133	MMA7660FCT - FREESCALE SEMICONDU	n	n		5	0.85	4.25				
2295818	L3GD20 - GYROSCOPE - STMICROELECT	n	n		2	6.73	13.46				
	Carriage and packing										

Sub total ex VAT	95.59
VAT if chargeable	16.73
Grand total inc VAT	112.32

Framework agreement (tick one for all orders)

Supplier used is a framework agreement supplier

This supplier has a significantly better price than framework agreement supplier

Framework agreement supplier could not meet delivery deadline

There is no framework agreement supplier for these goods/services

For orders over £5k and not purchased against a framework agreement

Either: I have attached three quotes

Or: I have attached a Single Tender Approval form for the following reason (tick one)

Works, goods or services required only available from one source

For standardisation or compatability it is necessary to purchase from a single source

Unavoidable urgency precluded competitive quotations or tenders being sought

*COSHH: If any of the goods ordered are Substances Hazardous to Health then attach risk assessment form

Authorisation

Budget holder signature:

Date:

Finance office use

OF Purchase requisition number	
OF Purchase order number/ Purchasing card reference number	

Appendix J - RN42 Library (TH)

RN42.h

```
#include "mbed.h"

short RN42_init(void);
short RN42_reset(void);
short RN42_SPP(void);
short RN42_HID(void);
short RN42_key(char key);
short RN42_mouse(signed char x,signed char y);
short RN42_click(char l,char r,char m);
short RN42_specaial_key(char key);
void RN42_connect(void);

#define INSERT_KEY 0x49
#define HOME_KEY 0x4A
#define PAGE_UP_KEY 0x4B
#define DELETE_KEY 0x4C
#define END_KEY 0x4D
#define PAGE_DOWN_KEY 0x4E
#define RIGHT_ARROW_KEY 0x07
#define BACKSPACE_KEY 0x2A
#define TAB_KEY 0x2B
#define ENTER_KEY 0x28
#define LEFT_ARROW_KEY 0x0B
#define DOWN_ARROW_KEY 0x0C
#define UP_ARROW_KEY 0x0E
#define F1_KEY 0x3A
#define F2_KEY 0x3B
#define F3_KEY 0x3C
#define F4_KEY 0x3D
#define F5_KEY 0x3E
#define F6_KEY 0x3F
#define F7_KEY 0x40
#define F8_KEY 0x41
#define F9_KEY 0x42
#define F10_KEY 0x43
#define F11_KEY 0x44
#define F12_KEY 0x45
#define ESCAPE_KEY 0x29
#define CAPS_LOCK_KEY 0x39
#define SCROLL_LOCK_KEY 0x47
#define BREAK_PAUSE_KEY 0x48
#define NUM_LOCK_KEY 0x53
#define LEFT_CTRL_KEY 0xE0
#define LEFT_SHIFT_KEY 0xE1
#define LEFT_ALT_KEY 0xE2
#define RIGHT_CTRL_KEY 0xE4
#define RIGHT_SHIFT_KEY 0xE5
#define RIGHT_ALT_KEY 0xE6
```

RN42.cpp

```
#include "RN42.h"
//#include "C12832_lcd.h"

//Serial pc2(USBTX, USBRX);
Serial RN42(PTD3,PTD2);
DigitalOut RN42_reset_pin(PTC2);
Serial pc2(USBTX, USBRX);
//C12832_LCD lcd2;

short RN42_init(void)
{
    RN42.baud(115200);
    pc2.baud(115200);
    //lcd2.printf("starting reset");
    return RN42_reset();

}
short RN42_reset(void)
{
    char buff[3];
    short i=0;

    RN42_reset_pin = 0;
    wait_ms(500);
    RN42_reset_pin = 1;    wait(2);
    RN42.printf("SH,0232\n");
    while(RN42.getc() != 75);
    while(RN42.getc() != 10);
    RN42.printf("R,1\n");
    while(RN42.getc() != 33);
    while(RN42.getc() != 10);
    wait_ms(500);
    /*RN42.printf("$$$");
      for(i=0; i<3; i++) {
         buff[i] = RN42.getc();
      }
      //pc2.printf("buff2 = %c%c%c",buff[0],buff[1],buff[2]);
      if((buff[0] == 67) && (buff[1] == 77) && (buff[2] == 68)) {
         RN42.printf("SH,0231\n");
         while(RN42.getc() != 75);
         while(RN42.getc() != 10);
         RN42.printf("---\n");
         while(RN42.getc() != 68);
         while(RN42.getc() != 10);*/
         return 1;
            // }
        }

    }
    return 0;
}
short RN42_key(char key)
{
    RN42.putc(key);
    return 1;
}
short RN42_mouse(signed char x,signed char y)
{
    RN42.printf("%.2c%.2c%.2c%.2c%.2c%.2c%.2c", 0xFD, 0x5, 0x2, 0x00, x, y, 0x0
0);
    return 1;
}
short RN42_click(char l,char m,char r)
{
    char temp;

    temp = (r << 2) | (l << 1) | m;
    //pc2.printf("temp val=%.2x\n",temp);
    RN42.printf("%.2c%.2c%.2c%.2c%.2c%.2c%.2c", 0xFD, 0x5, 0x2, temp, 0x00, 0x0
0, 0x00);

    return 1;
}
short RN42_specaial_key(char key)
{
    RN42.printf("%.2c%.2c%.2c%.2c", 0xFE, 0x1, 0x0, key);
    pc2.printf("%.2x %.2x %.2x %.2x", 0xFE, 0x1, 0x0, key);
    wait_ms(500);
    RN42.printf("%.2c%.2c", 0xFE, 0x0);//0xfe
    return 1;
}

void RN42_connect(void)
{
    char buff[3];
    short i=0;
     pc2.printf("start\n");
        RN42.printf("$$$");
        for(i=0; i<3; i++) {
            buff[i] = RN42.getc();
        }
        RN42.printf("C\n");
         pc2.printf("searching\n");
        wait(4);
        while(RN42.readable())
        RN42.getc();

        pc2.printf("initiate connection\n");

        ///RN42.printf("CFI\n");
        //wait(5);
       // RN42.printf("---\n");
        //while(RN42.getc() != 68);
        //while(RN42.getc() != 10);
        //pc2.printf("exited cmd\n");
}
```