Founded 1905

WEARABLE ROBOTIC SYSTEM FOR INTERACTIVE DIGITAL MEDIA

WANG BIN

(B.Eng., NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2011

DECLARATION

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Wang Bin

10 Apr 2012

Acknowledgements

I would like to express my sincere gratitude to my supervisor Professor Shuzhi Sam Ge who has inspired me into this research direction and guided me throughout the whole journey.

I would also like to give my sincere thanks to my co-supervisor Professor Tong Heng Lee for his guidance, teach and support to my research.

I would like to appreciate the support and helpful discussions on my research work from Dr He Wei, Mr Zhang Qun, Dr Tao Pey Yuen, Ms He Peiwen and Mr Bian Haojie from the Edutainment Robotics Lab, Department of Electrical and Computer Engineering, the National University of Singapore (NUS). Same thanks go to Mr Benny Jaya and Ms Joleen Seto from Portege Pte Ltd for the support on the experiments of this thesis.

Last but not least, I would also like to give my special thanks to my family who have given me unyielding support.

Summary

This thesis studies the design and development of Wearable Robotic System (WRS) for Interactive Digital Media (IDM). WRS is defined as a wearable device system that user can wear on the body and use motion and gesture to interact with the digital contents naturally on computer. The research work includes design and development of a wearable robotic system for human gesture and motion detection, design and development of a wireless Universal Serial Bus (USB) plug-and-play device to interface to computer, and development of an interactive Taichi game played by WRS for exercises.

The current solutions of Human Computer Interface (HCI) that people use in daily life mainly include flat keyboard, joysticks and mouse. This research work mainly contributed on developing a wearable robotic system to improve the current HCI on mobility and interactivity and provide computer game players with intuitive, interactive, immersive and convenient user experiences.

- The first part of the thesis focuses on the design and development of an embedded system of wearable robotic sensory device for human motion and gesture capture. A mathematical model for gesture recognition is proposed and experiments are set up to investigate on the proposed model. This part of the thesis also illustrates on the detail process on the design and implementation of a wireless USB cross platform plug and play human-machine interfacing device such that the system can work wirelessly and does not need users to install additional driver.

- Second part of the thesis focuses on developing of an interactive Taichi game on computer to be played by the wearable robotic system developed in this project. The aim is to develop an interactive digital application to illustrate the application of WRS. A complete development cycle for creating interactive digital contents on WRS is elaborated. A total of 20 game testers participated the game play testing and the game play scores are discussed.

Contents

List of Figures

List of Symbols

HCI	Human Computer Interface
HRI	Human Robot Interface
WRG	Wearable Robotic Glove
LDR	Light Dependent Resistance
WRS	Wearable Robotic System
IDM	Interactive Digital Media
USB	Universal Serial Bus
HID	Human Interface Device

Chapter 1. Introduction

1.1. Overview

In this thesis, the research work of design and implementation of Wearable Robotics System through Sensor Fusion (WRS) for Interactive Digital Media is presented and discussed.

The traditional human computer interface (HCI) has limitations. The keyboard and mouse requires the users to remain at one position to perform typing keys and moving mouse which limits their freedom. While wireless versions of these peripherals are available, it is still not possible for users to use a mouse without a desk or a flat surface. Research has shown that 70 percent of all meaning for human interactions are derived from nonverbal body language which involves body motion, gesture and facial expressions [1]. As such, to move beyond the limitations of current human computer interface and towards the scenario portrayed in Minority Report shown in Figure 1, this research work develops a wearable robotic system which incorporate the motion and gesture into HCI. The user wears the WRS on different parts of the body such as wrist and ankle. The WRS is able to capture the user's motion and gesture and wirelessly transmit the information to a USB device which is plugged into the computer. These motions will have to be interpreted by the system into commands to the computer by this USB device.

Figure 1: WRG System in Imaginary Movie Minority Report

1.2. Problem Statement and Motivation

Currently the interface for human computer interaction mainly comprises flat keyboard, mouse and pad-like controllers for entertainment purposes. The flat keyboard has to be statically put on a surface for typing, and the mouse function are separated from the keyboard which requires the hand of the user to constantly switch control between mouse and keyboard which results in a reduction of interactivity and enjoyment in entertainment. Research has shown that the long time usage of keyboard and mouse interface could result in tiredness and even occupational disease such as carpal tunnel syndrome [2]. Hence design a more interactive and mobile interface for HCI is the objective of this project.

In this research work, the aim is to research and develop wearable robotic devices to enhance interaction experiences for the user with digital media in daily life. The research work of Wearable Robotics for Interactive Digital Media focuses on improving human machine interaction experience through wearable robotic technology, in particular catering for the interaction with digital media contents and applications. The research approach is adopted from the standpoint of wearable

robotics, since many of the latent issues [7, 8] in next-generation human-computer interfaces, including real-time computational processes [9,10,11], sensor fusion [12], intelligent system [13,14], wireless sensor network, have already been studied extensively by the robotics community, and some of these fields including sensors have since matured and developed results in technologies [15] that can be further studied for potential extensions to human-computer interfaces.

In recent years, wearable robotics has become a reality with the development of military robotic exoskeletons [3, 4, and 5] which extends the lifting capabilities and endurance of soldiers beyond human limitations. Wearable robotics [6] is defined as robots that are self-contained and/or part of a distributed system of robots that can function as or be integrated with our clothing and accessories. In contrast to conventional concept of 'robots' as anthropomorphic or animal-like machines with mobility and intelligence, we generalize our understanding of 'robots' as agents with sensing and information processing capability, which interact with the physical world. In general, robots need not have mobile arms or legs, but can be a ubiquitous artifact, such as a chair. They do not even need to be able to move, but just have means of interaction with the world, through capturing user's motion and gesture.

1.3. Objective

The research work has the following objectives:

- To investigate on the gesture recognition model using accelerometer, and hence develop the wearable robotic sensory device for human motion and gesture

capture. Investigate on the best solution to achieve a wireless USB plug and play interfacing device

- To built and test on a workable prototype to prove the value of WRS system
- To develop an interactive Taichi Game to illustrate the application of WRS on digital media content.

In particular, the design of the WRS should have the following considerations.

1) Convenience: Basing on the relevant ergonomics [16, 17] available and experimental testing, the new interaction interface should be easy-put-on, portable, durable, and reliable and no harm to health.
2) Interactive: Unlike the traditional graphical user interface, the new interface make it possible that user directly control the virtual object in virtual world using their physical motion and the robot gives the user a more tactile feeling about the interaction with other players or virtual characters.
3) Immersive: The user is fully engaged in the interactive process. User is not simply clicking and dragging mouse, instead he or she could use full body motion in real daily life to control the virtual character. On the other hand, the user is not only dealing with the plain monitor, rather that he or she can interact with a more intelligent robot
4) Affordability: The material chosen for the glove minimize its weight. And it also gives the comfortable feeling when people wear it and type for input by its hand shape design interface. Hence in selecting the sensor and other electronics components, cost is an important factor.

5) Convenient: The new interface should be easy to put on and easy to take off. And it is highly mobile that people can take around more conveniently and it can even be used when people are moving around.

1.4. Contributions

This research worked has studied and developed a wearable robotic system to achieve an intuitive interface between humans and the personal electronics. The wearable robotic system in this thesis is an integrated system consisting of sensors, computational equipment and interactive devices with the purpose to aid humans in managing and manipulating objects in the information space. The personal electronics in this project are defined as the means in which humans interacts digital machines including Computers and televisions. The research and development of wearable robotic systems has built a workable prototype and seamlessly integrate the user into the virtual digital world in order to access or manipulate objects in the digital space [18]. The contributions of the research work can be summarized into three parts.

- The research work has designed and developed an embedded system of Wearable Robotic Sensory System for human natural input capturing. The system includes the sensors, computational unit which we aim to select a low cost and efficient Microcontroller and provide digital output terminals to interface with communication module. Key challenges of signal processing, calibration and gesture estimation are studied.
- The research work has developed a cross platform and plug and play wireless USB interfacing device. This part of work has solved the problem such that user

do not need to install any software driver on their computer to use the WRS system. The wireless feature enables the mobility of the system. Two major challenges are discussed in this part, including the implementation of a wireless network for multiple wearable sensors through Radio Frequency 2.4G[19] and issues to achieve a cross platform the machine interfacing engine through designing a customized USB device to be compatible with different operating systems on personal computer including Windows, iOS and Linux.

- Other than the workable prototype was built, a proof-of-concept interactive Tai Chi game was built through game engine Unity3D [20] by deploying WRS system. The result has demonstrated a complete development cycle for creating interactive digital contents on various hardware platforms using the WRS interface. And testing and discussion on the result have been conducted.

Chapter 2. Literature Review

The rapidly improving communication technologies and the availability of various sensory devices provide opportunities for creating new generations of human computer interfaces, in particular for digital entertainment and healthcare [21]. Several electronic companies and research institutes have already taken the initiative to integrate electronics to the items we wear or carry [22, 23, and 24]. One of the recent typical developments is the application of wireless sensors for real-time monitoring of physiological signs such as electrocardiogram (ECG) and photoplethysmogram (PPG) sensors are leading ubiquitous healthcare services for most chronic diseases to reality. Another development is done by MIT Media Laboratory using inertial measurement framework of gyroscope for gesture recognition [25].

In this project, instead of choosing complex and high cost sensors of EEG and Gyroscope, the research work proposes a simpler solution using accelerometer only for HCI, in particularly for the digital game playing application. The research develops an algorithm for gesture recognition on the accelerometer output. The research and development also make the system to be plug-and-play and working wirelessly. The research work aims to make improvements on existing system in terms of reducing cost, simplifying the complexity, enhancing convenience and mobility. The following session will review on the state of art solutions of motion tracking solution for HCI and also discuss on the proposed solution for improvement

on simplifying the system by deploying accelerometer, wireless module and enabling cross-platform plug and play interface.

2.1. State of Art Solutions

Various researchers have studied motion capture and tracking and different motion capture technologies [26, 27, and 28] have been proposed in the last few decades. The merits and defects of the mainstream approaches are argued in several excellent surveys. In this brief summary, we review optical, magnetic, inertial, acoustic, and hybrid systems, mentioning a few exemplary systems in each category.

Optical motion capture systems [29] and modern systems track retro-reflective markers or light-emitting diodes placed on the body. Exact 3D marker locations are computed from the images recorded by the surrounding cameras using triangulation methods. These systems are favoured in the computer-animation community and the film industry because of their exceptional accuracy and extremely fast update rates. The major disadvantages of this approach are extreme expenditure and lack of portability. To reduce cost and improve portability, some systems use a small number of markers in conjunction with standard video cameras.

Magnetic systems, such as MotionStar by Ascension Technology Corporation, detect the position and orientation using a magnetic field (either the Earth's magnetic field or the field generated by a large coil). These systems offer good accuracy and medium update rates with no line-of-sight problems. However, it requires high cost and high power consumption, and is sensitive to the presence of metallic objects in the environment.

Acoustic systems use the time-of-flight of an audio signal to get the marker locations. Most current systems are not portable and handle only a small number of markers. With the Bat system, an ultrasonic pulse emitter is worn by a user, while multiple receivers are placed at some fixed locations in the surroundings. A system extends ultrasonic capabilities by using broadband signals. The Cricket location system [30] fills the surroundings with a number of ultrasonic beacons that send pulses along with RF signals at random times in order to minimize possible signal interference. This allows multiple receivers to be localized independently. Lastly, the WearTrack system [31], developed for augmented reality applications, uses one ultrasonic beacon placed on the user's finger and three fixed detectors placed on the head-mounted display. This system can track the location of the finger with respect to the display, based on time-of-flight measurements.

Inertial motion capture systems, such as Xsens's Moven (xsens.com) and Verhaert's ALERT system (verhaert.com), measure rotation of the joint angles using gyroscopes or accelerometers placed on each body limb. Like the mechanical systems, they are quite portable, but cannot measure distances and positions directly for applications that must sample the geometry of objects using system. More importantly, the measurements drift by significant amounts over extended time periods. In addition, the motion of the root cannot be reliably recovered from inertial sensors alone, although in some cases this problem can be alleviated by detecting foot plants [32].

Hybrid systems combine multiple sensor types to alleviate their individual shortcomings while improve the accuracy and update rates. More importantly, they

aim to improve performance, rather than decrease cost and increase portability. For example, an acoustic-inertial system, Constellation TM, has been developed for indoor tracking applications [33]. The system corrects inertial drift using ultrasonic time-of-flight measurements to compute exact distances between receivers and ultrasonic beacons placed at known locations. Another acoustic-inertial system [34] uses a wrist-worn microphone and a 3-axis accelerometer for gesture recognition. Similarly, MERG sensors [35] enable inertial-magnetic systems that account for the drift by using a reference magnetic field. In the same manner, Hy-BIRDTM by Ascension Technology Corporation (ascension-tech.com) combines optical and inertial technologies to handle occlusion problems.

In this project, we proposed a simple 3 axis accelerometer for gesture recognition. The core different from the current solutions is to simply the sensor structure and mathematical model to cater for computer gaming interaction. Improvements are made on developing a low cost and efficient wireless network using RF2.4G technology. And a customized USB cross platform plug and play device is also developed to make system working without forcing user to install driver. In the following session, the literature review on developing the warless network and plug and play USB device are elaborated.

2.2. Wireless Communications

In our project, a wireless sensor network (WSN) is built consisting of spatially distributed autonomous sensors to cooperatively transfer the sensor output to the target computer.

2.2.1. Comparison of Existing Wireless Technology

A wide variety of different wireless data technologies now exist, some in direct competition with one another, others designed to be optimal for specific applications. Of the standards evaluated, these can be grouped as follows: UWB, Bluetooth, ZigBee, and Wireless USB are intended for use as so called Wireless PAN systems. They are intended for short range communication between devices typically controlled by a single person. A keyboard might communicate with a computer, or a mobile phone with a hands free kit, using any of these technologies.

WiFi is the most successful system intended for use as a WLAN system. A WLAN is an implementation of a LAN over a microcellular wireless system. Such systems are used to provide wireless Internet access and access to other systems on the local network such as other computers, shared printers, and other such devices throughout a private property. Typically a WLAN offers much better bandwidth and latency than the user's Internet connection, being designed as much for local communication as for access to the Internet, and while WiFi may be offered in many places as an Internet access system, access speeds are usually more limited by the shared Internet connection and number of users than the technology itself. Other systems that provide WLAN functionality include DECT and HIPERLAN.

