WEAR A MOUSE

By

Asra Abid Siddiqui

Noor ul ain Ali

Memoona Tahira

Maham Liaqat

Submitted to the Faculty of Electrical Engineering

National University of Sciences and Technology, Islamabad in partial fulfillment

for the requirements of a B.E. Degree in Teleccom Engineering

JUNE 2014

ABSTRACT

Our Project ‘Wear a Mouse’ is a wireless gesture-based input device that aims to replace the traditional computer mouse. It comprises of two separate units namely the ‘glove’ and the ‘base station’. The purpose of the glove is to take and process user input from an accelerometer and sensors and send it over the wireless medium to the base station. The purpose of the base station is to receive the data packets sent by the glove, convert them to HID format understandable by the computer and forward the data to the computer through a serial interface. The data then translates into mouse actions on the computer screen. These actions are cursor movements, scrolling, left and right clicking along with other specific functions. Each unit uses a different microcontroller and respective peripheral devices.

These include an accelerometer for taking user input, two different microcontrollers for coding and decoding of user input, a set of radio transceivers for wireless communication between the microcontrollers and a microcontroller development board specific to USB based projects.

CERTIFICATE OF CORRECTNESS AND APPROVAL

It is certified that the work contained in this thesis title “Wear A Mouse” , carried out by Asra Abid Siddiqui, Noor ul ain Ali, Memoona Tahira and Maham Liaqat under the supervision of Lt. Col. Dr. Adnan Ahmed Khan in partrial fulfillment of Degree of Bachelour of Telecommunication Engineering, is correct and approved.

Approved by

_____

LT. COL. DR. ADNAN AHMED KHAN

PROJECT SUPERVISOR

Military College Of Signals, NUST

DECLARATION

No portion of the work presented in this dissertation has been submitted in

Support of another award or qualification either at this institution or elsewhere.

DEDICATION

In the name of Allah, the Most Merciful, the Most Beneficent

To our parents, without whose unflinching support and cooperation,

a work of this magnitude would not have been possible.

ACKNOWLEDGEMENTS

There is no success without the will of ALLAH Almighty. We are grateful to ALLAH, who has given us guidance, strength and enabled us to accomplish this task. Whatever we have achieved, we owe it to Him, in totality. We are also grateful to our parents and family and well-wishers for their admirable support and their critical reviews. We would thank our supervisor Lt. Col. Dr. Adnan Ahmed Khan for his guidance and support throughout the year. We would specially like to thank Lecturer Intisar Rizwan e Haque for his commendable guidance through each and every phase of our project. We would also like to thank our fellow student Ali Tanveer for helping us when we were critically stuck. Further we will thank our juniors from Fatima Coy especially Maheen and Amna for their constant support and motivation. The team would like to pay gratitude to the team leader for bearing us the whole year and for billion repeated document edits and lastly Capt Saad Munir for always being there for helping us whenever needed.

Table of Contents

LIST OF FIGURES:

LIST OF TABLES:

LIST OF ABBREVIATIONS:

ADC	Analog to digital convertor
MCU	Microcontroller Unit
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous/asynchronous receiver/transmitter
USB	Universal Serial Bus
HID	Human interface device

CHAPTER ONE

1 INTRODUCTION

1.1 OVERVIEW

In terms of functional level entities, our project mainly consists of two parts: a glove and a base station. The Glove unit would be a standard sized glove (made of Velcro or similar material) resembling any regular glove. Operation of the device relies on the input obtained by a user-worn glove. A user can use hand tilt orientation and finger presses to operate the glove. Two types of sensing methods are used to take input from the user: an accelerometer and finger contact pads.

Both the glove and the base station are fitted with microcontrollers that process data at each end. After the glove's microcontroller manipulates the input data, it forwards the processed data, in the form of a data packet, to a transceiver present on the glove unit worn by the user. The transceiver then transmits this packet wirelessly to its brother transceiver on the base station. The receiving transceiver forwards the message to the base station microcontroller. The data from the packet is extracted by the base station microcontroller, converted to a computer understandable HID format, and then sent over a USB link to the computer to perform the user's command.

1.2 PROBLEM STATEMENT

Previously, attempts made to replace the traditional mouse have focused on implementing a gesture-controlled interface using cameras or light sensors. Such systems translate user

movements in to mouse movements through gesture recognition by employing complicated methods that involve intensive computing (such as image processing techniques) and require the establishment of a database from where recorded gestures are matched to actions.

Main concern was to design a device that performs the same actions with half the computing effort required. Other devices which have also tried this approach and employed simpler techniques to get similar results have shown flaws in terms of sensitivity and accuracy. Some are limited by their line-of-sight restrictions.

Wear a mouse will help to overcome these problems. It will be a very accurate pointing device with the right amount of sensitivity and no need of any large database. Users will be able to operate their computers with their hands in midair without the hassle of desks or hard surfaces or wires and would not need to keep their hands in line of sight of any sensor.

It will allow greater freedom of movement to the user and the computer end of device will connect to a computer using standard USB port.

1.3 APPROACH

1. Hand tilt orientation and finger presses to operate the glove

2. Accelerometer and finger contact pad sense the orientation and direction of tilt of hand

3. The glove's microcontroller processes the input data.

4. Based on it, forwards a message a transceiver mounted on the glove unit which transmits this message wirelessly to a transceiver on the base station.

5. The receiving transceiver forwards the message base station microcontroller.

6. Finally, the microcontroller converts the message into a computer HID user friendly format and moves the computer cursor appropriately.

7. The data is read though USB HID port.

8. Cursor moves on screen

1.4 SCOPE

Our project will provide and interactive and intuitive control of the computer mouse to the user. The user can control the cursor without any hassle of a desk or a hard surface. He can perform the movements as well as the mouse clicks in midair and that too from a considerable distance from the computer itself.

1.5 OBJECTIVES

1. Design and build a wireless computer-pointing device with accelerometer based movement control.

2. Implementation of a USB-HID device.

3. Building a control and base station to transmit data wirelessly featuring low power radio transceivers using frequency hopping spread spectrum

1.6 ORGANIZATION

The first part of thesis is the abstract which describes the main details of our project 'Wear a Mouse', followed by the introduction section which specifies the problem

statement, approach, scope and objectives. The literature review section state the various resources read online before the commencement of the project. They include learning about diseases and syndromes caused by excessive use of a common mouse. The design and development part illustrate the diagrams which describe the detailed design of the Wear a Mouse, its components, interfaces and data necessary for the implementation phase. The analysis and evaluation part is the detail of the testing the actual results against expected results. The future work gives states the enhancements that can be applied to the application.

CHAPTER TWO

2 LITERATURE REVIEW

As the basis of our project was movement sensing devices and how they function, we had to do a great deal of research on the different methods that can be used in this regard.

Human-Computer Interaction has been moving towards more intuitive and natural interfaces, as opposed to the classical keyboard and mouse. One of the main techniques that can be and have been used to achieve this end are accelerometer-based devices.

2.1 ACCELEROMETERS

An electromechanical device used to measure accelerative forces is called an accelerometer. These forces can be static, such as the constant gravitational force or, they can be dynamic in nature for sensing vibrations and movements.

The angle of tilt of a device with respect to the earth can be found out by measuring the amount of static acceleration due to gravity. The movement of a device can be analyzed by observing the degree of dynamic acceleration of the device**.**

A dynamic accelerometer determines the angle at which a device is tilted with reference to the Earth by measuring the gravitational pull acting on it. By detecting the amount of acceleration, users can analyze the nature/direction of movement of the device.

The type of accelerometer we used in our project involves the use of capacitance sensors. This type of accelerometer contains microstructures. It detects changes in the capacitance between these miniscule structures as it is moved in space. Whenever an accelerative force is applied that causes movement of one of the structures, a corresponding change

will occur in the capacitance and this variation will be translated by the accelerometer in to voltage which is then interpreted.

A particular type of accelerometer called a three axis accelerometer measures acceleration along three axes (in 3 Dimensions). The acceleration in each direction is represented by a different voltage, and each voltage is in turn are represented by data vectors. Theses vectors are analyzed to obtain values of tilt along each axis. Filtering is done by simple application of a threshold, which eliminates all vectors that do not meet the minimum tilt limit for each axis. The tilt values are modeled and manipulated to give cursor movement.

Figure 2-1 How the accelerometer's voltage data is digitized.

Since our project aims to design a glove based mouse device, we also did a detailed research of similar projects and products. Previous attempts made at commercial level in regard to producing such a device include:

2.2 ION WIRELESS AIR MOUSE:

The Wireless Air Mouse is in the form of a snug-fitting spandex glove. The mouse uses a 2.4GHz signal to connect wirelessly to the computer through a USB 2.0 interface, and has a range of about 35 foot. On the fingertip button attachment the user has three small buttons, two of which are for left and right clicking and one is used to pause motion

control. Employing these buttons gives the feels less like clicking a mouse and more like pinching something between thumb and forefinger.

Once turned on, the glove serves as a functional air mouse, meaning that users control cursor by moving the mouse in the corresponding direction—up for up, down for down. The mouse is limited to right and left clicking only. The default sensitivity setting is of 1200 dpi, so the mouse seems to be display a little bit of lag when it is initially plugged in. A normal mouse does not track user's every fidget, but this one does, and because of that a bit of a learning curve is involved as users take time to grow accustomed to keeping their hand very steady.

2.3 MISTER GLOVES:

Mister Gloves is a wireless USB gesture input system that enables a person to use a computer by performing intuitive hand and finger motions in the air.

While wearing a glove controller on the right hand, clicking can be performed by curling one's thumb or index finger and the cursor can be moved by imitating a pointing gesture. Other gestures of the right hand can be used for enabling scrolling and for providing access to several different types of keyboard shortcuts. The two (right and left) controllers are connected to each other but for wireless communication with the computer they connect to it via a base station. The device employs a low-speed interface namely USB 1.1 and there is no need for any extra drivers to be installed.

2.4 **PLUG AND PLAY**:

Our main concern while designing our device was to make it such as to allow users maximum ease of use and minimum hassle. One of the key points in this regard was to come up with a method which would make our device a PnP (Plug and Play) device.

Plug and Play (PnP) is a technology through which a computer's operating system is able to sense internal and external peripheral devices and adapters and configure them accordingly. Hardware components can then be found and configured by the computer without the need to reset any jumpers or toggle any DIP switches. PnP is also used to refer to devices such as USB sticks.

When a PC is started up, identification of the different peripherals attached is carried out by PnP. It regulates the corresponding internal settings by performing configuration of the interrupt requests (IRQ) , the direct memory access (DMA) and the input/output (I/O) addresses.

Basically, being PnP would mean the user wouldn't have to do any complicated configuration of settings or installation of any additional drivers to use our device. He would just plug in the device and start using it.

2.5 HID FORMAT

In addition to this we needed to make our device conform to the HID standard so that the data inputted to it could be understood by the computer as mouse movement commands.

A human interface device (HID) is a method by which a human interacts with an electronic information system by either inputting data or providing output. HID achieves a standardization that covers the exchange and connection of data between a device and

the host computer. The user is unaware of the details involved in this since they're hidden from him. They play a role in facilitating the implementation of the device and play an instrumental role in fast innovation and proliferation of HIDs.