Some systems are designed for point-to-point line-of-sight communications, such as RONJA and IrDA; once 2 such nodes get too far apart to directly communicate, they can no longer communicate. Other systems are designed to form a wireless mesh network using one of a variety of routing protocols.

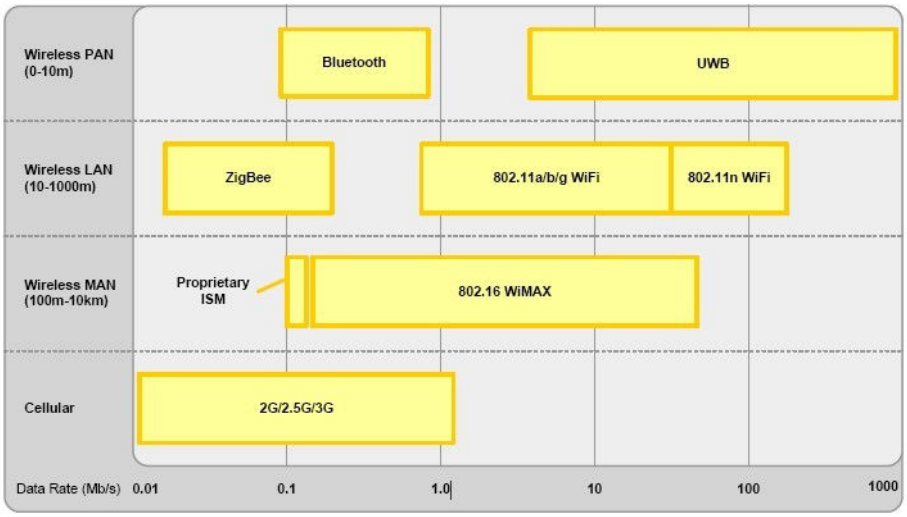


Figure 2: wireless data technologies

Wireless technologies can be evaluated by a variety of different metrics. The main metrics of consideration are communication range, throughput, and cost of device and power consumption. The comparison is made in bellow for existing technologies.

	Bluetooth	UWB	ZigBee	802.11a/b/g	802.11n	Proprietary	802.16a	2G/2.5G/3G
Typical Range	< 10m	10-30m	70-300m	100m	100m	10km	50km	Cellular Network
Modulation	Adaptive FHSS	OFDM or DS-UWB	DSSS	DSSS	DSSS	FHSS	QAM	CDMA/GSM/ AMPS
Freq. Range	2.4GHz	3.1-10.6GHz	868/915MHz 2.4GHz	2.4GHz -b/g 5.8GHz - a	2.4GHz	915MHz & 2.4GHz	2-11GHz	869-894MHz
Network	P2P	P2P	Mesh	IP & P2P	IP & P2P	P2P	IP	IP
IT Network Connectivity	No	No	No	Yes	Yes	No	Yes	Yes
Cost of Data	Free	Free	Free	Free	Free	Free	Free	Monthly Charge
Application	Cable replacement	Sync and Transmission of video/ audio data	Sensor networks	LAN, Internet	LAN, Internet	Point to point connectivity	Metro area broadband Internet connectivity	Celular telephones and telemetry

Figure 3: Parameter Comparisons of Wireless Data Technologies

In this project, the Zigbee is selected targeting specifically for wireless sensor network which is a promising field that integrates sensor technologies, embedded system and wireless communication together to produce small, low cost, low power

and reliable system capable of monitoring specific events. The sensor network is generally applied in monitoring applications with non-critical data, where longer latency is not a critical issue. Such applications usually do not need high data throughput but emphasize on power saving to maximize battery life. It has been used in a variety of applications including commercial and industrial monitoring, home automation and networking, consumer electronics, personal computer peripherals, home security, personal healthcare, toys and games, automotive sensing, agriculture etc. Figure 4 lists the data transfer rate and latency requirement of a few applications [10]. From the table, it can be observed that higher data transfer rate is required in entertainment and gaming applications. For personal health care and automation, lower data rate and higher data latency are acceptable.

Applications	Maximum data rate required (kb/s)	Maximum acceptable latency (ms)
Consumer Electronics	3	16.7
PC Peripherals	115.2	16.7
Home Automation	10	100
Personal Health Care	10	30
Toys and Games	115.2	16.7 - 100

Figure 4: Wireless Data Rate and Latency for Various Applications

Even lower rates can be considered with the resulting effect on power consumption. As already mentioned, the main identifying feature of 802.15.4 among WPAN's is the importance of achieving extremely low manufacturing and operation costs and technological simplicity, without sacrificing flexibility or generality.

Important features include real-time suitability by reservation of guaranteed time slots, collision avoidance through CSMA/CA and integrated support for secure

communications. Devices also include power management functions such as link quality and energy detection. 802.15.4-conformant devices may use one of three possible frequency bands for operation.

2.3. Cross Platform and Plug Play implementation through USB

The WRS system is interfaced with the computer through a Universal Serial Bus (USB) device. USB is a communications architecture that gives a personal computer (PC) the ability to interconnect a variety of devices using a simple four-wire cable. The USB is actually a two-wire serial communication link that runs at either 1.5 or 12 megabits per second (mbs). USB protocols can configure devices at start up or when they are plugged in at run time. These devices are broken into various device classes. Each device class defines the common behaviour and protocols for devices that serve similar functions. The HID class consists primarily of devices that are used by humans to control the operation of computer systems.

In custom applications such as a printer, a driver needs to be installed on computer. However, some drivers are available for most common host systems for the most common classes of devices such as keyboard, mouse and other Human Device Interface devices. Hence in this project, the core of achieving a plug and play feature for the system is to design a customized USB device which is able to send the sensor data into the computer at the mean time it can be recognized by the computer as a common class device.

2.3.1. HID libraries Integration Algorithms Design

Normally each USB port connect to a single device which only report to operating system with one HID class type, however this research carried out in this thesis design a virtual HID hub that combines few HID classes. The advantage to do so is to make the wearable sensor system more powerful and able to control and emulate multiple devices including keyboard, mouse, joystick and able to send customized raw data for communication.

Typical examples of HID class devices include keyboards and pointing devices—for example: standard mouse devices, trackballs, and joysticks. To design a customized compliment HID device, a new information package about a USB device should be stored in segments of its ROM (read-only memory). These segments are called descriptors. An interface descriptor can identify a device as belonging to one of a finite number of classes. .

A USB/HID class device uses a corresponding HID class driver to retrieve and route all data. The routing and retrieval of data is accomplished by examining the descriptors of the device and the data it provides.

Figure 5: HID Device Driver

A HID class device communicates with the HID class driver using either the Control (default) pipe or an Interrupt pipe.

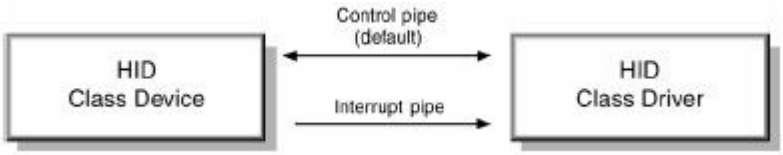


Figure 6: HID Class Driver

The Interrupt Out pipe is optional. If a device declares an Interrupt Out endpoint then Output reports are transmitted by the host to the device through the Interrupt Out endpoint. If no Interrupt Out endpoint is declared then Output reports are transmitted to a device through the Control endpoint, using Set_Report (Output) requests.

At the topmost level, a descriptor includes two tables of information referred to as the Device descriptor and the String descriptor. A standard USB Device descriptor specifies the Product ID and other information about the device. For example, Device descriptor fields primarily include Class, Subclass, Vendor, Product, and Version. And in this project, the core of the design is to define the descriptor package as review in Figure 7.

- **Device Descriptor**

The device descriptor provides general information such as manufacturer, product number, serial number, the class of the device and the number of configurations.

There is only one device descriptor.

- **Configuration Descriptor**

The configuration descriptor provides information on the power requirements of the device and how many different interfaces are supported when in this configuration.

There may be more than one configuration for a device (i.e., low-power and high-power configurations).

- **Interface Descriptor**

The interface descriptor details the number of endpoints used in this interface, as well as the class of the interface. There may be more than one interface for a configuration.

- **Endpoint Descriptor**

The endpoint descriptor identifies the transfer type and direction, as well as some other specifics for the endpoint. There may be many endpoints in a device and endpoints may be shared in different configurations.

- **String Descriptor**

Many of the previous descriptors reference one or more string descriptors. String descriptors provide human readable information about the layer they describe. Often these strings show up in the host to help the user identify the device. String descriptors are generally optional to save memory and are encoded in a Unicode format.

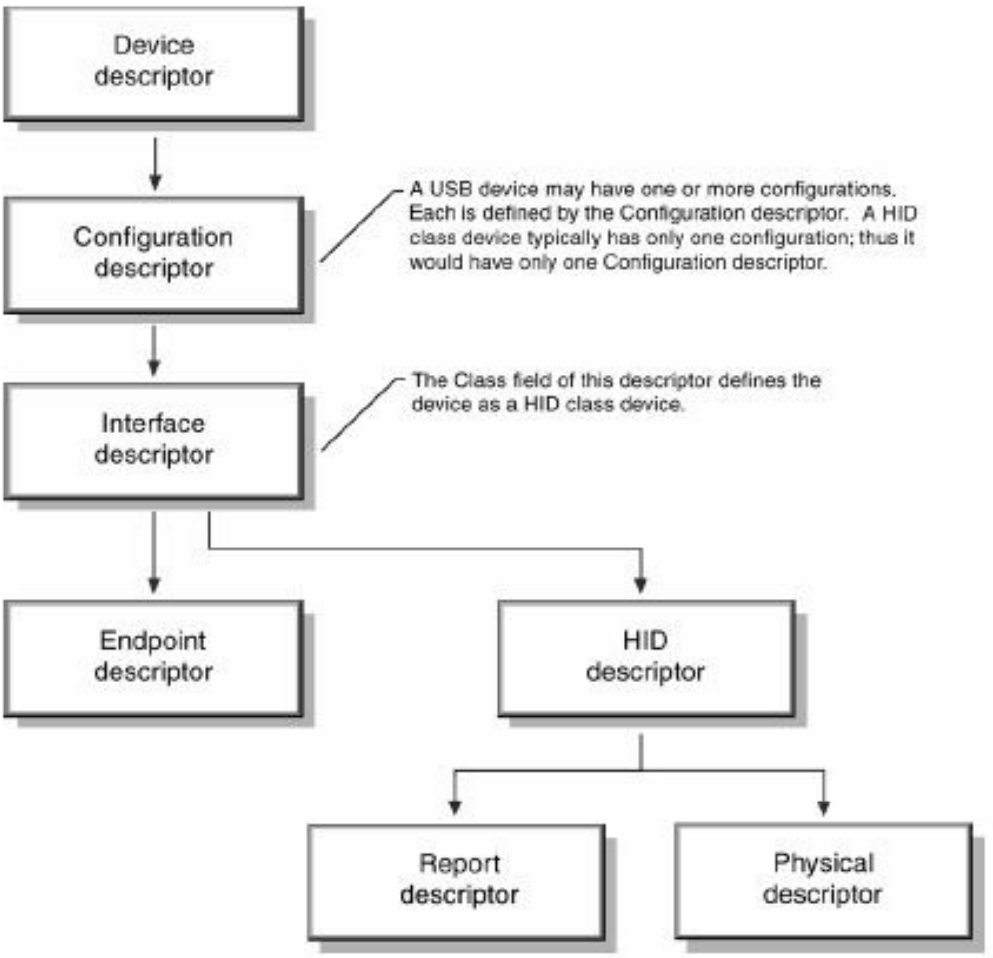


Figure 7: USB Descriptor

2.3.2. Intellectual Property Protection

To protect the IP of the research and development work in this project, patent filing is under the process with the assistant from NUS Enterprise at the National University of Singapore.

The digital contents, graphics and game images used in this thesis belong to the property of Portege Pte Ltd.

Chapter 3. Development of Wearable Robotic System

This chapter discusses on the detail implementation process of developing the Wearable Robotic System. Initially we form the system architecture based on the design objective and engineering constrains. The first prototype was developed through sensor fusion using Light Dependent Resistance (LDR) Sensors. Evaluation and observation on the first prototype provide more collective information to achieve the design objective. Improvements on mobility and intuitiveness of the system are made in the second prototype design iteration with adopting sensor fusion using Microelectromechanical (MEMS) sensors [36].

3.1. System Architecture

The Wearable Robotic System contains three modules including the Wearable Sensory Module, the Wireless Communication Module and the Plug and Play Universal Serial Bus (USB) interface module. To make the system convenient to carry and use, physically the prototype of System is presented in two devices. The first device is the wearable sensory device in the form of glove, belt, wristband or shoes. The Wearable Sensory Module and a wireless transceiver are embedded into the wearable sensory device in order to capture the user input and transmit the data into the target computer. The second device is a USB apparatus which can be plugged into the target computer. The USB apparatus contains a wireless transceiver and a Human Device Interface (HID) [37] module in order to receive the information from

the sensory system and further process the information into interpretable data of the target computer. The USB apparatus is also able to transfer information of the computer back to the wearable sensory device. The USB apparatus is designed to be a Plug and Play device such that user will not need to install software driver before using the system.

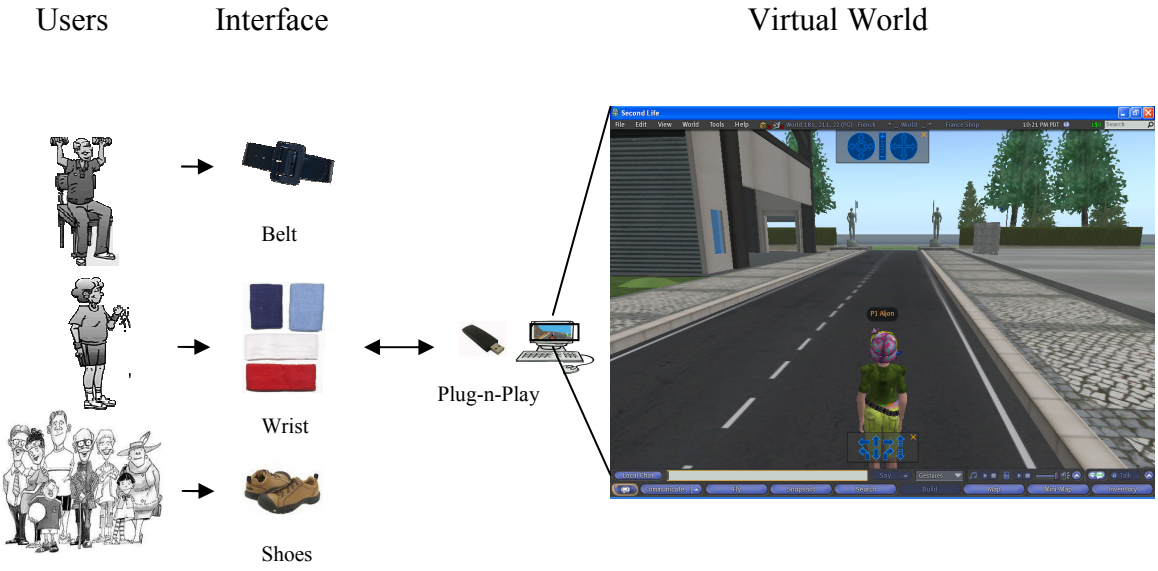


Figure 8: WRS System Diagram

The flowchart of user scenario is demonstrated in Figure 9. The user will wear the Wearable Robotic System which is able to capture the user input information and wirelessly transfer to the USB Interface Device. The user input information will be interpreted and transferred by the USB Interface Device into the data format that the low level Human Device Interface (HDI) drivers can recognize as equivalent to standard HDI control information such as keyboard strokes and mouse cursor movements. The HDI control information will be further sent to the interactive digital content and applications such as games.

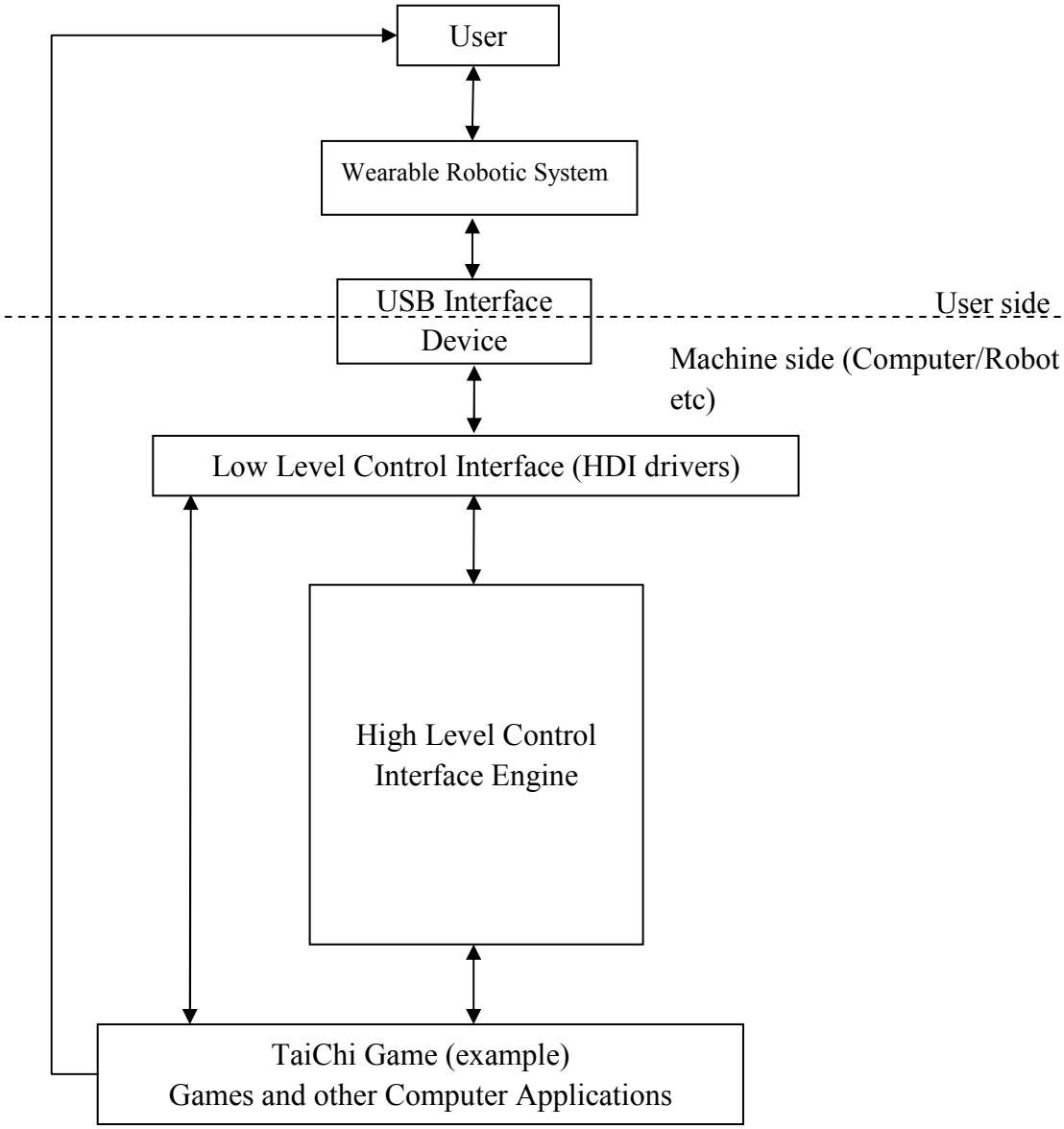


Figure 9: Flowchart of WRS

The following session will discuss on the detail development process for the Wearable Sensory Module, the Wireless Communication Module, the USB Interfacing Module and the system integration.

3.2. Development of the Wearable Sensory Module

There are two versions of prototype built using different sensors and ergonomic design. The first one is in the form of wearable glove and the second version is made into distributed wearable sensors which users can wear or attach to the body.

3.2.1. Version 1: Light Dependent Sensor Glove

The wearable device is presented into a glove. At each finger tip, a sensor fusion module is deployed using Light Dependent Resistance (LDR) Sensors. The microcontroller and wireless module are hidden at the side of the glove and battery is also installed to provide portable power supply. The 3-Dimensional model of the ergonomic design is drawn using 3dsMax and shown as Figure 10.

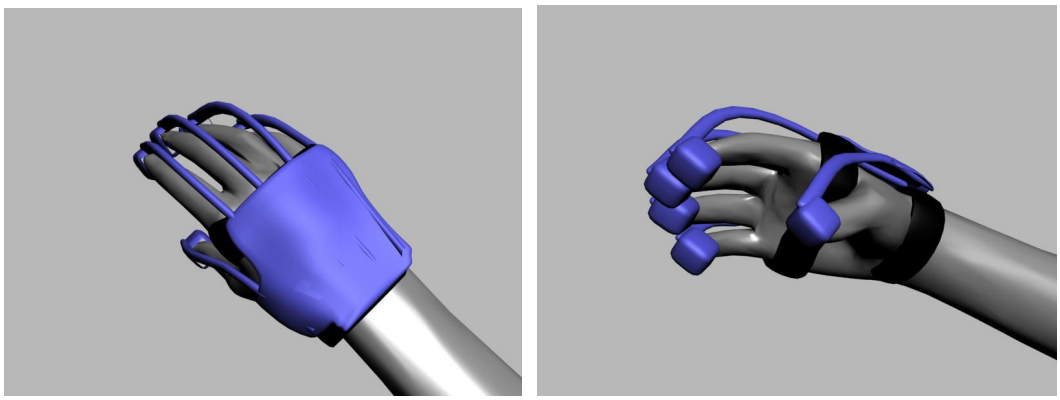


Figure 10: Top View and Side Perspective View

3.2.1.1. Microcontroller: the Computational Unit

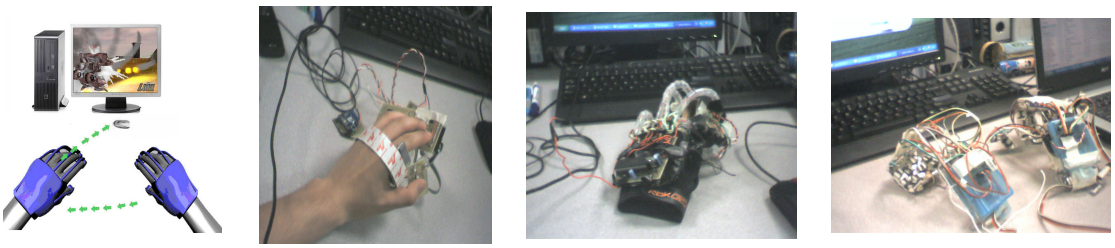
The microcontroller we used in the system design is PIC16F877. The microcontroller is connected to the sensor and read the analog input and then convert into digital data through the Analog to Digital Converter (ADC) module and then

further process the data controlled and interface to the wireless module. The firmware inside the microcontroller is attached in Annex A of this thesis.

3.2.1.2.Sensor Fusion Module

LDR sensors are used and system are further customized which have low power consumption and good reaction time, kept within 1 second.

Each finger will be given a sensor fusion module consisting of three sensors. Each of the sensors serves as the basis sensing unit which correspondingly output to one unique command to the machine; here we take computer for example. The sensor is touched based. Hence when the finger in the covered module reaches and makes contact with one specific touch sensor, the sensor will be triggered and output is made following the touch event and will be captured by the microcontroller. Bellow in Figure 11, it shows the three versions of the LDR Wearable Glove.



Version 1 Wearable Glove Version 1.1 Version 1.2 Version 1.3

Figure 11: LDR Solution Prototype

The first prototype Version 1.1 was built with an LDR sensor Printed Circuit Board (PCB) shown in Figure 11. When the finger covers the LDR sensor from the environment light, the LDR will change its resistance and result in the voltage change at the microcontroller input pin. Such trigger of event will be regarded as one user input. However due to the uncertainty of environment light, the Version 1.2 designs a

cover sensor fusion module, an additional light source was added as a stable light source to the LDR, as shown in Figure 12 such that the system can work regardless of the change of environment light. Then Version 1.3 of two handed module was built for both left and right hand.

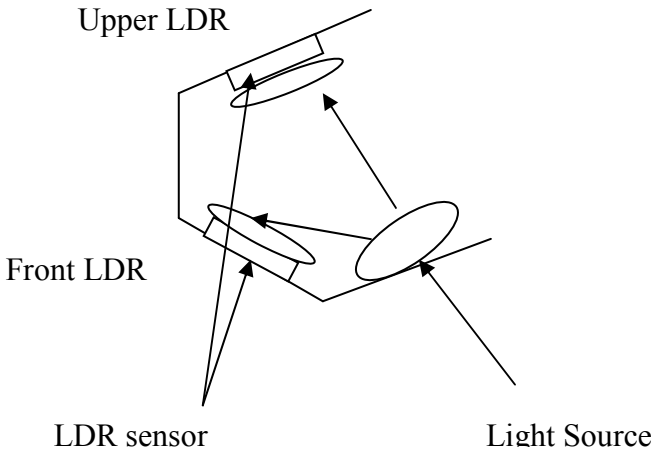


Figure 12: LDR Sensor Fusion Version 1.2

3.2.2. Version 2: Full Body Wearable Sensor System

Although the Wearable Glove is able to capture the finger gesture and free the user from the desk to type keyboards and dragging mouse, the interactivity, mobility and intuitiveness of the system still need to be improved by observing the user behavior. The user shall need to wear the two gloves every time before they use. Among the 20 participants of user test, 70% of the participants feedback that there is challenge to adjust the second glove after the user already put on the first glove as the first hand become inconvenient with the glove on. There is also space to improve on the mobility and interactivity as only finger gesture is engaged for the human machine interaction. Hence the Version 2 of the prototype is looking for a solution which can utilize the richness of the full

body motion and gesture and achieve a more convenient device for wear, carry and use.

With the design considerations discussed above, the Version 2 is developed into a wearable sensor system for full body motion capture. The same microcontroller as Version 1 is also engaged. The version 2.1 is featured in Figure 13. The User can wear maximum of such five devices at different parts of the body including wrist, ankle, head and waist. The Wearable Sensor System measures the accelerations. The computational unit samples the signals from the sensor circuits and conduct further signal processing, and finally recognize motion and gesture. Such information will be transferred to the target computer for Human Computer Interaction purpose.

Figure 13: Ergonomics Design of Wearable Sensor System

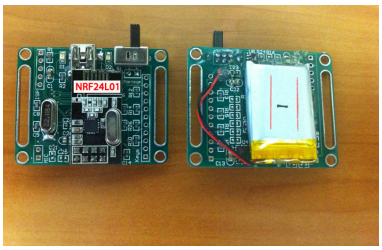


Figure 14: Electronics of Wearable Sensor System

The driver circuit is powered by a rechargeable Lithium-Ion battery pack with 250mAHr, which provides ten hours of safe operation of the device as the current drawn by the system is 20mA maximum.