The term "HID" most commonly refers to the USB-HID specification. The term USB-HID refers to a class of devices that are human interface devices and connect to the computer via a USB port. Many predefined functions exist in the USB HID class. The purpose of these functions is to give hardware manufacturers the freedom to design a product according to the USB HID class specifications and expect it to work with any software that also meets these specifications.

The main underlying purpose of the HID class definition is to:

- Save data space of device by being as compact as possible.
- Allow software applications to leave out any unwanted and unknown information.
- Be robust and extensible.
- Allow for generic software applications by being self-describing.

A USB/HID class device makes use of corresponding HID class drivers to collect and route all information. The collection and routing of information is done by analyzing the device descriptor and the information it provides.

The purpose of the HID class device descriptor is to indicate the presence of other HID class and give information about their sizes. For example, the function of a 'Report'

descriptor is to give a description of each individual piece of information generated by the device and what this information is actually signifying or measuring.

For example, the purpose of the Report descriptor is to identify items that give a description of states of a button or information about positions. Data about items is then used for:

- Determining where the input information should be sent—for example, route the input data to the API of the mouse or joystick.
- Allowing the software/application to determine what functions the input data is supposed to perform—for example, use the input of the joystick to cause movement of the cursor.

Keeping this in mind we had to find a microcontroller that could offer us in-built functionality that would encompass all our design requirements while at the same time help us in implementing a USB-HID device so we could focus on the details of the user end of the project.

CHAPTER THREE

3 DESIGN AND DEVELOPMENT

3.1 DESIGN REQUIREMENTS

Our project requires an accelerometer which measures the analog hand tilts and movements in terms of analog voltages and passes this information to the microcontroller which will further process the information. For this purpose we chose MMA7361 accelerometer because it suited our requirements which are listed below in the table 2. The table 3-1 also gives a comparison between our chosen accelerometer and other similar accelerometer available in the market.

Table 3-1 : Features of accelerometers compared

Sr. no	FEATURES	MMA7361	ADXL345
1	Dynamic and Static acceleration	YES	YES
2	Low drop out voltage regulator	YES	NO
3	Low current consumption	YES	YES
4	Minimum clock noise	YES	NO
5	Analog output	YES	YES
6	Cost	PKR 1200	PKR 3000

The features provided and economical cost of MMA7361 made us chose this accelerometer module.

After that we are required to convert these analog voltages to digital voltages so that we can change these digital voltages into packet form that would be transmitted by the wireless transceivers to the other end. We also require that the Glove Unit's MCU is connected to the input of a wireless transceiver. Specifically, it is connected to port D which uses the USART channels supported by the Atmega1284.The following table

consist of a comparison between Atmega1284p and commercially available and widely deployed ADC.

Table 3-2 : Comparison between Atmega1284p and ADC10D1500

Sr. no	FEATURES	ATMEGA1284P	ADC10D1500
1	10 bit analog to digital conversion	YES	YES
2	Programmable serial USART channels	YES	NO
3	Locally available	YES	NO
4	Self-programmable Flash program memory	YES	NO
5	Cost	PKR 1400	PKR 13000

We chose Atmega1284p because most importantly it supports USART channel and is cost effective. Next is our wireless transceiver, since we want mobility for our mouse user we want a transceiver with maximum range so even at a considerable distance from the screen our mouse works properly.

Table 3-3 : Comparison between transceivers

Sr. No.	FEATURES	TRM-915-R25	**ADF7242**
1	Range	7 miles	Short Range
2	Compatibility with Teensy ++2.00	Yes	No
3	Frequency range	902-928 MHz	2400-2483.5 MHz
4	UART Interface type	Yes	No
5	Cost	PKR 4000	PKR 3500

We need two of these wireless transceivers; one will be used at the glove side and will work as a transmitter and the other as a receiver at the base station side. We chose TRM 915 R25 because it provides us the maximum range in affordable price and has UART interface type which makes them compatible in use with our micro controllers used at the glove and base station side.

Finally at the base station side after receiving the digital packets we required some processing which converts these packets to mouse cursor movements. The base station

microcontroller receives input from the wireless transceiver's output. Communication is processed via USART in the same fashion as in the glove unit.

Table 3-4 : Comparison of Teensy with Arduino

Sr. no	FEATURES	TEENSY	ARDUINO
1	Additional microcontroller AT90USB1286	Yes	No
2	Extensive licensed code examples that provide access to USB device classes	Yes	No
3	Cost	PKR 2000	PKR 3000
4	Plug and play	Yes	No

The main reason why we chose Teensy was because it uses a microcontroller that can do USB natively without the use of additional circuitry like the Arduino. The second reason we prefer the Teensy platform is that it provides open source USB device classes. The Teensy is an extremely small board that is very well designed. Finally we really prefer the Teensy because hardware programming requires no processing as in case of Arduino.

The Teensy++ 2.0 is microcontroller development board which features an AT90USB1286 microcontroller. We also chose to use a Teensy over the duplicate of the glove MCU for several reasons. Teensy development board with AT90USB1286 has an advantage of USB communication. This way, we wouldn't have to develop additional software for the required USB HID class. Additionally, the Teensy carried an Atmel microcontroller that used very similar specifications to what we were already used to from the Atmega1284. The teensy package is available as very small sized package that has example codes available.

3.2 PROPOSED SYSTEM DIAGRAM:

This figure summarizes the flow of our design:

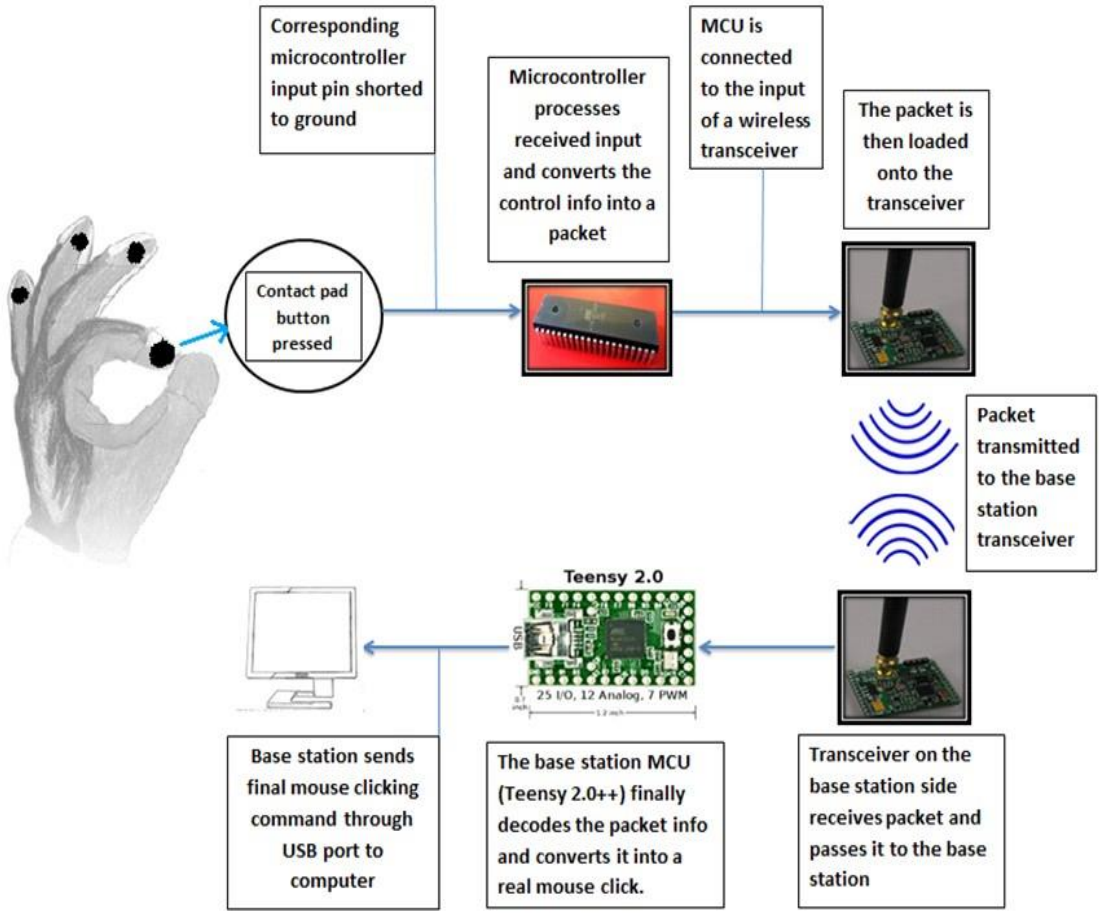


Figure 3-2 : Complete Block Diagram of the Design

First, we will explain the component's part in our design in detail to understand why each component is used:

3.3 ACCELEROMETER WORKING:

In order to detect and then map the user hand motions to cursor movement, we used an accelerometer to measure hand tilt and alignment. We chose to use a device called 3-Axis Accelerometer Module. The module carries MMA7361 three-axis analog accelerometer. The 3-Axis Accelerometer Module has a low-dropout voltage. It takes

3.3V power to turn on the module. The MMA7361 and 10nF capacitors on the outputs of the accelerometer are used to minimize clock noise.

3.4 ACCELEROMETER TESTING:

Problems:

Accelerometer testing: no simulation software (Cadence Virtuoso Spectre, Orcade PSpice, Proteus ISIS 7.7)

Solution:

We used a Google application for understanding the working of accelerometer and then moved on to hardware implementation of accelerometer.

Contact pads are needed in order to detect pointing device button commands. Each contact pad features two output wires that are shorted together when contact pad is pressed. Contact pads will be connected to ground and the other to a 330-Ohm resistor in series with the MCU's input ports.

3.5 MICROCONTROLLER:

We chose Atmega1284p because most importantly it supports USART channel and is cost effective. This high-performance Atmel 8-bit AVR microcontroller combines 128KB ISP flash memory with readwhile-write capabilities, 4KB EEPROM, two USARTs, a byte oriented 2-wire serial interface, an 8channel 10-bit A/D converter. The main part of our project was to write a C code to program the microcontroller to decipher the input from the two input mechanisms, namely the accelerometer and the buttons and switches.

At the end, this microcontroller packages the user's instruction of what to do into an 8-byte packet and forwards it to the transceiver over the serial channel.

Problems:

The main problem we had using this ATMEGA1284p is that it has problem interfacing with its peer microcontroller at the base station side, which is AT90USB1286. According to our study, the interfacing between the microcontrollers was no problem because both the microcontrollers belong to the same ATMEL family.

The pins to which the buttons for clicks and scroll are connected are working properly but the output of the microcontroller gives us less than 0.5 volts that is not enough for the transceiver, connected after the microcontroller, which requires at least 1.4 volts to power up.

Next is our wireless transceiver, since we want mobility for our mouse user we want a transceiver with maximum range so even at a considerable distance from the screen our mouse works properly.

3.6 WIRELESS TRANSCEIVERS:

The glove unit and the base station both support the same transceiver. The glove unit's transceiver receives input from the microcontroller attached on the glove and transmits data to the base station's transceiver, which will then forward the data to the base station microcontroller.

Problems:

1. Simulation of RF transceivers: no simulation software available that can integrate transceivers with our rest of circuit design

2. We require a 50 ohm trace antenna with our transceiver module to work efficiently, other than that the coupling affects degrade the performance of the transceivers severely.