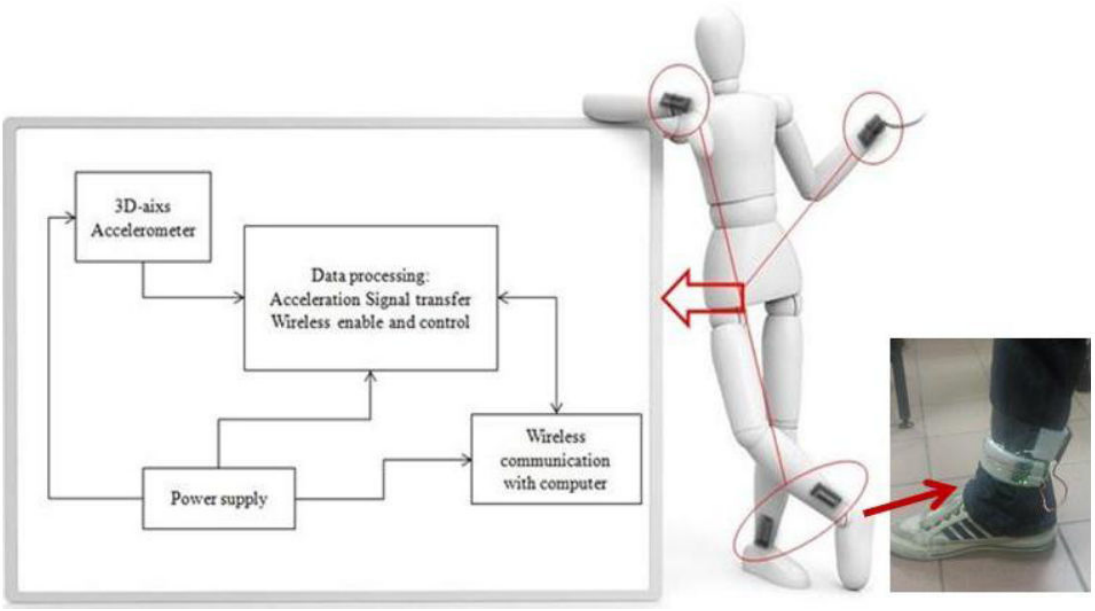


Figure 15: WRS sensor system

3.2.2.1. The Sensor Fusion Module

The acceleration sensing system is the basic system to capture the motion of human and transfer these data to the microcontroller in driver circuit. As this acceleration capturing module was attached on human body, the whole module should be small, light, and low power consumption as well as high performance. The accelerometer was the key components in this module. If the choice of accelerometer is not suitable, the requirements of the subsystem can hardly be approached. Consequently, low cost capacitive micromachined (MEMS) accelerometers could be one of proper candidates. They are named as capacitive accelerometers since the detecting mode of acceleration. The accelerometer can be modelled as a set of beams attached to a movable central mass that move between fixed beams. The movable beams can be deflected from their rest position by subjecting the system to acceleration. As the beams attached to the central mass move, the distance from them to the fixed beams on one side will

increase by the same amount that the distance to the fixed beams on the other side decreases. The change in distance is a measure of acceleration. The g-cell beams form two back-to-back capacitors. As the centre beam moves with acceleration, the distance between the beams changes and each capacitor's value will change.

$$C = A\varepsilon/D,$$

where A is the area of the beam, ε is the dielectric constant, and D is the distance between the beams.

This type of accelerometer is a successful commercial product which is widely used in the laptop, mobile phone, wireless mouse and even handhold equipments due to its stable performance, low power consumption, and miniature size which is shown in the figures. According to the system functional, a low-g high sensitivity three-axis MEMS accelerometer was employed in our design. As shown in figure 16, this accelerometer has a 1-pole low pass filter for each axis, temperature compensation and g-Select which allows for the selection among 4 sensitivities. Zero-g offset full scale span and filter cut-off are factory set and require no external devices. It also includes a Sleep Mode that makes it ideal for handheld battery powered electronics.

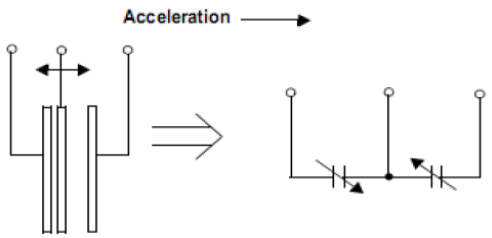


Figure 16: Simplified Transducer Physical Model

Figure 17: Accelerometers versus coin

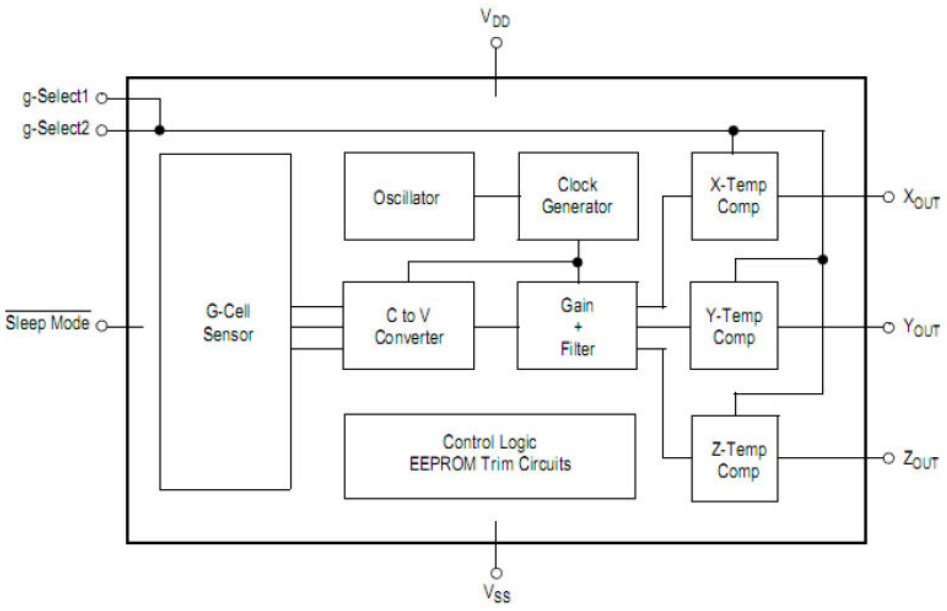


Figure 18: Simplified accelerometer functional block diagram

The circuits of acceleration capturing module was consisted of the accelerometer and related peripheral electronic components. The basic schematic of this module was shown in the figure 19.

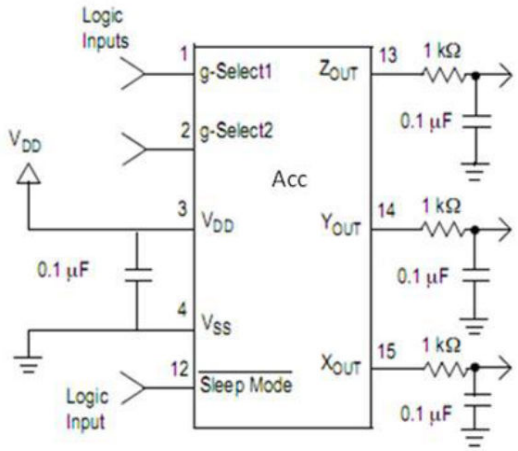
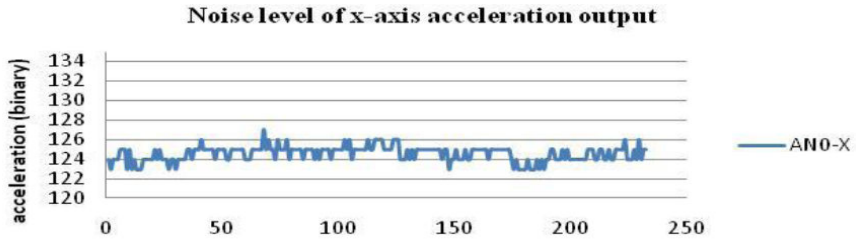


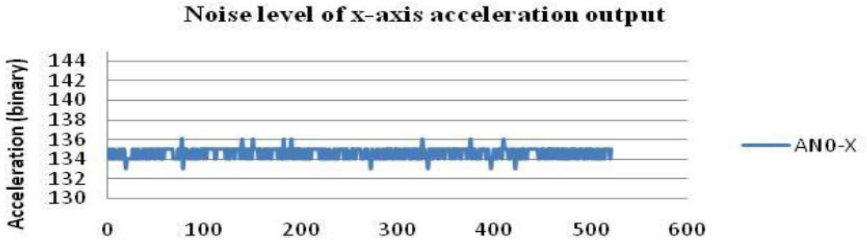
Figure 19: The basic connection diagram of motion sensing module

Although the accelerometer had internal low-pass filter and the peripheral bypass capacitors for power and output signals, the signal-to-noise ratio of output signal was not satisfied. Thus the value of bypass capacitor for DC power was increased from

0.1uF to 2.2uF, which can provide a less noise voltage supply into the accelerometer, and benefit to the output voltage. On the other hand, an amplifier with unit gain was utilized as a voltage buffer amplifier (voltage follower) is used to transfer a voltage from a first circuit, having a high output impedance level, to a second circuit with a low input impedance level. The interposed buffer amplifier prevents the second circuit from loading the first circuit unacceptably and interfering with its desired operation. According to the experimental data shown in figure 34 (b), the noise level was significantly improve when these two changes of hardware design had been done.



(a) Output noise level before hardware improvements



(b) Output noise level after hardware improvements

Figure 20: Data of noise level for acceleration

Comparing the sensor outputs before and after hardware development, it indicated that the improved PCB of acceleration module can provide a relative stable baseline for the acceleration data analysis.

3.2.2.2. Computational Unit Development

Our inertial subsystem and microcontroller works independently. The inertial sensors output signals keeps capturing motions and only stops working when powered off. At the same time, the accelerometers measure accelerations of movements. On the other hand, the microcontroller with each sensor board samples them with 8-bits analog-digital-converters (ADCs). In addition, a mean filter, a type of simple low-pass filters is applied to reduce the disturbances from small body vibrations and to provide a smooth wave. Finally, these processed data from the three accelerometer axes are encoded as a digital stream and then transmitted to computer via wireless networks. The sampling rate of the inertial data is 140Hz.

Since an 8-bit PIC microcontroller is used to process sensing data, the firmware work development tools are MPLAB® IDE Integrated Development Environment and the MPLAB C30 language suite. It is a free, integrated toolset for the development of embedded applications employing Microchip's PIC® and dsPIC® Microcontrollers. MPLAB IDE runs as a 32-bit application on MS Windows®, is easy to use and includes a host of free software components for fast application development and super-charged debugging. MPLAB IDE also serves as a single, unified graphical user interface for additional Microchip and third party software and hardware development tools.

The core of the driver circuit is the microcontroller chip from the PIC18 series, which is used for processing signal from sensor subsystem.

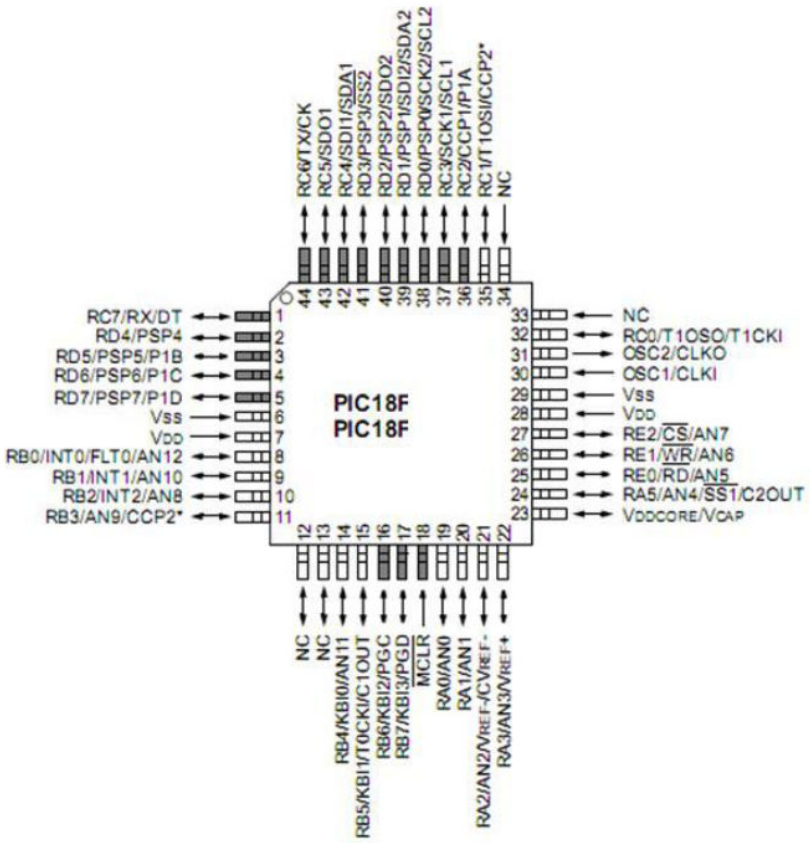


Figure 21: Pin diagram of 44-Pin PIC18 series MCU

The acquisition for the acceleration of movement uses ADC function of the PIC. ADC pins involved are. As the driver circuit and inertial sensing subsystem are in separated PCBs, the connections of these two parts are via signal wires. The signal processing is mainly control by firmware functions named *ADC_ACC* and *TX_ACC*. The *ADC_ACC* is used to configure the ADC registers to enable the analog-to-digital converting and define the pins of RA0, RA1, and RA2 to be analog pins. The *TX_ACC* is used to store the acquired data and control the transmittal of data to the computer.

For hardware design, the driver circuit in the main PCB which also includes the interface to inertial sensing subsystem, the transceiver module for wireless networks,

and the charging circuit for battery. In another words, this PCB includes the analogy signal, digital signal, high speed components and thermal components in a very small area, approximately 30mm×50mm. Therefore the PCB layout should be designed carefully with several considerations to ensure the signal integrity without interference between each function. Firstly, the PCB is divided into several areas by different functions, and the electronic components for each function should be place into corresponding area. Another important thing for avoiding EMI is the traces of ground. Despite the four layer PCB layout with one layer of ground plane and one layer of power plane, which has excellent capacity to keep signal integrity and avoid EMI, the ultimate PCB layout has 2 layers which are due to the much lower cost (half of the 4-layer PCB's). This double sides PCB also can be made to perform quite well with careful PCB design. And the key design for EMI compliance is loop area. A closed circuit of tracks comprises a loop. Interference can both exit and enter the PCB into areas inside loops. The smaller the dimensions of each loop, the smaller the magnitude of interference to be dealt with. The circuit on the board may be contained inside several small loops, through having a gridded power supply distribution. The best distribution of the lot is one or two continuous planes or sheets of copper, especially for high speed circuits. Therefore, in this PCB, two high speed (high frequency) parts: transceiver module and crystal oscillator for microcontroller, are enclosed by an overall ground. Additionally, no other electronic components place on the opposite side of these two parts to further avoid interference. The final thing should be mentioned is that the interface for the inertial sensing subsystem is routing

near one edge of this board isolated from other circuit at all to protect the most sensitive analogy signals against the interference.

The software used for hardware design is named Altium Designer, widely used EDA software. In early years, to deal the limitations imposed by separate stand-alone design tools, Altium Company developed a unified electronics design system, which uses a single data model to hold all of the design data required to create a product. A variety of editing tools could then be used to access and manipulate the design, covering areas such as board layout and design, schematic capture, routing (EDA), testing, analysis and FPGA design. And then they created its own platform called Design Explorer (DXP), hosted on Microsoft's Windows operating system, which formed the foundation of the Altium Designer product. The basic operation windows and function of Altium Designer is shown in the above figure. It can be seen that the *Viewer* supports the display of multiple design documents of differing type. Schematic, PCB, OpenBus, CAMtastic and OutJob documents are opened and viewed within their respective *Document Editors*. All other text-based document types are opened in the *Text Editor* for viewing. Documents, projects and design workspaces can be opened using the relevant commands available from the main file menu.

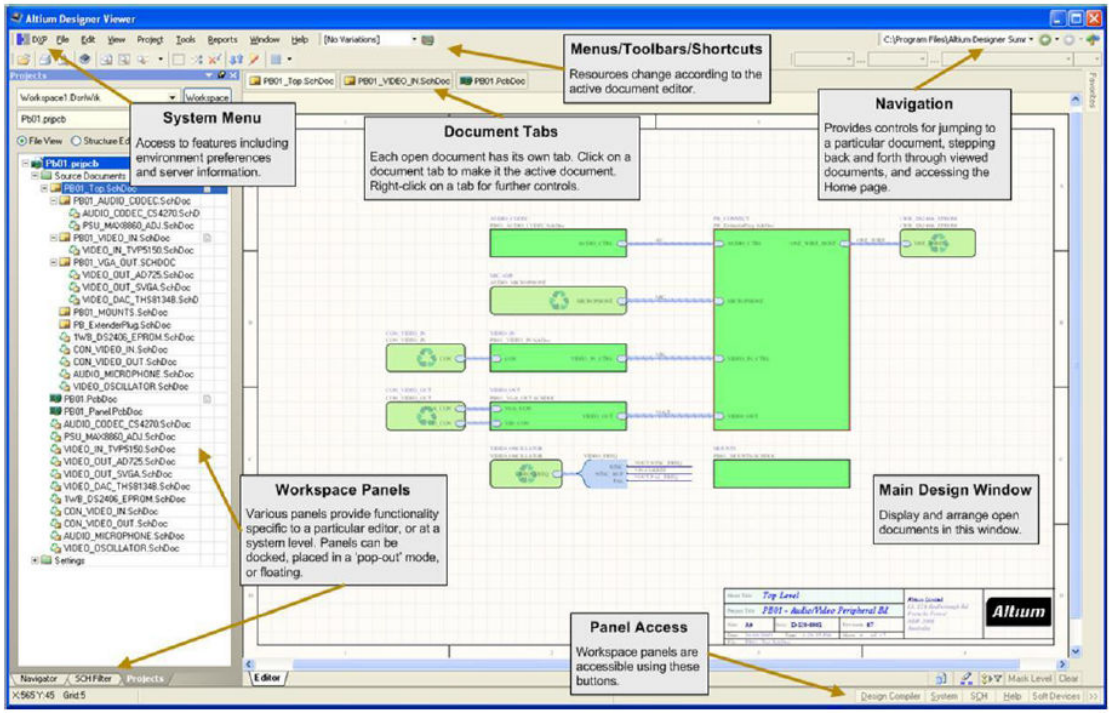
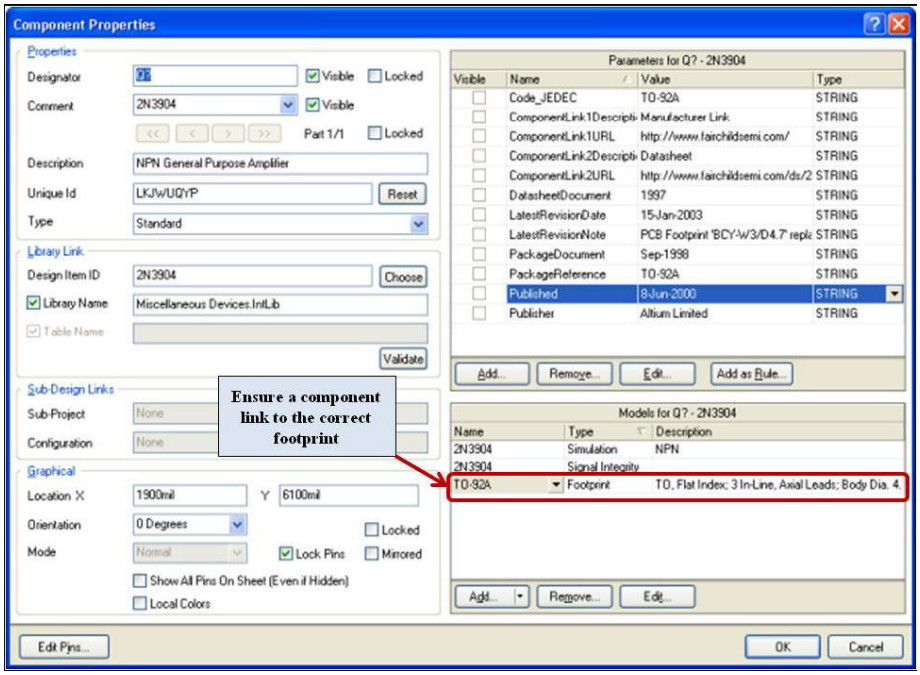


Figure 22: The key elements of viewer environment

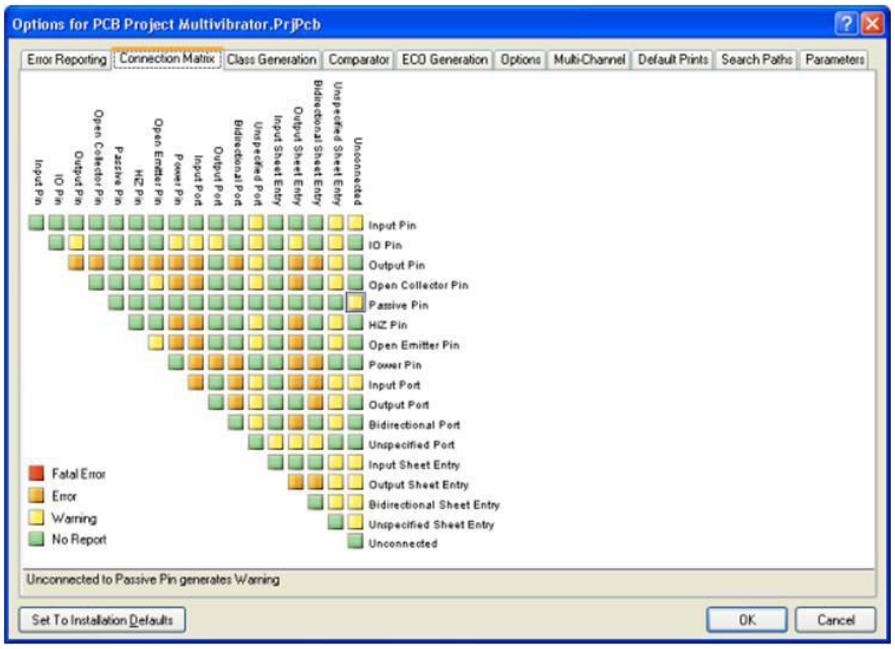
Alternatively, you can drag-and-drop a document, project file or design workspace file directly into the Viewer. A key ingredient of the Viewer's environment is workspace panels. Whether specific to a particular document editor or used on a more global, system-wide level, they present information and controls that allow you to efficiently interrogate a design.

Since the Altium Designer is a powerful yet user-friendly system development platform for PCB, FPGA, and embedded software development built on unique layered platform architecture. It was used to develop the hardware of our system, including schematic and PCB design. Therefore the design tools for schematic and PCB layout should be familiarized. For drawing schematic, two points should be highlighted: one is confirming that the Footprint is set to right one for every component, and the other is checking electrical properties according to the rules set up

in the Error Reporting. Both are fundamentals for providing a correct PCB layout.



(a) An example of footprint setting



(b) Electrical properties checking matrix

Figure 23: key point of schematic design

When the schematic is completed, the design needs to transfer from the Schematic

Editor to the PCB Editor. At this step a new PCB design should be created at least with a board outline. The following step is transferring design of schematic into PCB layout. The process of transferring a design from the capture stage to the board layout stage is launched by selecting *Design->Update PCB Document* from the menus. After the transfer, the components and nets will appear in the PCB workspace. Since the PCB Editor is a rules-driven environment, meaning that as the design changes, like placing tracks, moving components, or auto-routing the board, Altium Designer monitors each action and checks to see if the design still complies with the design rules. If not, then the error will be indicated immediately as a violation. Setting up the design rules before start routing allows designers to focus on designing, be confident in the knowledge that any design errors will immediately be flagged for your attention. The design rules fall into 10 categories, which can then be further divided into different design rule types. The design rules cover electrical, routing, manufacturing, placement and signal integrity requirements. It is a very useful tool to support PCB design.

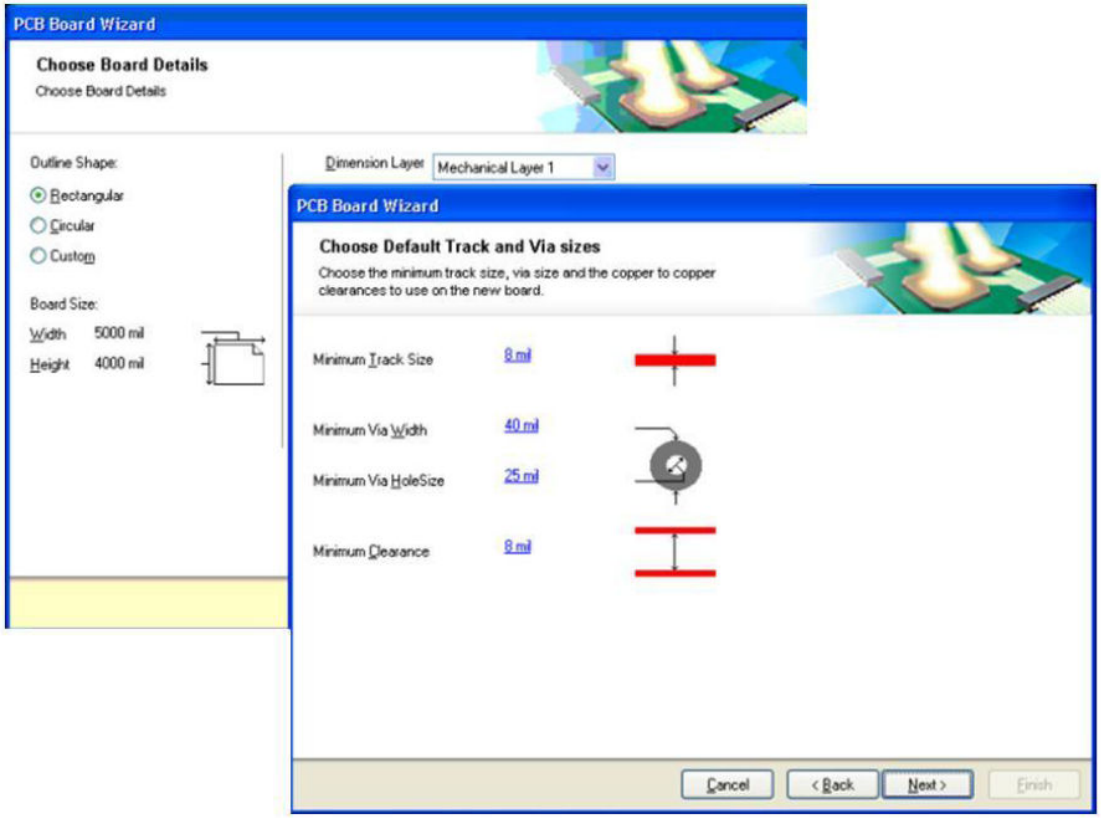


Figure 24: an example of using *PCB Board Wizard* to create a new PCB design file

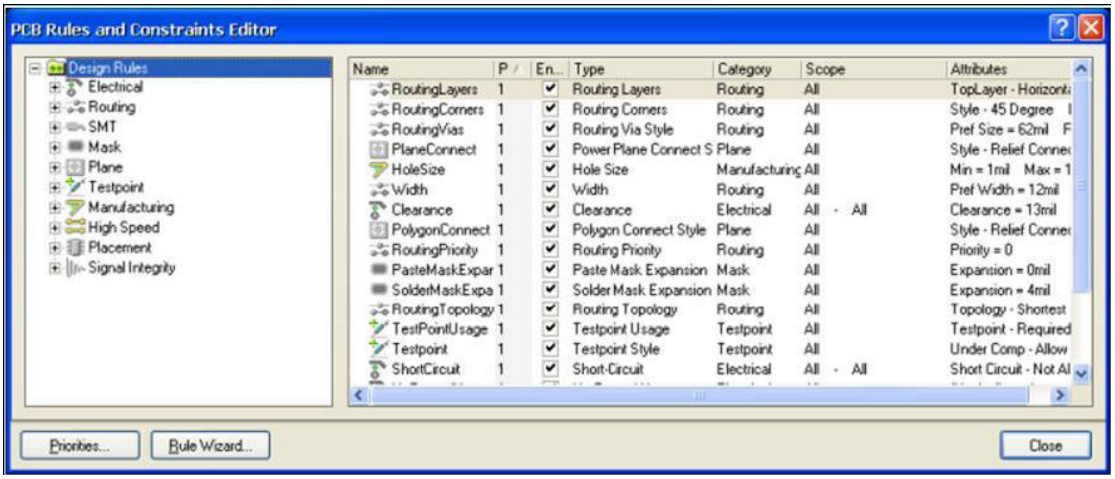


Figure 25: PCB Design Rule Editor Interface

Sometimes Printed Circuit Board (PCB) is considered as an older technology, which is decreasingly applied in related areas. This is simply not the case. PCB is used more than ever before, for almost every component, subsystem, and system in use because the transformation of modern lifestyle under way and is becoming a more connected,

electronic, and instrumented style. In the past, many systems were free of electronics, but the majority commodities may include electrical elements in the future. PCB is still playing a key role in electronics hardware used in communications, computer-controlled systems, and automatic control systems. Therefore for all electronics to perform interconnection, PCB is required.

Because of wide range of applications, PCB is a complex and highly adjustable technology, from millimetres to tens of millimetres in size. The thickness, layers, types, and numbers of interconnections, materials, and chemistry mean that the full range of capabilities and technology is difficult for a facility to be able to produce. This complexity is very attractive for commercial performance in that it gives designers various options.