Solution:

The alternative transceivers we considered lacked USART communication with HID friendly format. For these transceivers, the lack of simulation software posed a problem. We tried alternatives such as Cadence Virtuoso Spectre but that did not prove helpful because it did not allow us to simulate the transceivers with the rest of the circuit. We tried to rectify the problem by hardware testing.

Each microcontroller is used to power up its respective transceiver by Vcc output voltage, and has all six of its ground contacts connection to microcontroller ground. The transceivers operate at 902 to 928 MHz. The base station microcontroller receives input to RXD1 (pin D.2) from the wireless transceiver's TXD0 output. Communication is processed via USART in the same fashion as in the glove unit. The wireless transceiver Vcc and ground are connected to MCU Vcc and ground. The MCU is powered via the Teensy's USB mini B plug.

3.7 TEENSY ++2.0:

It features an AT90USB1286 microcontroller. There are many reasons as to why we chose to use a Teensy over the duplicate of the glove MCU. One being that this particular device supports USB communication. If we did not use this feature then we were

required to make additional and extensive software adjustments to our project, which is a project in itself.

Moreover the Teensy contains an Atmel microcontroller i.e. AT90USB1286 which uses very similar specifications to what we have already used for the Atmega1284. Lastly, the Teensy is available as a very simplified, small and lightweight module, user-friendly package with several supporting example code libraries.

3.8 TEENSY TESTING:

We tested teensy++2.0 using software teensy loader application. The results of teensy testing are shown below:

Figure 3-2 : Testing of Teensy++2.0

The above image shows the test program running on the teensy through the teensy loader software. This same software is used to program teensy.

3.9 WORKING DETAILS OF THE DESIGN

The following figure shows the glove unit in detail:

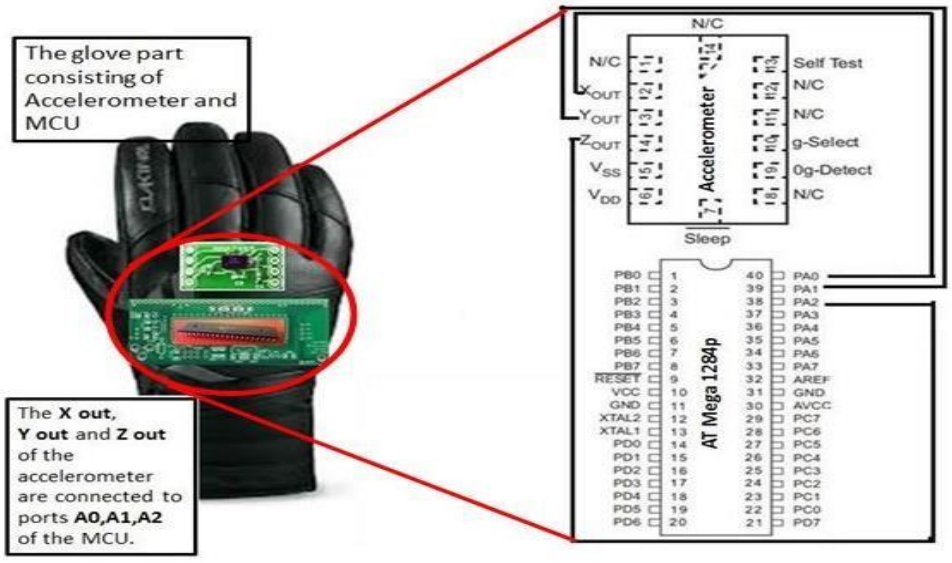


Figure 3-3 : Glove Unit in Detail showing the Accelerometer and AT mega 1284 Microcontroller.

The following figure shows the input pins and ports used by the ATmega1284:

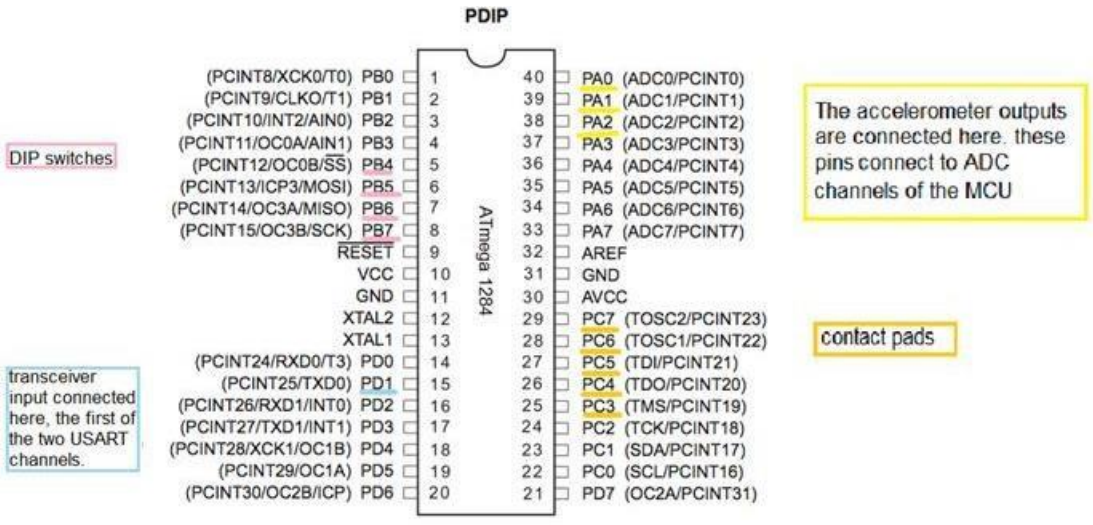


Figure 3-4 : ATmega 1284p Pin Out with used Port/Pins highlighted

The detail of the figure is given in the next paragraphs.

The primary modes of input are hand orientation for mouse movement control and buttons for mouse clicks and control over movement enabling. Hand orientation is sensed by the 3-axis accelerometer in terms of tilt, and outputs an analog voltage to pin A.0, A.1, and A.2 of the glove MCU. These pins are connected to the Atmega1284's internal analog-to-digital convertors, and are converted to a char value between -128 and 127. Since we are using a analog accelerometers that will output a voltage level within a predefined range, we need to convert it to a digital value using an ADC (analog to digital converter) module. This job is done by interfacing them to the ADC channels of the ATmega 1284 which converts the analog values to digital bits (A.0, A.1, A.2).

Button presses are sensed by contact pads placed on the sides and pads of the glove's fingers. The contact pads are connected to pins C.3 through C.7 on one side and MCU ground. Each pad simply shorts the microcontroller's input pin (originally pulled high) to ground. Contact pads are wired in series with a 330-Ohm resistor to protect respective microcontroller ports. When the move or scroll is enabled, a calibrating function is called to read the current tilt values. This function calculates offset values, which are used as the starting reference point. By subtracting those offsets from every ADC sample, the user can start at any position and use that hand orientation as the current reference axes for this gives us the major advantage of not having to return to any default position.

Cursor acceleration reduces lag of the mouse, which is seen as dragging of the cursor behind the user hand movements. To adjust it, we calculate the rate of change of tilt, simple subtraction of the previous value of tilt from current value of tilt would be carried out. At the same time, since we want to avoid any abrupt unwanted burst of movement we will make use of nonlinear scaling to calculate acceleration of the cursor.

Similarly, four DIP switches are wired from ground in series with a 330 Ohm resistor to MCU pins B.4 through B.7 to control mouse user preference features such as sensitivity, axis inversion, and rapid fire clicking.

Moving on to button clicking, we have the right, left and middle click button to perform a click or unclick. When these buttons are pressed and then released, the corresponding flag is toggled. If move enable flag is on, the ADC is sampled and the cursor speed is transmitted to the base station. Similarly, the scroll command is transmitted to the base station only if the scroll enable flag is set. We designed scroll enable to have priority over move enable.

In terms of outputs, the Glove Unit's MCU is connected to the input of a wireless transceiver. Specifically, it is connected to port D.1, the first of two USART channels supported by the Atmega1284.This port is appropriately chosen to interface with the Radiotronix transceiver used, which communicates via USART. To see how it is done, we consider the state machine with 4 states.

3.10 PROTEUS DESIGN:

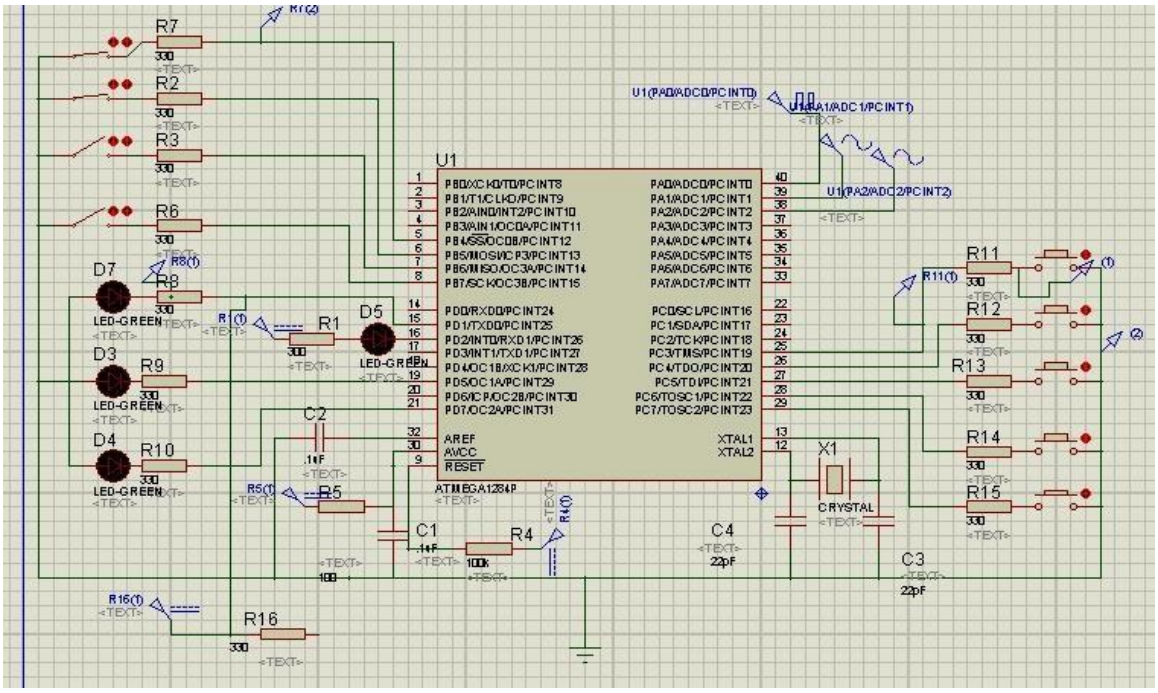


Figure 3-5 : Simulation for Glove side, showing all the inputs (accelerometer, button, DIP switches)

It is the snapshot of our Proteus simulation. We approximated the accelerometer with analog voltage sources since the Proteus component library does not include any accelerometers. The use of analog sources is based on the assumption that data given by the accelerometer for the x,y,z axes readings is analog and variable in nature and outputted as a varying voltage.

At the port B, we added switches to mimic functionality of the user defined function. These switches represent the actual switches that will be used for activating software features like axes inversion, rapid fire and sensitivity settings. In addition, at port C, we placed push buttons to approximate mouse clicks.