Because the requirement of maintaining legacy systems based on the concern of increasing marketing, the range of complexity needed to fulfil present, and future needs grows exponentially. On the other hand, as the trend of miniature in commercial products, the density of interconnections and components in small area becomes higher. This complexity with increasing EMI induces more challenges for the PCB design.

This entire picture is overlaid with the need for more secure, robust, and reliable PCB. New applications will require longer life, higher reliability under demanding environmental conditions, protection against tampering.

From electrical and mechanical aspects design for PCB, the electrical characteristics of PCB and multichip electrical connection substrates have become a critical functional product definition, and design requirement, for many electrical and electronic products. Until the late 1980s, most PCB designs were printed writing designs, in that, with the exception of power and ground distribution, component placement and the arrangement of conductive and nonconductive patterns were not critical for functional electrical requirements. However, since the late 1980s, electrical signal integrity has become a more serious design consideration in order to meet both functional performance and regulatory compliance requirements.

Electrical signal integrity is a combination of frequency and voltage/current, depending on the applications. In low level analog, very small leakage voltages or currents, thermal instabilities, and electromagnetic couplings can exceed acceptable limits of signal distortion. In a similar manner, most digital components can erroneously switch by application of less than 1 V of combined DC and AC signals.

The design of some analog PCB is a critical balance of all of the known parameters and characteristics of the complete design-through-use product development, manufacturing, assembly, test, and use processes. Analog designs cover all or portions of the complete electromagnetic spectrum, from DC all the way up into the GHz range of frequencies. Active and passive electronic components and materials have various levels of sensitivity to operating environments and conditions, like temperature, thermal shock, vibration, voltage connections, and especially analog signal grounds are critical to analog integrity.

One of the key methods to improve analog signal integrity is to separate the more critical or sensitive portions of the design. Sensitive circuit may be susceptible to one or more external forces, such as electromagnetic, voltage, and grounding systems, mechanical shock/vibration, and thermal. Sometimes the more sensitive circuit is repackaged into a separate function and module that provides its own isolation and separation from the offending condition. Isolation and separation can be provided by physical separation, electromagnetic and thermal barriers, improved ground practices and design, power source filtering, signal isolators, shock and vibration dampeners, and elevated or lowered temperature controlled environment.

Below a few Milli-volts, thermal electromotive forces (EMFs) have a significant impact on low-level analog signal integrity. Thermal EMFs of a non-symmetrical sequence of various metal junctions (conductors) or symmetrical sequence of various metal junctions operating at different temperatures will generate and inject undesirable voltage (or induce unwanted currents) into the electrical signal path. This thermocouple effect is desirable in the case of temperature measurements. However, in the case of other low-level measurements it is an undesirable characteristic. Therefore, the requirement for low-signal level PCB is to ensure that all components and electrical interconnection networks and corresponding electrical terminations (such as soldered, welded, wire-bonded, or conductive adhesive) are symmetrical and isothermal. Electrical components, such as thin/thick-film resistors of different values (resistance), may have resistor elements manufactured from different formulations or

compositions of materials and will have a designed-in thermal EMF error due to component selection.

Each digital logic family of integrated circuits (ICs) has manufacturer-specified electrical operating parameters and signal transfer characteristics, many of which have become industry standards due to multiple manufacturing of the family of components. The electrical signal integrity requirements for digital ICs are primarily the high and low electrical (voltage/current) requirements for the output, input, clock, set, reset, clear, and other signal names; the signal rise/fall times, clock frequency and setup/hold times; and the voltage and ground connections as are necessary for the control and operation of the IC. The input, output, and electrical signal transfer parameters for digital ICs vary from logic or microcontroller family to family. ICs are large matrix of components, consisting of the semiconductor substrate materials, such as silicon, silicon-germanium, and gallium arsenide, which make up the various types of transistors. The large number of digital IC families available creates a complex matrix of design issue and requirements. The electrical signal integrity of the rise and fall times of electrical signals is a major driver and concern for high-speed and high-frequency circuits

Electromagnetic compatibility (EMC) is a serious design requirement for both functional performance to design and regulatory compliance requirements. EMC encompasses the control and reduction of electromagnetic fields (EMF), electromagnetic interference (EMI), and radio frequency interference (RFI) and covers the whole electromagnetic frequency spectrum from DC to 20 GHz.

Worldwide, the electronics industry has had to pay increasing attention to EMC to comply with both national and international standards and regulations. EMC involves major design considerations that ensure proper function within the electronic component, assembly, or system in order to limit the emission (radioactive or conductive) from one electronic component assembly, or system, to another, and reduce the susceptibility of an electronic component, assembly, or system to external sources of EMF, EMI, or RFI.

Good design requires up-front determination of a noise budget, which should be included in the product definition requirements. The noise budget is the summation of all of the DC and AC voltages/currents that from a boundary within which the component, assembly, or system is designed to function. The DC noise budget consists of the voltage settings of the power supplies, the operating tolerance of the power supply, and the series DC voltage drops of the voltage distribution system. The AC noise budget consists of the effectiveness of the local bypass capacitor, the amount of decoupling between the load, the bulk decoupling capacitor, and the power distributions system, the local voltage drops in the component's voltage/ground conductors, and the component's input voltage tolerance. As mentioned in the previous, many of the operating electrical, mechanical, thermal, and environmental parameters and conditions can have influence on the noise budget. Additional noise that may need to be considered are EMC radiated and conducted emissions from other electromagnetic equipment and thermocouple effect due to electrical connections with differing layers of metals operating at different temperatures.

3.3. Development of Wireless Cross-Platform USB Interface Module

From the above discussion, the standalone wearable robotic sensor is able to recognize the motion and gesture. If all the sensors are connected through wire and then connect again to computer using cable, the mobility and portability will be much compromised. A wireless wearable sensor system based on ZigBee RF2.4G technology is built. The system should support five remote wearable robotic sensors working concurrently and be able to establish communications between the five sensors and the target computer.

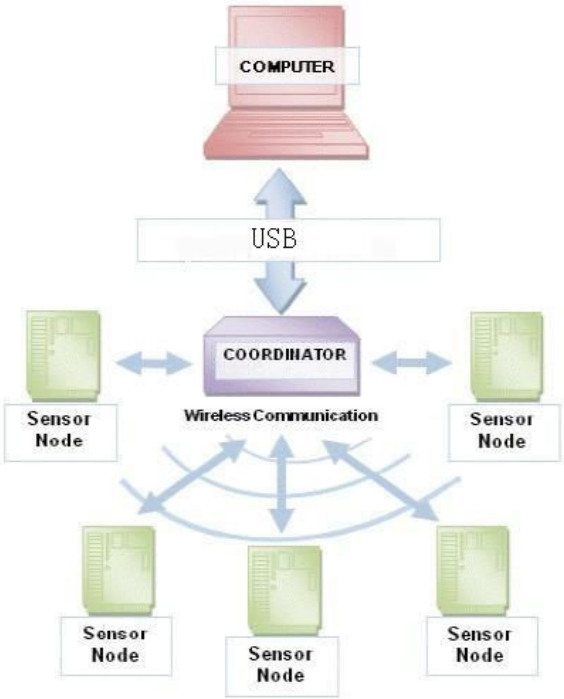


Figure 26: Wireless Sensor Network

Figure 26 shows the configuration of the system. From the figure, it can be observed that the system consists of five wearable robotics sensor defined as sensor node that will communicate wirelessly with the coordinator based on a star network

configuration. The coordinator is in turn connected to a personal computer via a USB device which we will further discuss on in later session.

Each sensor node has an on board Microcontroller and some sensors such as accelerometer and gyro. For the wireless communication, both the sensor node and coordinator use 2.4 GHz RF.

3.3.1. Selection of Chip Model

As discussed in the Literature Review session, the Zigbee RF2.4G technology is selected for the wireless communication. In order to choose the optimized chip, a comparison is made between the most popular models on the market as shown in figure bellow.

Model Name	NRF24L01	RF2915	BC418	XC1201	CC400
Voltage	1.9~3.6V	2.4~5V	2.5~3.4V	2.4~5.5V	2.7~3.3V
Maximum Output power	+4dBm	+5dBm	+12dBm	+5dBm	+14dBm
Speed	2Mbps	9.6Kbps	128Kbps	64Kbps	9.6Kbps
No. of Antenna	1	1	2	2	1
Supporting electronics components	14	50	50	20	25

Figure 27: Comparison of different wireless chips

In consideration of cost and performance, the chip of NRF24L01 is chosen for its lower voltage, less supporting electronics components required with highest speed. The current drawn when works with -6dBm output power, the current drawn of the transmitter is measured only as 9mA and current drawn for the receiver is only 12.3mA.

Based on the schematic diagram as shown in Figure 28, the SPI communication is chosen for the interface between wireless chip and microcontroller.

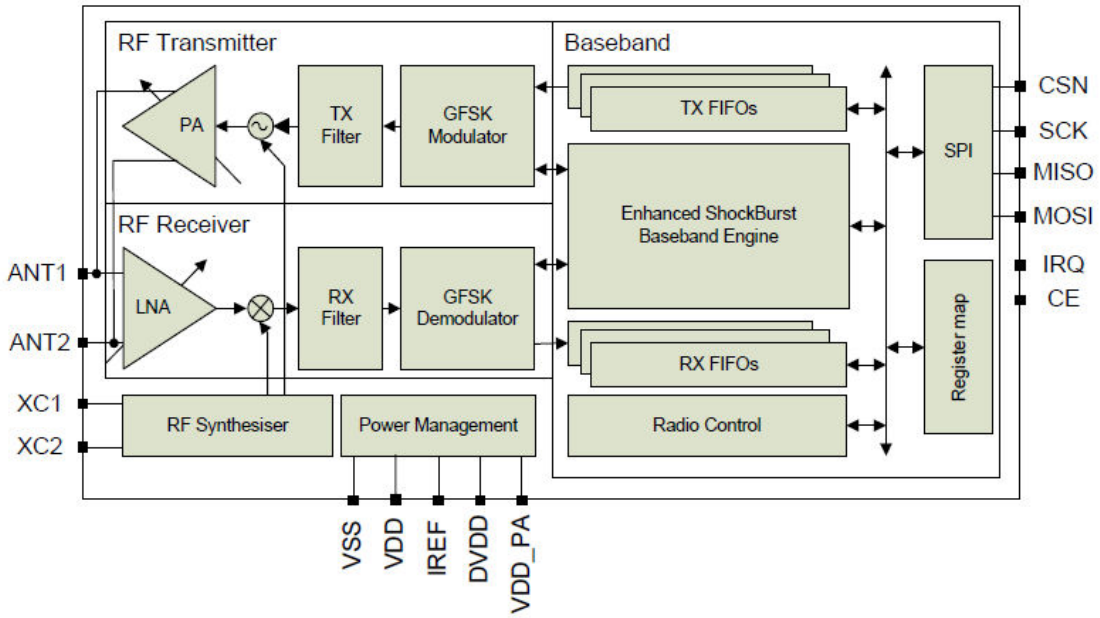


Figure 28: Comparison of different wireless chips

We discussed the development of the wearable robotic sensor module and wireless communication module. At the transmitter side, the wireless module is interfaced with microcontroller of PIC16F which read MEMS output and send the sensor data to the receiver wirelessly. In this session, we will discuss the development of a USB cross-platform module inside the microcontroller C8051F321 which interface with the wireless receiver mentioned in session 3.2.2. The purpose is to develop such a plug and play USB device that user can easily insert into computer and interpret the sensor data received from wireless module into standard Human Interface Device data to control the computer.

3.3.2. Circuit design

Microcontroller C8051F321 is selected to interface with the NRF24L01 chip. In next session, further discussion on development of HID functions within C8051F321 will be made. This part, we only focus on developing the circuit with C8051F321 to interface with wireless chip NRF24L01.The circuit for the wireless communication module is constructed in Altium Designer. The design process includes schematic design and circuits routing. The schematic design is illustrated in Figure 29a with connecting the resister, compactor and diode as requested. The pin connection is summarized in Figure 30.

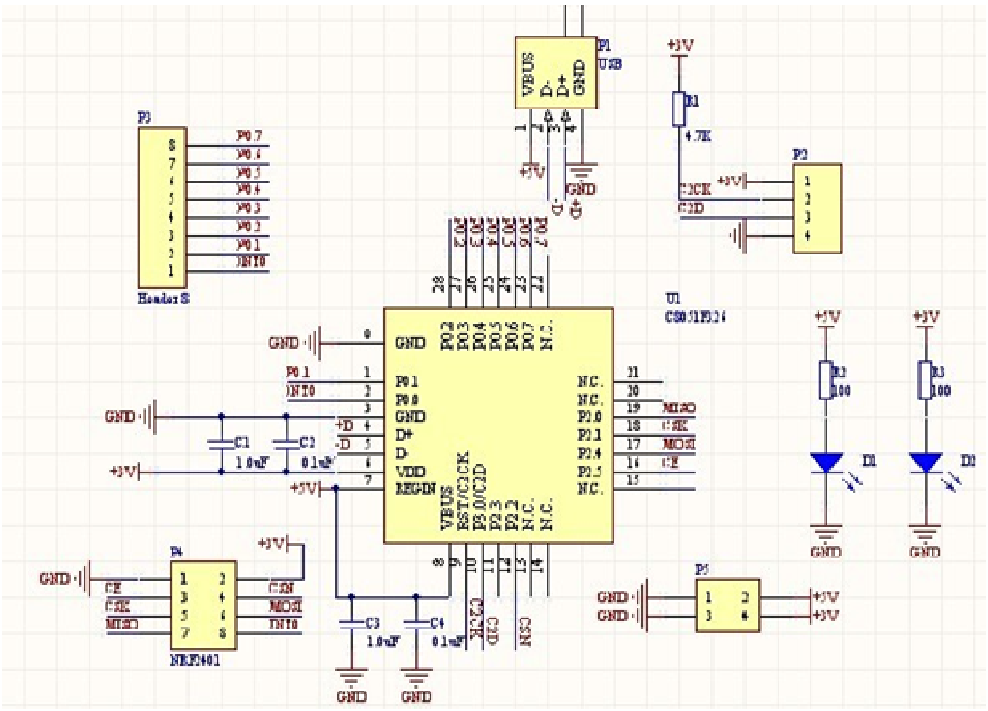


Figure 29a: Schematic design of USB receiver

Pin	Name	IO	Set up in this project
1	CE	Digital input	RX or TX selection
2	CSN	Digital input	SPI mode
3	SCK	Digital input	SPI timer

4	MOSI	Digital input	SPI digital input
5	MISO	Digital output	SPI digital output
6	IRQ	Power out	Not connected
7	VDD	Power	+3V
8	VSS	Ground	Grounded
9	XC2	Analog output	Oscillator
10	XC1	Analog output	Oscillator
11	VDD_PA	Analog output	Output power to RF amplifier
12	ANT1	Antenna	Antenna
13	ANT2	Antenna	Antenna
14	VSS	Ground	Grounded
15	VDD	Power	+3V
16	IREF	Power in	Reference current
17	VSS	Power	Grounded
18	VDD	Power	+3V

Figure 29b: Pin configuration for NRF chip

The next step is completing the circuit routing; the two major considerations are board size for portability and electromagnetic interference. In order to minimize the electromagnetic interference to the signal, the analog and digital rout is separated. And the final routing is shown in Figure 30.

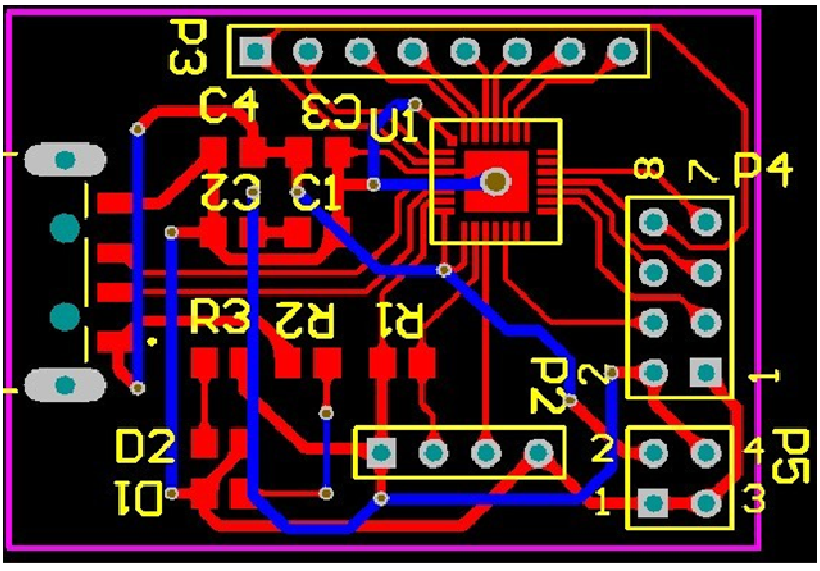


Figure 30: Routing of USB device

After printing the circuit board and soldering the components, the prototype is shown as Figure 31.

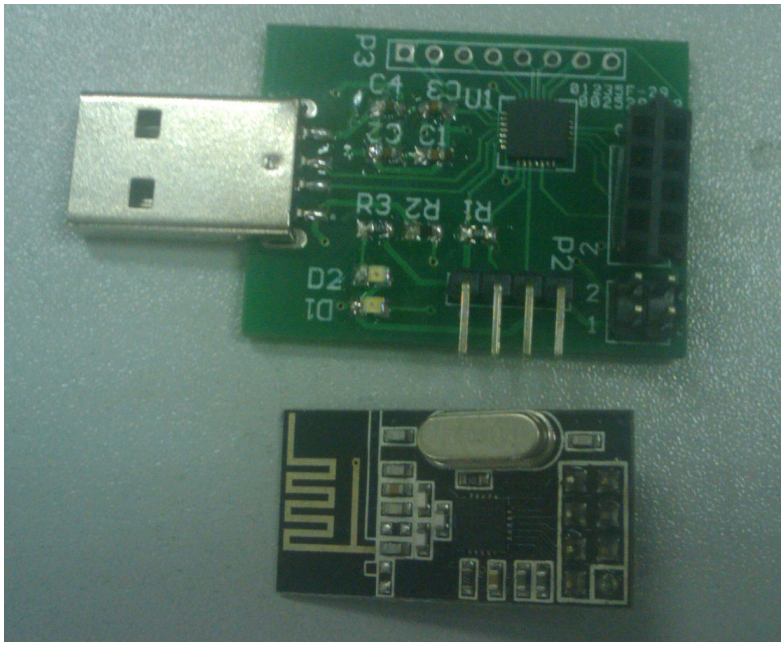


Figure 31: Actual Board of USB Receiver

3.3.3. Firmware Development

The standard USB devices that we commonly used in daily life include keyboard, mouse, printer and thumb drive. As discussed in the literature review session, we aim to develop a plug and play USB device that does not request user to install any system driver additionally. USB specifications include class specifications which operating system vendors optionally support. Examples of classes include Audio, Mass Storage, Communications and Human Interface (HID). In most cases, a driver is required at the host side to 'talk' to the USB device. In custom applications, a driver may need to be developed. Fortunately, drivers are available for most common host systems for the most common classes of devices. Thus, these drivers can be reused. And base on such system characteristics for Windows, Linux and Apple iOS, we select reusable driver which by default is installed on user's computer such that the USB device in this project does not request any driver.

3.3.3.1. Device Configuration Level Design

USB device functionality is structured into a layered framework. Each level is associated with a functional level within the device. The highest layer, other than the device, is the configuration. Hence in the development, we selected a standard HID configuration and emulate the USB receiver as a HID device. A HID device may have multiple configurations; for example, a particular device may have multiple power requirements based on Self-Power Only or Bus Power Only modes.

3.3.3.2. Interface Level Design

For each configuration, there may be multiple interfaces. Each interface could support a particular mode of that configuration. And in this project, we define three interfaces including a keyboard interface, a mouse interface and a raw HID data transmitting interface.

Below the interface is the endpoint(s). Data is directly moved at this level. There can be as many as 16 bidirectional endpoints. Endpoint 0 is always a control endpoint and by default, when the device is on the bus, Endpoint 0 must be available to configure the device.

Information communicated on the bus is grouped into 1 ms time slots referred to as frames. Each frame can contain many transactions to various devices and endpoints.

3.3.3.3. Descriptor Design

Descriptors design is the core of the USB cross platform design process. It ultimately specifies the data package after the device configuration and interface level is defined. There are eight different standard descriptor types. For the three device descriptors, we follow the standard data package of mouse and keyboard for the first two and design a customized package for the HID raw data package to transfer our sensor data into computer through USB HID port. The code is attached in Annex B of the thesis. And the design of the descriptor for the HID customized data package with 8bits data size, value ranging from 0 to 255, can be summarized as bellow.

Descriptor Parameter	Design Value	Design Discussion
Usage Page	0x06, 0x00, 0xFF,	Set Vendor Defined Page
Usage (Vendor Usage 1	0x09, 0x01,	Default
Collection (Application)	0xA1, 0x01	Default
Usage Minimum	0x19, 0x01	Default
Usage Maximum	0x29, 0x0B,	Set 64 input usages total (0x01 to 0x40)
Logical Minimum	0x15, 0x00,	Sensor data bytes in the report may have minimum value = 0x00)
Logical Maximum	0x26, 0xFF, 0x00,	Sensor data bytes in the report may have maximum value = 0x00FF = unsigned 255
Report Size	0x75, 0x08	Set 8-bit field size
Report Count	0x95, 0x0B	Make sixty-four 8-bit fields
Input	0x81, 0x00	(Data, Array, Abs): Instantiates input packet fields based on the above report size, count, logical min/max, and usage

Figure 32: Descriptor Design

3.4. Prototype evaluation and Gesture Reorganization

3.4.1. Experiment

3.4.1.1. System Integration

Based on the development from session 3.2 to 3.3, the overall system is integrated as illustrated in Figure 33. The MEMS sensor is interfaced with PIC16F897J50 which again connect with wireless transmitter NRF24L01. The wireless signal is received by the receiver chip of a paired NRF24L01 which is interfaced with microcontroller C8051F321. Microcontroller C8051F321 is programmed into a combinational USB plug and play device including a keyboard device, a mouse device and a customized HID compliant data package device.

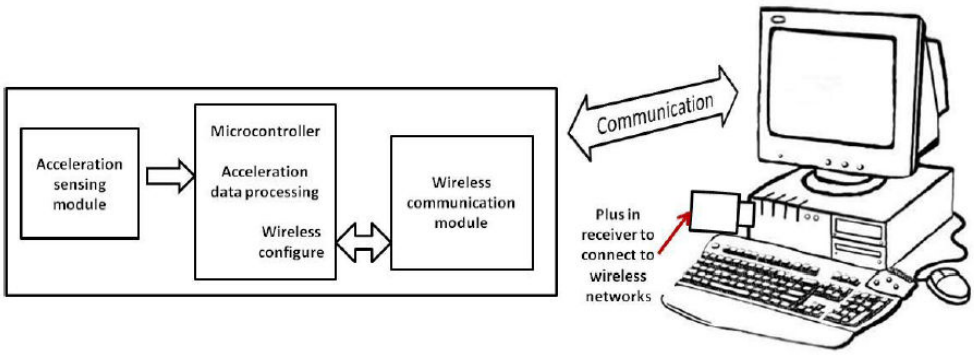


Figure 33: System Integration

3.4.1.2. Hardware Setup

On the receiver side, once plugging the USB receiver to computer, three devices on Windows Device Manager will pop up as designed. When the device is connected to the Windows, the two reporters will interface with HID driver will be recognized as a

HID compliant device, and further be recognized as three devices of keyboard, a mouse and a customized HID device for data transfer as shown in Figure 34. In this evaluation experiment, we will only use the HID device for data transfer to analyze on the sensor output for gesture estimation.

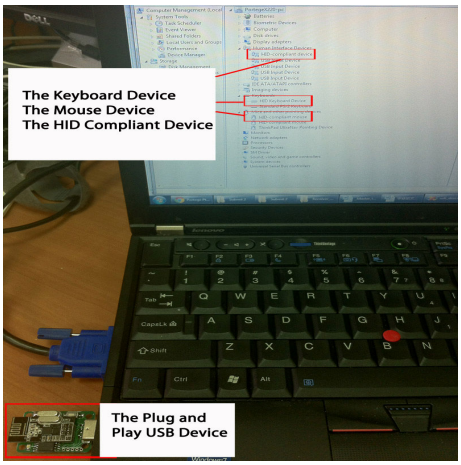


Figure 34: Interfacing with Windows on Personal Computers

On the transmitter side, user needs to wear the sensor on one part of the body. For the illustration purpose, the user in this setup wears the sensor inside the wrist band to both of the hands for arm lifting gesture estimation. The sensor is facing toward back of the hand as shown in Figure 35.

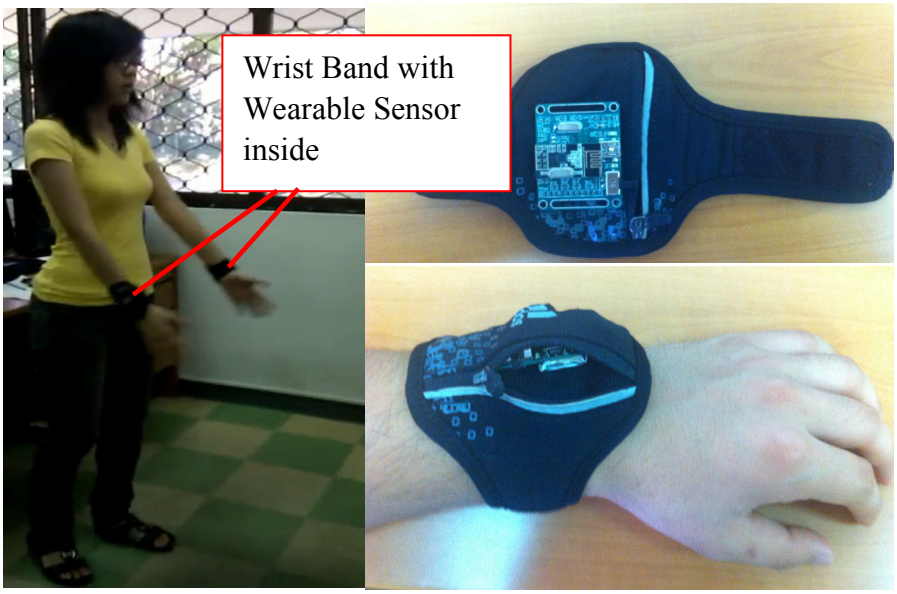


Figure 35: Wearable Sensor Evaluation Set Up

3.4.1.3. Software Setup

The software we programmed is in C++ and based on Win32 library environment. The software finds the HID device through vendor ID and Device ID. Subsequently it read the data into an 8 bit buffer and print it onto screen. Data is also written to a excel file. The code of the software is attached in Annex C of the thesis.