The simulation shows the internal resistors are pulled high via programming because whenever a button is pressed, the pin is grounded which is shown as zero by the voltage

probe. Since this is a temporary press, the initial high state of the pin is restored when button is released. The same applies to switches except for high state of pin is only restored when switch is opened.

During the testing phase of our project we came across many problems, one of them being the interfacing between the two microcontrollers.

To eliminate the problem of interfacing between the micro controllers we used the same teensy module on the glove side as well. The circuit diagram with basic connections is as below:

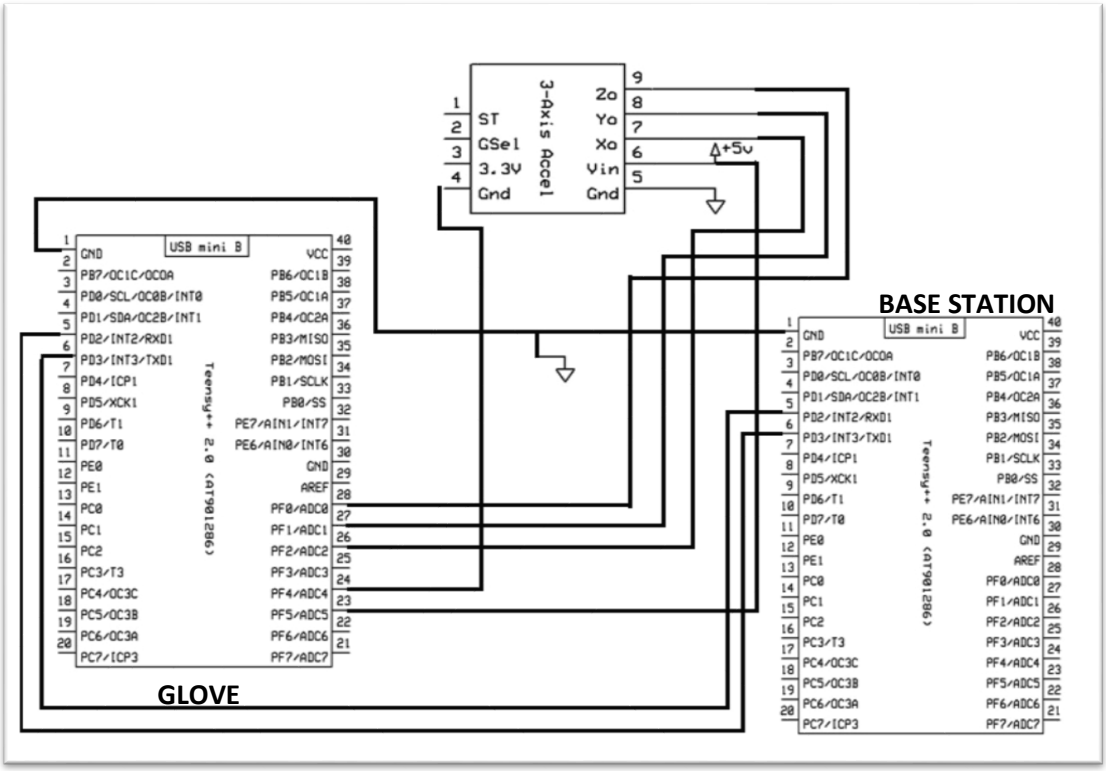


Figure 3-6 : Revised schematic diagram

Both the glove and the base station now contain the teensy 2.0++. The left side is the glove side and we will refer to it as Teensy1.The right side is the base station side to

which we refer as Teensy2. The port F of Teensy1 is now used for analog to digital conversion rather than the ATMEGA1284p. The pin PF0 of Teensy1 is connected to the z axis pin of the accelerometer, PF1 is connected to y axis pin and PF2 is connected to the x axis of the accelerometer. The analog voltages at the output of the accelerometer will come to port F of Teensy1 for conversion to digital voltages. The accelerometer takes its source and ground from the output of the microcontroller on pins PF5 and PF4 respectively.

The following table shows the new connections for the Teensy compared with the previously used AtMEGA1284p.

ATMEGA 1284p	Teensy 2.0++
A0 (z axis)	F0
A1 (y axis)	F1
A2 (x axis)	F2
C7 (left click)	C5
C6 (middle click)	C6
C5 (right click)	C7

Figure 3-7 : Basic connections for Teensy 2.0++

For serial transmission the RXD1 of glove side teensy1 is connected to the TXD1 of the base station side teensy2 and the RXD1 of base station teensy2 is connected to the TXD1 of the glove side Teensy1. The grounds pins, which are pin0 in both the teensy modules are connected together and are given a common ground. The above circuit is finally implemented and tested.

CHAPTER FOUR

4 ANALYSIS AND EVALUATION

The analysis will go through each phase of development step-wise and document what the output at each stage was.The first phase included the testing and interfacing of accelerometer with teensy ++2.0. We needed to know the approximate voltages generated at teensy's x-pin, y-pin and z-pin to map them to a range to calibrate the input. We used an android app to learn about the accelerometer's behavior.

Figure 4-1: Accelerometer testing application

This accelerometer app measures and displays a graph of G-Force vs. Time. It that can be exported in as a .txt attachment with comma separated values that act as logged values. These values are represented in the form of graphs in the apps and can be used in MATLAB to display results. Acceleration and G force are measured along the three axes x, y, and z and the results are displayed individually on the graph. The combined sum of the vectors along each axis is then digitally depicted.

Step 1: The app was downloaded and installed in an android phone. It was then tilted and turned to know how the axis values changes when we moved the mobile in different direction.

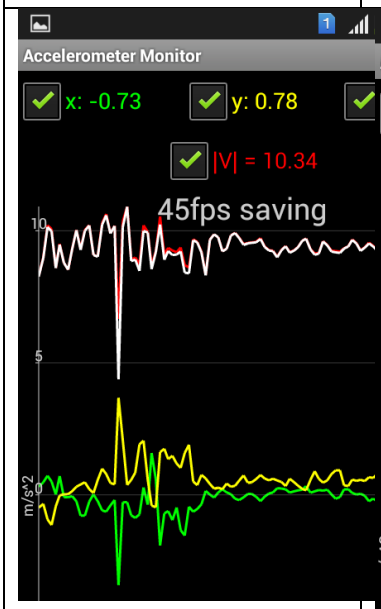
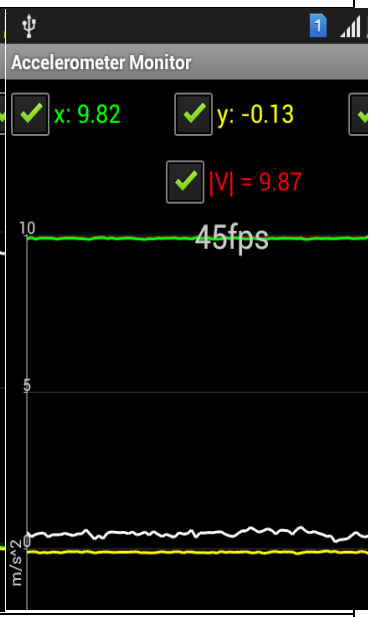
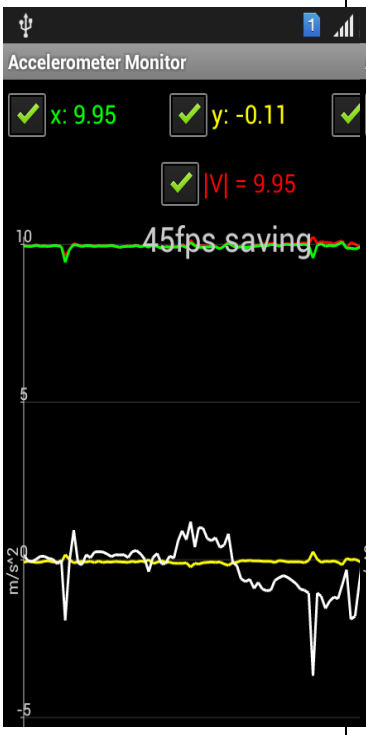
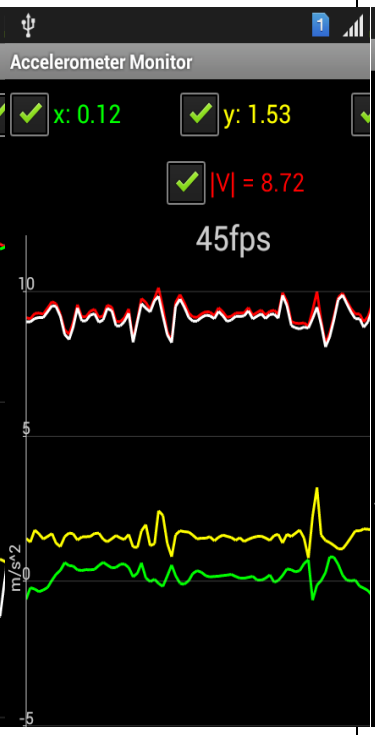
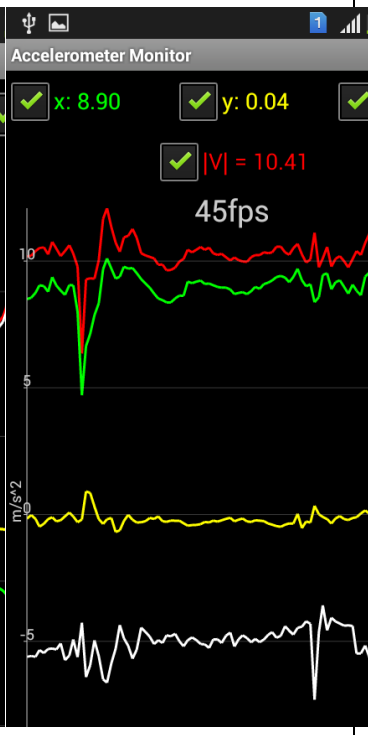
UP-DOWN	HORIZONTAL	VERTICAL
		
TILT	LEFT TO RIGHT	RIGHT-TO-LEFT
		

Figure 4-3: Testing of Accelerometer

Step 2: Some basic orientations were tried out like moving it right, left, upwards, downwards and so on in different intermediate directions. Here are different snapshots showing the change in axis when we moved the accelerometer in different directions.

Step 3: Conclusion

After trying various positions and tilt, a fair idea was gained on how to use the accelerometer in the project by getting to know how the values of different axes would change with user's hand movements.

MATLAB LOG FILE:

Table 4-1: Log file showing voltage values at X-pin, Y-pin and Z-pin for a certain orientation.

7	-0.096	2.099	8.954
8	-0.136	2.25	8.678
9	-0.167	2.331	8.629
10	-0.108	2.303	8.614
11	-0.12	2.285	8.787
12	-0.301	2.347	8.841
13	-0.172	2.383	8.717
14	-0.123	2.365	8.734
15	-0.019	2.134	8.886
16	-0.368	2.413	8.79
17	-0.525	2.594	8.686
18	-0.554	2.631	8.811
19	-0.422	2.446	8.781
20	-0.217	2.238	8.836
21	-0.276	2.282	8.811
22	-0.29	2.288	8.718
23	-0.365	2.445	8.603
24	-0.27	2.452	8.792
25	-0.217	2.401	8.815
26	-0.239	2.4	8.862
27	-0.245	2.371	8.828
28	-0.254	2.325	8.725
29	-0.302	2.393	8.693

After trying various positions and tilt, it was decided to map the voltage to a resolution of 1024, that would be in steps of one from 0 to 1023.

The code for taking the input and calibration of accelerometer was uploaded on teensy 2.0++ and analyzed in the serial monitor of Arduino Software.Figure 4-4 was the result:

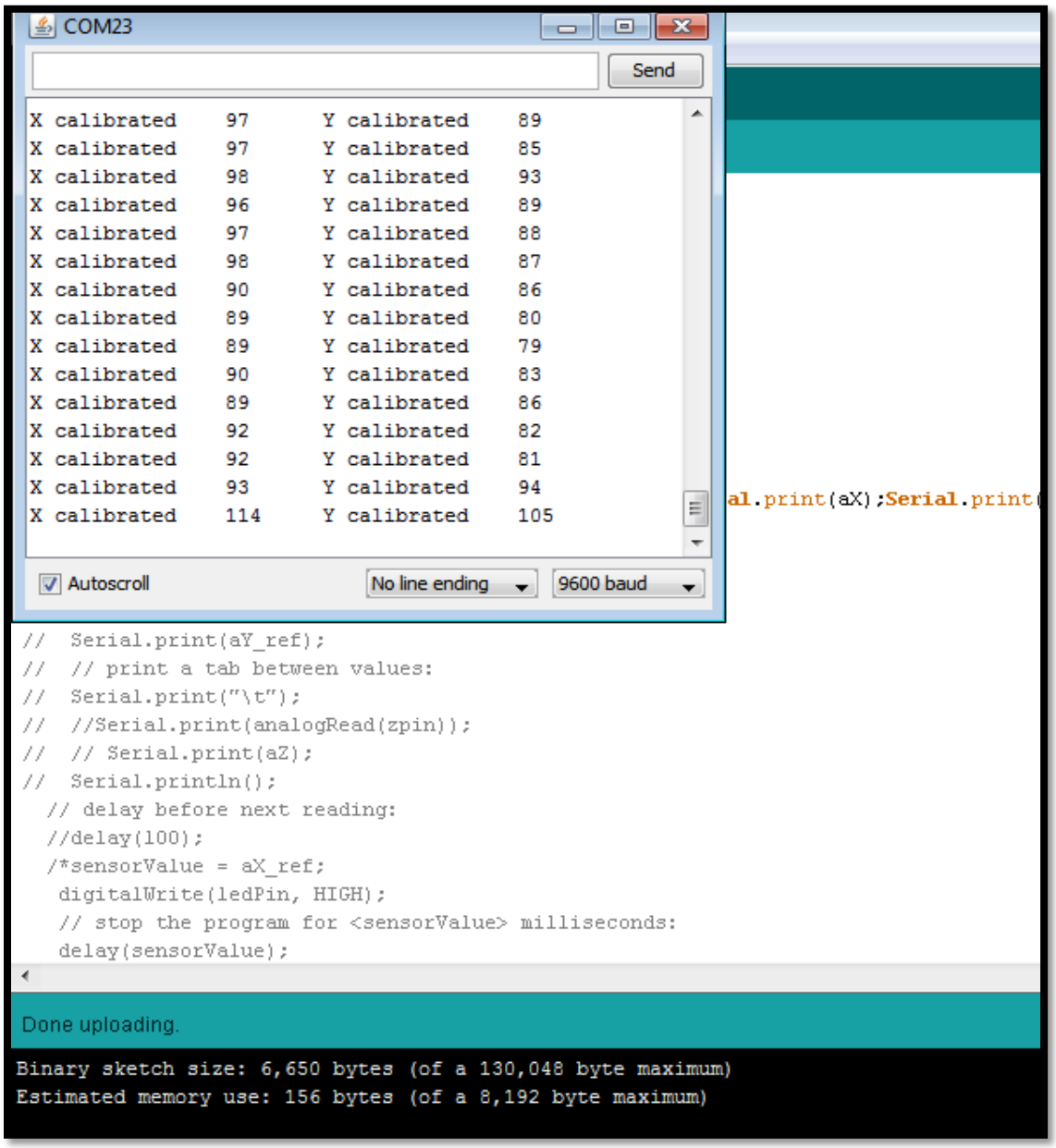


Figure 4-4: Accelerometer calibration Results

This calibrated data was given as input to a function that calculated user's hand movement acceleration and moved the mouse cursor according to that.

For the clicks, the buttons were connected to pins that were pulled internally high, when a button is pressed, the pins went to low state and an input was triggered which translated to a mouse click (left, right, middle) on screen.

The next task was to send the data over the UART of teensy to the base station teensy. The serial communication transmitted the mouse movement and clicking information to the receiver. The mouse cursor moves on screen when the base station teensy ++2.0 is plugged in as USB>type Mouse.

The results of the cursor movement were plotted as a graph on Processing, an Arduino compatible software that graphed the mouse movements happening due to accelerometer movement.

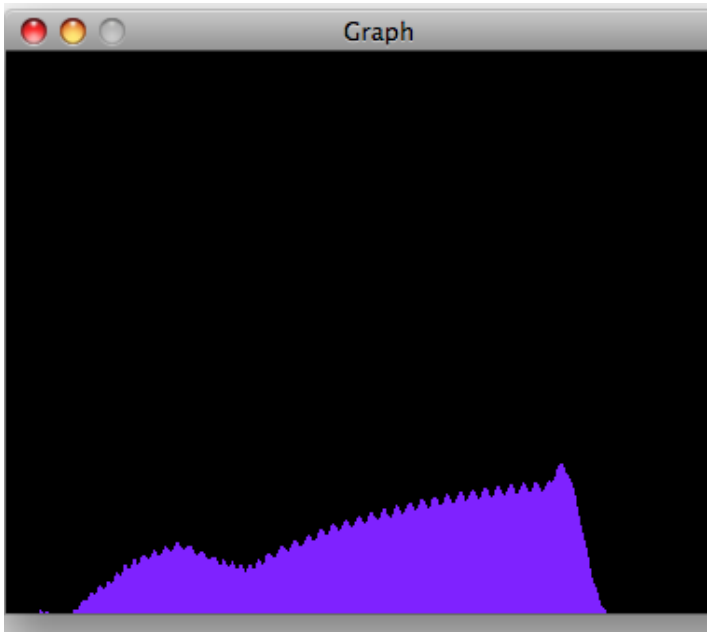


Figure 4-5: The plot of change in cursor movements with change in analog sensor output

Results on Bread Board:

Figure 4-6 shows the assembled circuit on breadboard.

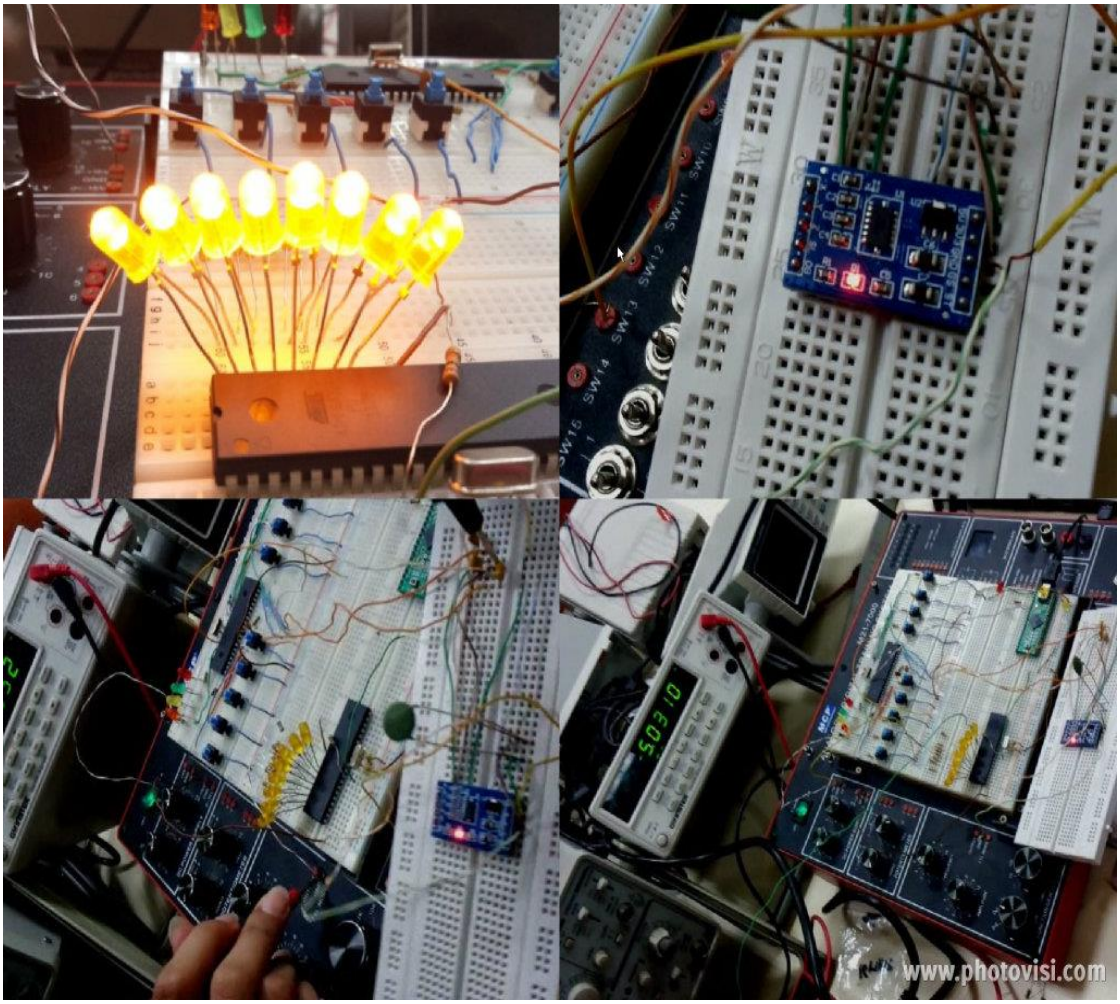


Figure 4-6: Bread board implementation

This was the initial microcontroller and accelerometer interfacing, which produced satisfactory results. The problem faced was in the interfacing with the teensy (base station). This problem was overcome in the PCB implementation.

PCB Implementation: The problem was narrowed down to mismatch of Baud rate in both our microcontrollers. ATmega 1284 was being operated at a baud rate at 57600 and teensy ++2.0 was operating at 64000. After studying the serial ports of the

microcontroller and troubleshooting, the two microcontrollers were interfaced successfully.

4.1 DETAILED DESCRIPTION OF DATA EXCHANGED BETWEEN TRANSMITTER AND RECEIVER FOR WIRELESS COMMUNICATION

The glove side would send a packet to the base station side containing the following bytes:

Table 4-2: Wireless Communication Packets and their data types with description

Data Type	Description
SYNC	To allow the transceiver to synchronize with transmitter to receive a packet in order
ADDR	It prevents receiver from reading packets of transceivers other than the glove transceiver
X axis	The change in cursor movement along x-axis
Y axis	The change in cursor movement along y-axis
Scroll	The change of the scroll moved
Click	The click status for left click and right click

The actions taken by the base station once it decodes the data sent by the glove mouse are:

Table 4-3: Action to be carried out by Base Station Based on Packet Type

Packet Type	Action
Tilt or scroll	Move the cursor or scroll
Button click	Press or release corresponding button

Finally, The Glove Unit features three status LEDs, connected to port D.1, D.5, and D.7. The purpose of these LEDs is to display information about the power being on or off, which type of button clicking has been performed, and finally if communication is taking place between the MCU and the transceiver.

CHAPTER FIVE

5 FUTURE WORK

5.1 INTRODUCTION

This chapter will focus on the ideas and suggestions given as input during the work done on the project. These ideas show how to expand the project into a more practical solution for real life control of inanimate objects, whether they are PC and laptops or control systems.

5.2 UNDERSTANDING THE AVENUES OF FUTURE DEVELOPMENT

Our device takes down a layer of separation between machine and computer by allowing intuitive hand gestures to interact with the computer. Any future work will be along the lines of making the product much smarter and easier to use as well as expanding its functionality to other machines and devices with whom humans interact.

5.2.1 PACKAGING

The basic factor to be considered is miniaturization of components used in the design. Using breakout boards and miniature IC packages to replace the existing IC's can do it. A compact PCB manufacturing with these miniaturized components can allow for a smarter design that takes up less space on the back of the palm. Miniaturization will also reduce the weight the user has to carry.

The pushbuttons placed at the finger pads can be replaced with an isotonic touch glove, the uses conductive thread. Using the thumb as ground terminal which, when it comes in contact with any of the other four fingers, shorts them to ground. This allows for more

natural and intuitive button pressing mechanism. This also beautifies the design to be more practical or efficient, as buttons tend to lose their elasticity over time.

5.2.2 IMPROVED WIRELESS OPTIONS

Because of the restrictions on sale of antennas in Pakistan due to rise in terrorism incidents over the past few years, it is imperative to find a band that is suitable for wireless communication, like the IS band. Imported transceiver have often utilization in a band that is free in the manufactured countries, but might be unusable in Pakistan, like the 900Mhz range, due to GSM using the same frequencies (915-935Mhz). The future work should focus on a robust wireless connectivity with lower interference factor.

This can be achieved by using either the 2.4 GHZ band or 4 MHz band. The transceivers available need to be configured extensively, but it can lower the cost (by about 3000Rs/-) for each transceiver used. In addition, these transceivers have widely used antennas that can be procured to enhance the data range. The limitation of designing a PCB trace antenna for the TRM-915-R25 antennas used in this project will be removed.

5.2.3 USING BLUETOOTH AS WIRELESS MEDIUM

To eliminate the equipment connecting to the laptop/PC, an alternative option is to use a Bluetooth module on the glove station that communicates directly with the laptop's Bluetooth module. This is suitable for short-range applications where the user will be in 1-meter radius of his laptop. Also, this places a restriction on the use with PC's due to their lack of Bluetooth module. The plus point of this is however, it eliminates the cost of using a pair of transceivers, cutting it down by half. It also increases portability, as base station does not need to be carried around with the laptop.

5.2.4 IMPLEMNTATION IN CONTROL SYSTEMS

Since the glove translates hand gestures to motions on the screen and controls the cursor, a robotic arm or any other mechanical system can be used in conjunction with the PC. This means that the user will move his hand, and the commands will be translated by the glove unit into robotic arm movements through either a wired or a wireless interface. This replaces the cursor movements with the robotic arm movements. Since the basic functionality of up/down and left right is part of the glove mouse, but it can be modified through coding to be interfaced with a robotic arm. This can be used in various fields to enhance Human-Computer interactions.

Using the ARDUINO software along with the ARDUINO language that works well with Teensy 2.0++ due to its open source nature and wide variety of libraries available. The Arduino.cc website and Arduino playground provide useful databases of knowledge with previous experiences of developers having a specific interest in control systems.

5.3 CONCLUSION

This section has offered future work areas that were perceived to be practical and were offered as critical but constructive criticism during the project work. They are also the results of the problem areas the team members faced as well as ideas got during brainstorming on how to improve this product with future effort.

CHAPTER SIX

6 CONCLUSION

6.1 OVERVIEW

'Wear a Mouse' is a wireless gesture input device that aims to replace the traditional mouse. It comprises of two separate units namely the 'glove' and the 'base station'. The purpose of the glove is to take and process user input from an accelerometer and sensors and send it to the base station. The purpose of the base station is to receive the data sent by the glove, convert in to a format understandable by user's computer and forward the data to the computer where it is translated in to mouse actions.

6.2 OBJECTIVES ACHIEVED

Our main objectives were to successfully obtain input from the user via accelerometer and convert this in to cursor movement. We were able to achieve this correctly. In addition to this we've also managed to implement mouse clicking via contact pad buttons.

6.3 LIMITATIONS:

Currently the device is showing a bit of lag in terms of translating user's movements in to cursor movements. The accuracy of translation can be further improved with time if effort is made.

6.4 APPLICATIONS:

Our project would be the equivalent of a fully functional mouse with the following applications:

- Beneficial venture for both consumers and as well as potential product manufactures (any companies that might be interested in picking our prototype and selling it commercially as their product).

- Consumers will be rid of the hassles that come with a regular mouse. No wires, no hard surface, no being tied to desks at all. Give presentations, surf the web, do all your regular mouse functions without being in touching distance of your computer.

- Making technology healthy.

While long term use of a traditional mouse harms the median nerve, our proposed design does not exert pressure at the user's wrist. This means prevention of carpal tuned syndrome and stress injuries caused by the regular mouse.

- Our project could serve as the starting point for numerous other gesture based applications. Further technical manipulation of our device could allow it to be developed as a video game controller using intuitive hand movements or a device that translates gestures in to speech or music.

- The possibilities of exploitation of this technology would be endless since control of the computer literally lies at the user's fingertips! Keeping this in mind, the device could be manipulated to develop applications specifically designed for the handicapped or special needs persons. For example, hand movements of the glove could be translated in to audio commands intended to enable a mute person to converse with other people.

APPENDIX A

GLOVE SIDE CODE FOR ATMEGA 1284

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#define F_CPU        16000000          // Define CPU clock frequency
#define BAUDRATE 57600  // Baud rate for the radio transmitter
#define UBRRVAL ((F_CPU/(BAUDRATE*16UL))-1)     // Calculate the baud rate
setting
#define SYNC 0XFF  // SYNC byte for packet
#define ADDR 0xAA // Address byte for packet
#define timeout 30              // Timeout value for the state machine
#define NoPush 1                // Wait for a button to be pressed
#define MaybePush 2             // Debounce the button press
#define Pushed 3                // Wait for a button to be released
#define MaybeNoPush 4           // Debounce the button release
#define left_click        0x80
#define middle_click  0x40
#define right_click             0x20
#define rapid_fire                              0x08
#define left_click_release      0x01
#define middle_click_release 0x02
#define right_click_release             0x04
```

```
unsigned char msg_C7;
unsigned char msg_C6;
unsigned char msg_C5;
unsigned char msg_C7_release;
unsigned char msg_C6_release;
unsigned char msg_C5_release;
volatile unsigned char tasktime;        // Timeout counter
unsigned char PushState;                // State machine for click buttons
unsigned char PushState_switch;         // State machine for move enable/disable
button
unsigned char butnumPressed;            // Stores which buttons were pressed for
debounce
unsigned char currentButton;        // Which buttons are just pressed //the three task
subroutines
unsigned char currentSwitch;        // Which switches are toggled
unsigned char switchToggled;            // Stores which switches were toggled for
debounce
// Task subrouting for debouncing
void task1(void);
// Mouse control parameters
volatile unsigned char xAxis; // X Axis of the mouse
volatile unsigned char yAxis; // Y Axis of the mouse
volatile unsigned char wheel; // Scroll position
```

```
volatile unsigned char click;   // Information for mouse button clicks
// Acceleration parameters
#define accel_timeout 2                 // Timeout value for the state machine
char move_en;                           // Flag for move enable
volatile unsigned char aX;              // Register for ADC sample of x axis tilt
volatile unsigned char aY;              // Register for ADC sample of y axis tilt
volatile unsigned char aZ               // Register for ADC sample of z axis tilt
volatile signed long aX_ref;   // x axis offset for calibration
volatile signed long aY_ref;   // y axis offset for calibration
volatile signed long aZ_ref;   // z axis offset for calibration
volatile signed int accel_x;     // Calibrated value of x axis tilt
volatile signed int accel_y;     // Calibrated value of y axis tilt
volatile signed int accel_x_prev;       // Store previous acceleration for cursor acceleration
volatile signed int accel_y_prev;       // Store previous acceleration for cursor acceleration
volatile signed int scale_x;     // Scale parameter for cursor acceleration
volatile signed int scale_y;     // Scale parameter for cursor acceleration
volatile signed int move_x;             // x axis tilt to transmit
volatile signed int move_y;             // y axis tilt to transmit
#define thresh 3
volatile unsigned char acceltime;
// Flag for scroll enable
char scroll_en;
// Time count for scrolling
```

```
volatile unsigned char scrolltime;
// Flag for rapid fire enable
char rapid_fire_en;
// Flag for transmitting rapid fire
char send_rapid_fire;
// Flag for invert_x enable
signed int invert_x;
// Flag for invert_y enable
signed int invert_y;
// Mode for axis frame
char accel_mode;
// Flag for sensitivity of cursor movement
unsigned char sensitivity;
// UART file descriptor
// putchar and getchar are in uart1.c
FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar,
_FDEV_SETUP_RW);
void USART_txByte(unsigned char data)
{
        while (!(UCSR1A & (1 << UDRE1)));          // Wait until UDR is empty
        UDR1 = data;  // Load data into UDR
}
void tx_packet( uint8_t x, uint8_t y, uint8_t wh, uint8_t cl )
```

```
{
        USART_txByte(SYNC); // Send SYNC byte
        USART_txByte(ADDR); // Send address to recognize correct receiver
        USART_txByte(x);    // Send X Axis of the mouse
        USART_txByte(y);    // Send Y Axis of the mouse
        USART_txByte(wh);  // Send Scroll position
        USART_txByte(cl);    // Send information for mouse button clicks
}
void readAccel(void)
{
        if (accel_mode == 1)
        {
                // Read z component (as x axis)
                ADMUX = (1<<ADLAR)|(1<<REFS0);       // Select ADC channel0 (z
component)
                ADCSRA |= (1 << ADSC);                    // Enable ADC
                while (ADCSRA & (1 << ADSC));             // Wait until ADC Data
register is full
                aX = ADCH;              // Read high byte of ADC Data Register
                // Read y component (as y axis)
                ADMUX = (1<<ADLAR)|(1<<REFS0)+1;   // Select ADC channel1 (y
component)
                ADCSRA |= (1 << ADSC);    // Enable ADC
```

```
                while (ADCSRA & (1 << ADSC));// Wait until ADC Data register is full
                aY = ADCH;              // Read high byte of ADC Data Register
    }
        else
        {
                // Read y component (as y axis)
                ADMUX = (1<<ADLAR)|(1<<REFS0)+1;   // Select ADC channel1 (y
component)
                ADCSRA |= (1 << ADSC);                          // Enable ADC
                while (ADCSRA & (1 << ADSC));             // Wait until ADC Data
register is full
                aY = ADCH;   // Read high byte of ADC Data Register
                // Read x component (as x axis)
                ADMUX = (1<<ADLAR)|(1<<REFS0)+2;   // Select ADC channel2 (x
component)
                ADCSRA |= (1 << ADSC);    // Enable ADC
                while (ADCSRA & (1 << ADSC));// Wait until ADC Data register is full
                aX = ADCH;   // Read high byte of ADC Data Register
        }
}
void calibrate(void)
{
  unsigned int cnt;
```

```
 cnt = 0;
 while (cnt < 1024)
 {
        ADMUX = (1<<ADLAR)|(1<<REFS0);      // Select ADC channel0
        ADCSRA |= (1 << ADSC);              // Enable ADC
        while (ADCSRA & (1 << ADSC));       // Wait until ADC Data register is
full
        aZ = ADCH;                          // Read high byte of ADC Data Register
        ADMUX = (1<<ADLAR)|(1<<REFS0)+1;    // Select ADC channel1
        ADCSRA |= (1 << ADSC);              // Enable ADC
        while (ADCSRA & (1 << ADSC));       // Wait until ADC Data register is full
        aY = ADCH;                          // Read high byte of ADC Data Register
        ADMUX = (1<<ADLAR)|(1<<REFS0)+2;    // Select ADC channel2
        ADCSRA |= (1 << ADSC);                      // Enable ADC
        while (ADCSRA & (1 << ADSC));       // Wait until ADC Data register is
full
        aX = ADCH;                          // Read high byte of ADC Data Register
        aX_ref = aX_ref + aX;
        aY_ref = aY_ref + aY;
        aZ_ref = aZ_ref + aZ;
        cnt++;
 }
 aX_ref = aX_ref >> 10;
```

```
 aY_ref = aY_ref >> 10;
 aZ_ref = aZ_ref >> 10;
 // Decide the axes based on the starting tilt postion
 // If the accelerometer is set handshake, use y and z axis.
 if ((60 < aZ_ref) && (aZ_ref < 100))
 {
        accel_mode = 1;
        aX_ref = aZ_ref;
 }
 // Otherwise, normal mode (use x and y axis)
 else accel_mode = 0;
 // Initialize cursor acceleration parameters
 accel_x_prev = aX_ref;
 accel_y_prev = aY_ref;
}
/* ------------------------------------------------------------
 * Initialize UART:
 *              Set up the baud rate and USART settings for data transmission. */
void USART_init(void)
{
        // Baud rate setting
        UBRR1L = (uint8_t) UBRRVAL;                     // Low byte
        UBRR1H = (uint8_t) (UBRRVAL>>8);                // High byte
```

```
        // Enable transimission
        UCSR1B = (1<<TXEN1);
        // USART setting: SPI mode, odd parity check enabeld, 2-bit stop bits, and 9-bit
data size
        UCSR1C = (0<<UMSEL10) | (0<<UMSEL11) | (0<<UPM11) | (0<<UPM10) |
(0<<USBS1) | (3<<UCSZ10);
}
/* ------------------------------------------------------------
 *  Timer 0 compare ISR:
 *              Interrupt occurs every 4 ms.
 *              The ADC sample is read and the tilt is computed as cursor movement or
scroll, then transmitted (every 8 ms).
 *              If in rapid fire mode, transmit a rapid fire pacekt every 30ms. */
ISR (TIMER0_COMPA_vect)
{
  // Decrement the task time counts if they are not already zero
  if (tasktime>0)         --tasktime;
  if (acceltime>0)  --acceltime;
  // Every 8ms, read ADC value from accelerometer and compute the change of the cursor
position
  if (acceltime==0) {
        acceltime = accel_timeout;     // Reset timeout counter
        readAccel();    // Read ADC value from accelerometer
```

```
        // Eliminate the offset by subracting the zero reference
        if (accel_mode == 1) accel_x = aX_ref - aX;
        else      accel_x = aX - aX_ref;
        accel_y = aY - aY_ref;
        // Set up threshhold to prevent drift
        if ((-thresh <= accel_x) && (accel_x <= thresh)) accel_x = 0;
        if ((-thresh <= accel_y) && (accel_y <= thresh)) accel_y = 0;
        // Compute scale parameters based on the cursor acceleration and position
inversion
        if (accel_x > accel_x_prev) scale_x = invert_x * (accel_x - accel_x_prev + 8) >>
3;
        else scale_x = invert_x * (accel_x_prev - accel_x + 8) >> 3;
        if (accel_y > accel_y_prev) scale_y = invert_y * (accel_y - accel_y_prev + 8)
>>3;
        else scale_y = invert_y * (accel_y_prev - accel_y + 8) >> 3;

        // Store the tilt value for computing cursor acceleration
        accel_x_prev = accel_x;
        accel_y_prev = accel_y;

        // Compute the change of the cursor postion to be sent to base station, based on
scale and senstivity
        move_x = (accel_x >> sensitivity) * scale_x;
```

```
        move_y = (accel_y >> (sensitivity + scroll_en)) * scale_y;
    }
    // Every other 8ms, transmit the cursor position or scroll message to the base station
    else if (acceltime == 1) {
        // If scroll is enabled, transmit the scroll message every 720ms
        if (scroll_en == 1) {
            scrolltime++;  // Increase the time count for scroll message transmission
            //if (move_y > (35 >> (sensitivity + 1)) * scale_y) tx_packet(0x00, 0x00, -
0x01, 0x00);
            //else if (move_y < - ((35 >> (sensitivity + 1)) * scale_y)) tx_packet(0x00,
0x00, 0x01, 0x00);

            if (scrolltime == 90) {
                scrolltime = 0;
                if (move_y > 0) tx_packet(0x00, 0x00, -0x01, 0x00);
                else if (move_y < 0) tx_packet(0x00, 0x00, 0x01, 0x00);
            }
        }
        // If move is enabled and scroll is disabled, transmit the cursor position message
to the base station
        else if (move_en == 1)          tx_packet(move_x, move_y, 0x00, 0x00);
    }
    // If in rapid fire mode, transmit left button click message every 30ms
```

```
 if (send_rapid_fire == 1) {
                if (tasktime % 15 == 2)  tx_packet(0x00, 0x00, 0x00, rapid_fire);
 }
}
/* ------------------------------------------------------------
 * Main loop:
 *              Endless while loop to schedule the task1.
 *              Before the loop, first initialize all the settings.
 *              Execute task1 every 120ms. */
int main(void)
{
        // DEBUGGING:  on the external LEDs at Port D.7,5
   PORTD |= 0x00;     // Turn off LEDs
        DDRD |= 0xA0;    // Output for LEDs at Port D.7,5
        // Set up switches
        DDRB = 0x00;          // Port B pins are input
        PORTB = 0xff;         // Turn on pullup registers on PORT B
        // Set up buttons
        DDRC = 0x00;          // Port C pins are input
        PORTC = 0xff;         // Turn on pullup registers on PORT C
        // Set up button related parameters
        currentButton = 0xff;
   butnumPressed = 0xff;
```

```
        currentSwitch = 0xff;
        switchToggled = 0xff;
    PushState = NoPush;
        PushState_switch = NoPush;
    tasktime = timeout;
        msg_C7 = left_click;
        msg_C6 = middle_click;
        msg_C5 = right_click;
        msg_C7_release = left_click_release;
        msg_C6_release = middle_click_release;
        msg_C5_release = right_click_release;
        scrolltime = 0;
        // Set up timer 0 as 4 milliseconds timebase
        TIMSK0= (1<<OCIE0A);      // Turn on timer 0 cmp match ISR
        OCR0A = 249;                        // Set the compare register to 250 time ticks
        TCCR0B= 4;                          // Set prescalar to divide by 256
        TCCR0A= (1<<WGM01) ;    // Turn on clear-on-match
        // ADC setup
        ADMUX = (1<<ADLAR)|(1<<REFS0);       // 1. Read ADCH (High byte of ADC
data register) 2. Select VCC as voltage reference
        ADCSRA = (1<<ADEN)|(1<<ADSC) + 7;    // Enable the ADC, start conversion,
and select 128 as ADC prescalar
        // Port A is an input for ADC
```

```
DDRA=0x00;
// Initialize acceleration related parameters
acceltime = 0;
aX = 0;
aY = 0;
aZ = 0;
accel_x = 0;
accel_y = 0;
scale_x = 0;
scale_y = 0;
// Initialize mode flags and parameters
move_en = 0;
scroll_en = 0;
rapid_fire_en = 0;
send_rapid_fire = 0;
invert_x = 1;
invert_y = 1;
accel_mode = 0;
// Initialize the UART1 for communicating with RF receiver
USART_init();
// Enable interrupt
sei();
while (1)
```

```
        {
                // Call the task for debouncing
      if (tasktime ==0) task1();
        }
}
/* ------------------------------------------------------------
 * Task 1:
 *              State machines for the button press debouncing and switch toggle
debouncing.
 *              Update the flags and transmit packets corresponding to the states. */
void task1(void)
{
 tasktime = timeout;     // Reset the task timer
 currentButton = PINC;              // Read currently pressed buttons
 currentSwitch = PINB;              // Read currently toggled switches
 switch (PushState)
 {
   case NoPush:
                // If any button is pressed, it may be a valid press
     if ((currentButton != 0xff)) {
                  PushState=MaybePush;
                  butnumPressed = currentButton;
                }
```

```
      break;
    case MaybePush:
                // If same button is pressed, it is a valid press
      if (butnumPressed == currentButton) {
        PushState=Pushed;
                  PORTD ^= 0x20;    // DEBUGGING: Toggle LED at Port D.5 to debug

                  // For normal button operations
                  if (button_setting == AllSet) {
                        // If Port C.7 button is pressed
                if ((~currentButton & 0x80) == 0x80) {
                              // Transmit a packet
                              cli();
                              if (rapid_fire_en == 1) send_rapid_fire = 1;
                              else     tx_packet(0x00, 0x00, 0x00, msg_C7);
                              sei();
                        }
                        // If Port C.6 button is pressed
                        else if ((~currentButton & 0x40) == 0x40) {
                              cli();
                              tx_packet(0x00, 0x00, 0x00, msg_C6); // Transmit
a packet
                              sei();
```

```
                                        }
                                        // If Port C.5 button is pressed
                                        else if ((~currentButton & 0x20) == 0x20) {
                                                cli();
                                                tx_packet(0x00, 0x00, 0x00, msg_C5); // Transmit
a packet
                                                sei();
                                        }
          }
                    }
                    // Otherwise, it is not a valid press
      else PushState=NoPush;
      break;
    case Pushed:
                    // If the button press has changed, it may be a valid release
      if (butnumPressed != currentButton)               PushState=MaybeNoPush;
      break;
    case MaybeNoPush:
                    // If the button press is actually not changed, it is not a valid release
      if (butnumPressed == currentButton) PushState=Pushed;
                    // Otherwise, it is a valid release
      else {
        PushState=NoPush;
```

```
                PORTD ^= 0x20;     // DEBUGGING: Toggle LED at Port D.5 to debug


                // If Port C.7 button is released
        if ((~butnumPressed & 0x80) == 0x80) {
                        cli();
                        tx_packet(0x00, 0x00, 0x00, msg_C7_release); // Transmit
a packet
                        sei();
                        send_rapid_fire = 0;
                }
                // If Port C.6 button is released
                else if ((~butnumPressed & 0x40) == 0x40) {
                        cli();
                        tx_packet(0x00, 0x00, 0x00, msg_C6_release); // Transmit
a packet
                        sei();
                }
                // If Port C.5 button is released
                else if ((~butnumPressed & 0x20) == 0x20) {
                        cli();
                        tx_packet(0x00, 0x00, 0x00, msg_C5_release); // Transmit
a packet
                        sei();
```

```
                    }
                    // If Port C.4 button is released (Move Enable)
                    else if ((~butnumPressed & 0x10) == 0x10) {
                            move_en ^= 1;// Toggle the flag for move enable
                            PORTD ^= 0x80;      // DEBUGGING: Toggle LED at
Port D.7 to debug
                            if (move_en == 1) {
                                    cli();
                                    calibrate();      // Set reference accel
                                    sei();
                            }
                    }
                    // If Port C.3 button is released (Scroll Enable)
                    else if ((~butnumPressed & 0x08) == 0x08) {
                            scroll_en ^= 1;// Toggle the flag for scroll enable
                            if (scroll_en == 1) {
                                    cli();
                                    calibrate();      // Set reference accel
                                    sei();
                            }}
                    butnumPressed = 0xff;   }
        break;
 }
```

```
// State machine for switches
switch (PushState_switch)
{ case NoPush:
                // If any switch is toggled, it may be a valid toggle
    if ((currentSwitch != switchToggled)) {
                  PushState_switch=MaybePush;
                  switchToggled = currentSwitch;
                }
    break;
  case MaybePush:
                // If same switch is still toggled, it is a valid toggle
    if (currentSwitch == switchToggled) {
      PushState_switch=Pushed;

                  // If Port B.0 is on
      if ((~currentSwitch & 0x01) == 0x01) sensitivity = 4;
                // If Port B.0 is off
                else if ((~currentSwitch & 0x01) == 0x00) sensitivity = 3;

                // If Port B.1 is on
                if ((~currentSwitch & 0x02) == 0x02) rapid_fire_en = 0;
                // If Port B.1 is off
                else if ((~currentSwitch & 0x02) == 0x00) rapid_fire_en = 1;
```

```
                    // If Port B.2 is on
                    if ((~currentSwitch & 0x04) == 0x04) invert_x = 1;
                    // If Port B.2 is off
                    else if ((~currentSwitch & 0x04) == 0x00) invert_x = -1;
                    // If Port B.3 is on
                    if ((~currentSwitch & 0x08) == 0x08) invert_y = 1;
                    // If Port B.3 is off
                    else if ((~currentSwitch & 0x08) == 0x00) invert_y = -1;
                }
                // Otherwise, it is not a valid toggle
  else PushState_switch=NoPush;
  break;
case Pushed:
                // If any switch is toggled, it may be a valid toggle
  if (currentSwitch != switchToggled) {
                    PushState_switch=MaybeNoPush;
                    switchToggled = currentSwitch;
                }
  break;
case MaybeNoPush:
                // If same switch is still toggled, it is a valid toggle
  if (currentSwitch == switchToggled) {
```

```
        PushState_switch=NoPush;
                    // If Port B.0 is on
        if ((~currentSwitch & 0x01) == 0x01) sensitivity = 4;
                    // If Port B.0 is off
                    else if ((~currentSwitch & 0x01) == 0x00) sensitivity = 3;
                    // If Port B.1 is on
                    if ((~currentSwitch & 0x02) == 0x02) rapid_fire_en = 0;
                    // If Port B.1 is off
                    else if ((~currentSwitch & 0x02) == 0x00) rapid_fire_en = 1;
                    // If Port B.2 is on
                    if ((~currentSwitch & 0x04) == 0x04) invert_x = 1;
                    // If Port B.2 is off
                    else if ((~currentSwitch & 0x04) == 0x00) invert_x = -1;
                    // If Port B.3 is on
                    if ((~currentSwitch & 0x08) == 0x08) invert_y = 1;
                    // If Port B.3 is off
                    else if ((~currentSwitch & 0x08) == 0x00) invert_y = -1;
      }
                  // Otherwise, it is not a valid toggle
                  else PushState_switch=Pushed;
      break;
        }
}
```

APPENDIX B

BASE STATION SIDE CODE FOR TEENSY

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdint.h>
#include <util/delay.h>
#include "usb_mouse.h"
#define BAUDRATE 57600  // Baud rate for the radio transmitter
#define UBRRVAL ((F_CPU/(BAUDRATE*16UL))-1)     // Calculate the baud rate
setting
#define SYNC 0XFF  // SYNC byte for packet
#define ADDR 0xAA // Address byte for packet
#define CPU_PRESCALE(n) (CLKPR = 0x80, CLKPR = (n))
// LED configuration and on/off for DEBUGGING
#define LED_ON          (PORTD |= (1<<6))
#define LED_OFF         (PORTD &= ~(1<<6))
#define LED_CONFIG      (DDRD |= (1<<6))
volatile unsigned char sync;   // SYNC byte
volatile unsigned char addr;   // ADDRRESS byte
volatile unsigned char xAxis; // X Axis of the mouse
volatile unsigned char yAxis; // Y Axis of the mouse
volatile unsigned char wheel; // Scroll position
```

```
volatile unsigned char click;   // Information for mouse button clicks
volatile unsigned char rapid_fire;      // Flag for rapid fire clicks
volatile unsigned char firetime;        // Timeout counter for rapid fire
volatile unsigned char step_x;          // Step for x axis movement
volatile unsigned char step_y;          // Step for y axis movement
/* ------------------------------------------------------------
 * Initialize UART:
 *              Set up the baud rate and USART settings for data transmission. */
void uart_init(void)
{
        // Baud rate setting
        UBRR1L = (uint8_t) UBRRVAL;                             // Low byte
        UBRR1H = (uint8_t) (UBRRVAL>>8);                // High byte
        // Enable Receiver and Interrupt on receive complete
        UCSR1B = (1<<RXEN1) | (1<<RXCIE1);
        // USART setting: SPI mode, odd parity check enabeld, 2-bit stop bits, and 9-bit
data size
        UCSR1C = (0<<UMSEL10) | (0<<UMSEL11) | (0<<UPM11) | (0<<UPM10) |
(0<<USBS1) | (3<<UCSZ10);
}
/* ------------------------------------------------------------
 * Receive a byte from the transceiver:
 *              Load a byte from UDR. */
```

```
uint8_t uart_getchar(void)
{
        // Wait until the USART Receive Complete and then return the data from UDR
        while (!(UCSR1A & (1 << RXC1)));
        return UDR1;
}
/* ------------------------------------------------------------
 *  USART Receive Complete Interrupt Serivce Routine:
 *              Interrupt occurs when USART recieves data.
 *              Decode the recieved packets and perform action on the mouse cursor. */
ISR(USART1_RX_vect)
{
   // Receive messages in order if sync byte matches
        sync = uart_getchar();
   if (sync == SYNC)
        {
                // Receive further messages if address matches
                addr = uart_getchar();
                if (addr == ADDR)
                {
                        // Get the other data
                        xAxis = uart_getchar();         // Cursor movement on x axis
                        yAxis = uart_getchar();         // Cursor movement on y axis
```

```
        wheel = uart_getchar();          // Scroll movement
        click = uart_getchar();// Click Status
// If Port C.7 button is pressed (Left Click)
        if ((click & 0x80) == 0x80) {
                LED_ON;
                usb_mouse_buttons(1, 0, 0);
        }
        // If Port C.6 button is pressed (Middle Click)
        else if ((click & 0x40) == 0x40) {
                LED_ON;
                usb_mouse_buttons(0, 1, 0);
        }
        // If Port C.5 button is pressed (Right Click)
        else if ((click & 0x20) == 0x20) {
                LED_ON;
                usb_mouse_buttons(0, 0, 1);
        }
        // If Port C.7 button is released (Left Click)
        else if ((click & 0x01) == 0x01) {
                LED_OFF;
                usb_mouse_buttons(0, 0, 0);
        }
        // If Port C.6 button is released (Middle Click)
```

```
                        else if ((click & 0x02) == 0x02) {
                                LED_OFF;
                                usb_mouse_buttons(0, 0, 0);
                        }
                        // If Port C.5 button isreleased (Right Click)
                        else if ((click & 0x04) == 0x04) {
                                LED_OFF;
                                usb_mouse_buttons(0, 0, 0);
                        }
                        // If Port C.7 button is pressed in rapid fire mode (Repeated Left
Clicks)
                        else if ((click & 0x08) == 0x08) {
                                LED_ON;
                                if (rapid_fire == 1) rapid_fire = 0;
                                else rapid_fire = 1;
                                usb_mouse_buttons(rapid_fire, 0, 0);
                        }
                        // Move the mouse cursor
                        usb_mouse_move(xAxis, yAxis, wheel);
                }
        }
}
/* ------------------------------------------------------------
```

```
 * Main loop:
 *              Endless while loop to wait for the receive interrupt.
 *              Before the loop, first initialize all the settings. */
int main(void)
{
        // Set CPU to run at 16 MHz
        CPU_PRESCALE(0);
        // Set up on-board LED
        LED_CONFIG;
        LED_OFF;
        // Initialize the usb interface and USART interface
        usb_init();
        while (!usb_configured())
        _delay_ms(1000);
        uart_init();
        // Initialize flag for rapid fire mode
        rapid_fire = 0;
        // Enable interrupt
   sei();
        // An infinite while loop
        while (1) {}
}
```

BIBLIOGRAPHY

REFERENCES

1. Coxworth, B , "ION Wireless Air Mouse". Retrieved January , 2014

   Available:http://www.pcmag.com/article2/0,2817,2403538,00.asp

2. "USB human interface device class". Retrieved January , 2013

   Available: http://en.wikipedia.org/wiki/USB_human_interface_device_class

3. "Device Class Definition for Human Interface Devices (HID) ".

   Retrieved January, 2014

   Available: http://www.usb.org/developers/devclass_docs/HID1_11.pdf

4. "Plug and Play ". Retrieved January , 2013

   Available: http://www.techopedia.com/definition/3413/plug-and-play-pnp

5. Goodrich, R, " Accelerometers: What They Are & How They Work".

   Retrieved October, 2013

   Available: http://www.livescience.com/40102-accelerometers.html

6. "Human Interface Device (HID)". Retrieved October , 2013

   Available: http://www.techopedia.com/definition/19781/human-interface-device-hid

7. "Mi.Mu Gloves for Music". Retrieved October , 2013
Available: http://theglovesproject.com/about-the-project/

8. , , " ". RetrievedJanuary , 2013
Available: https://github.com/shadabahmed/avr_lib/blob/master/ATmega128Lib/uart1.c#\

9. "Accelerometer". RetrievedJanuary , 2013
Available: http://sensorwiki.org/doku.php/sensors/accelerometer

10. , , "Accelerometers". RetrievedJanuary , 2013
Available: http://www.hobbytronics.co.uk/accelerometer-info

11. , " ". RetrievedJanuary , 2013
Available: http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=130510

12. , , " ". RetrievedJanuary , 2013
Available: http://avrprogrammers.com/devices/atmega64x128

13. , , "Atmel Avr". RetrievedJanuary , 2013
Available: http://en.wikipedia.org/wiki/Atmel

14. , , " ". RetrievedJanuary , 2013
Available: http://www.atmel.com/products/microcontrollers/avr/megaavr.aspx

15. , , " ". RetrievedJanuary , 2013
Available: http://physics.rutgers.edu/~aatish/teach/srr/workshop3.pdf