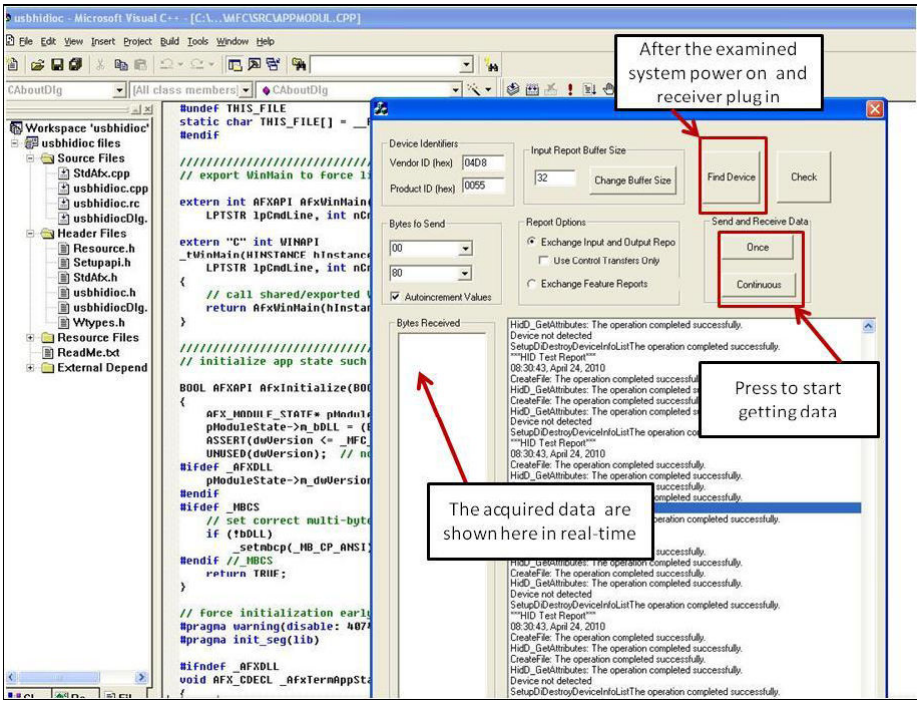


Figure 36: the testing software operation

3.4.2. Sensor Output and Gesture Recognition

3.4.2.1. Mathematical Model for Gesture Recognition with the Inertial System

The experiment will investigate basic motion and gesture of arm. The left arm starts from relax position when hand is put down and lift up toward front to the top, in the

process the arm keep straight, the whole process takes 10 seconds in a estimated constant speed

In the mathematical model, we define the following parameters:

1) Define Arm length as l;
2) When the arm is naturally relax downward, define the initial position of the hand where the sensor as 0, hence when the arm is fully lift up straight, the hand is at position of $2l$;
3) Define the angle between the arm and body which keep straight as a , and the vertical position relative from the initial position as h, at time t_1 ;
4) The gravity is denoted as g;
5) The direct reading of the output on x, y and z axis is X, Y and Z
6) We denote the sensor output value when the gravity projection is 0 as δ

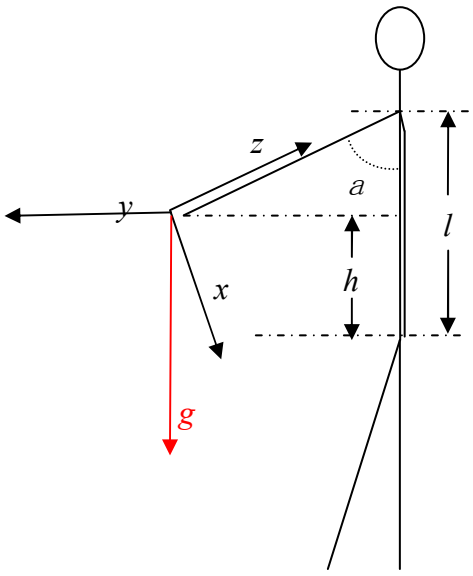


Figure 37: Model for Gesture Recognition

Hence Gravity projection on z axis

$$g_z = g\cos\alpha \tag{1}$$

Gravity Projection on y axis

$$g_x = g\sin\alpha \tag{2}$$

Gravity Projection on y axis is 0, since the x axis is always perpendicular to the gravity;

And it can be derived from the Figure 37 that

$$l - h = l\cos\alpha \tag{3}$$

From equation (1), it can be derived that $\cos\alpha = \frac{g_z}{g}$, substitute into (3) such that

$$l - h = l \cdot \frac{g_z}{g} \tag{4}$$

Noted that the g_z is the gravity projection at the z axis.

Condition: We assume at the end of the motion, the arm is still, meaning the velocity is 0 and the acceleration is also 0. Hence on the sensor output on the z axis, the only output is result from the gravity projection.

The output g_z can be further presented as:

$g_z = (Z - \delta)\cdot\lambda$, where Z is the output value directed read from the sensor, and Z is a digitized value within the range from 0 to 255, λ is the digitization coefficient of the accelerometer which is a constant

And substitute into (4), finally we have

$$l - h = l \cdot \frac{g_z}{g} = \frac{l\cdot\lambda}{g}\cdot(Z - \delta) \tag{5}$$

Hence it can be observed that the position *h* of the arm which wears the sensor is in a liner relationship with the sensor output with an coefficient of $\frac{l \cdot \lambda}{g}$

3.4.2.2. Experiment Data and Discussion

To investigate the validity of the proposed mathematical model, the experiment measures sensor output in relationship with the time and position of the arm.

The data recorded from the software as mentioned in 3.4.1 include the readings from the 3-axis of the accelerometer in the time interval of 10 seconds.

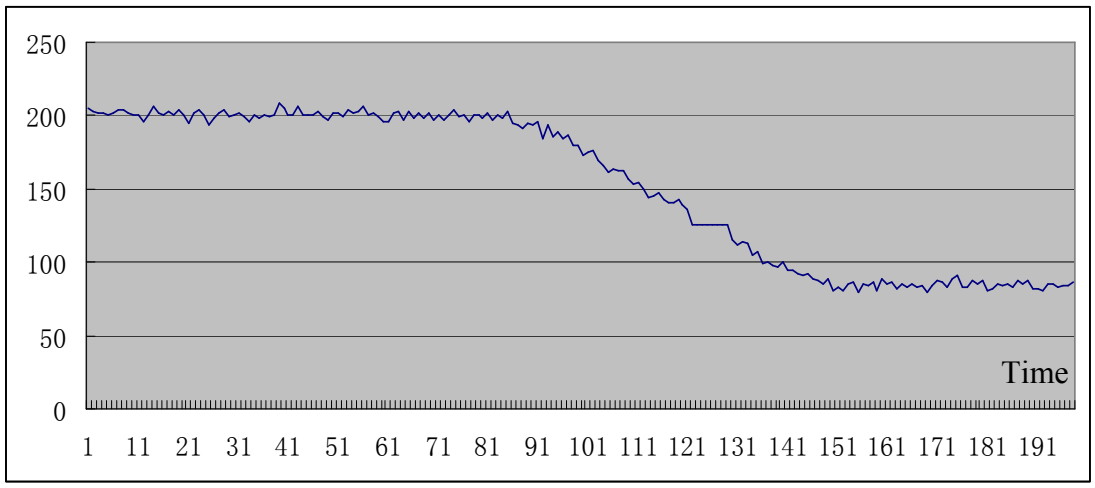


Figure 38: Accelerometer *z* axis sensor output

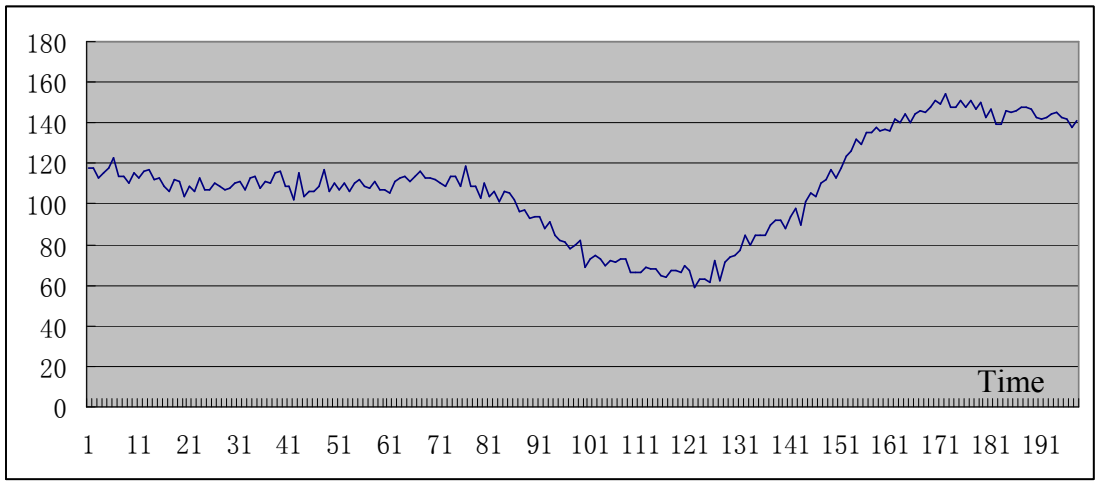


Figure 39: Accelerometer *x* axis sensor output

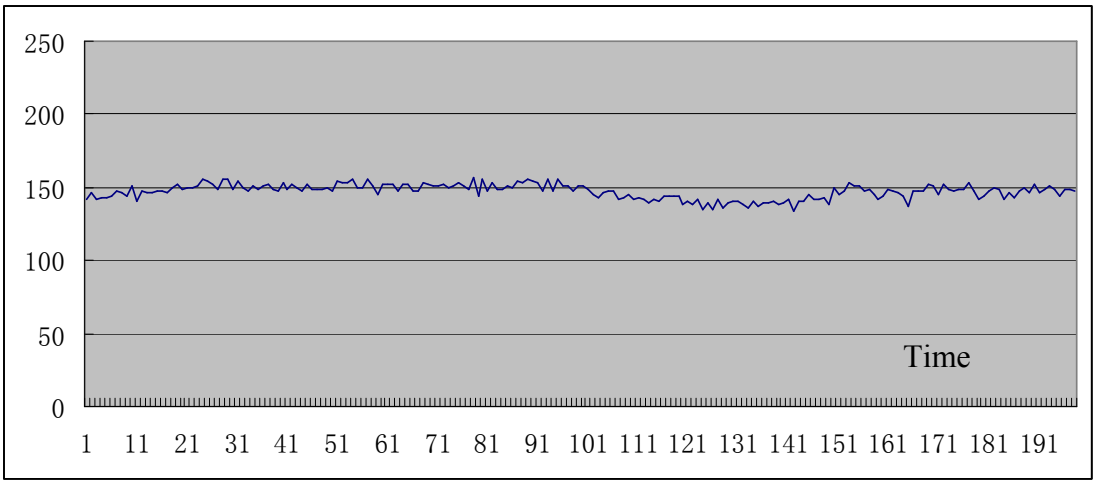


Figure 40: Accelerometer *y* axis sensor output

From above, it can be observed that z axis value keeps reducing in a linear relationship and saturates at minimum value when hand reach the top. This match our mathematical model in above session. The y axis output keeps the same within noise range. The x axis value start from a fix value goes to a minimum when the hand reaches the middle of the motion and then recover to the initial value when the hand reach the top. The experiment results match the discussion the in the literature review part where in the inertial system, the tilt of the sensor when the arm lift up result in projection of gravity into different axis of the accelerometer.

Find the Coefficient

And next, it is to examine the coefficients to the relationship between the sensor output and the orientation of the arm to estimate the gesture.

We take the mean of the initial value which is 200 when the gravity is fully projected on the positive axis and the hand is at 0 position, and take 80 as the value for gravity is fully projected to the negative axis when hand is at 2*l* position., Sub the two values into the equation $l - h = l \cdot \frac{g_z}{g} = \frac{l \cdot \lambda}{g} \cdot (Z - \delta)$

$$l - 0 = (200 - \delta)\frac{l \cdot \lambda}{g}$$

$$l - 2l = (80 - \delta)\frac{l \cdot \lambda}{g}$$

Combined the above two equations, we can find out that

$$\begin{cases} \delta = 140 \\ \frac{\lambda}{g} = \frac{1}{60} \end{cases}$$

And the relationship of the position of the arm and the sensor output can be derived as:

$$h = l(1 - \frac{1}{60} \cdot (Z - \delta)) \qquad (6)$$

3.4.3. Code implementation and Simulation

In order to visually simulate the experience result, the equation (6) is scripted using C# language and the value of *h* is used to control the arm gesture of a virtual avatar. The code is attached in Annex D of the thesis. The virtual avatar is a men character inside a game named Garden Mania developed by Portege Pte Ltd. The simulation result is shown as bellow:

Position 1 Position 2 Position 3

Position 4 Position 5 Position 6

Position 7 Position 8 Position 9

Figure 41: Arm Raise Simulation Using Game

From the above simulation, it proved that the linear relationship from the sensor output to the arm position is accurate. The player in the simulation program is able to move both of the arms wearing the sensor smoothly and control the virtual avatar to do a one-to-one mapping motion.

Chapter 4. Interactive Digital Application Development

In this part, an interactive Taichi game played by WRS was developed for users to do exercise. This part will only focus on the engineering development procedure of Taichi game and investigate on the game soaring mechanism assessed by WRS motion capture capability.

The Taichi game is currently deployed in Singapore General Hospital for trail and serves doctors as a research platform on investigating how playing an interactive Taichi game can influence the rehabilitation. The future work of this project will goes in-depth study on the rehabilitation application of WRG.

As shown in Figure 42, the user will wear the WRS sensor on the wrists and motion data can be captured and feedback to the game system. Scores will be given to player to evaluate their game play performance. The game of ’Taichi’ is a 3D game developed with Autodesk Maya [38] and Unity 3D in a team of designers and programmers with different skills and expertise in game creation. A game development pipeline explaining the production of the game from concept to release is presented. In addition, challenges faced and experiences gained during the production of ‘Taichi’ are also included in this document.

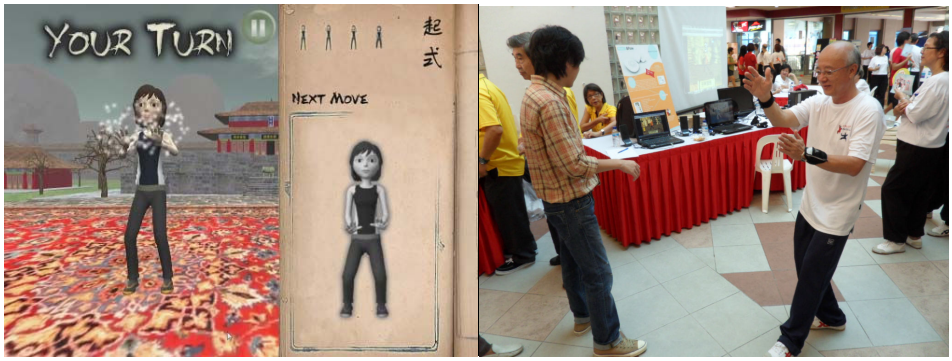


Figure 42: Interactive Tai Chi game played by WRS by Testers

More importantly, the cross platform feature of WRS could serves as an external plug in to the existing game engine of Unity3D and seamlessly capture user's natural input rather than through keyboard or joystick. Figure 43 shows the whole system architecture of the game. The game pipeline manages well on the graphics asserts, sounds and logic scripts, and HID classes serves as a plug in to interface WRS to the game. Consumers expect to enjoy a more realistic and better graphic games, therefore, the standards for graphic and realism of games needs to be improve too. This includes the improvement of modelling, texturing, rendering, lighting, more realistic animation and artificial intelligence behaviour and the overall content. Thus, the role for artists and animators is very important in the developing of the games, as their works can lead to the success of the game and the growing of the game studio.

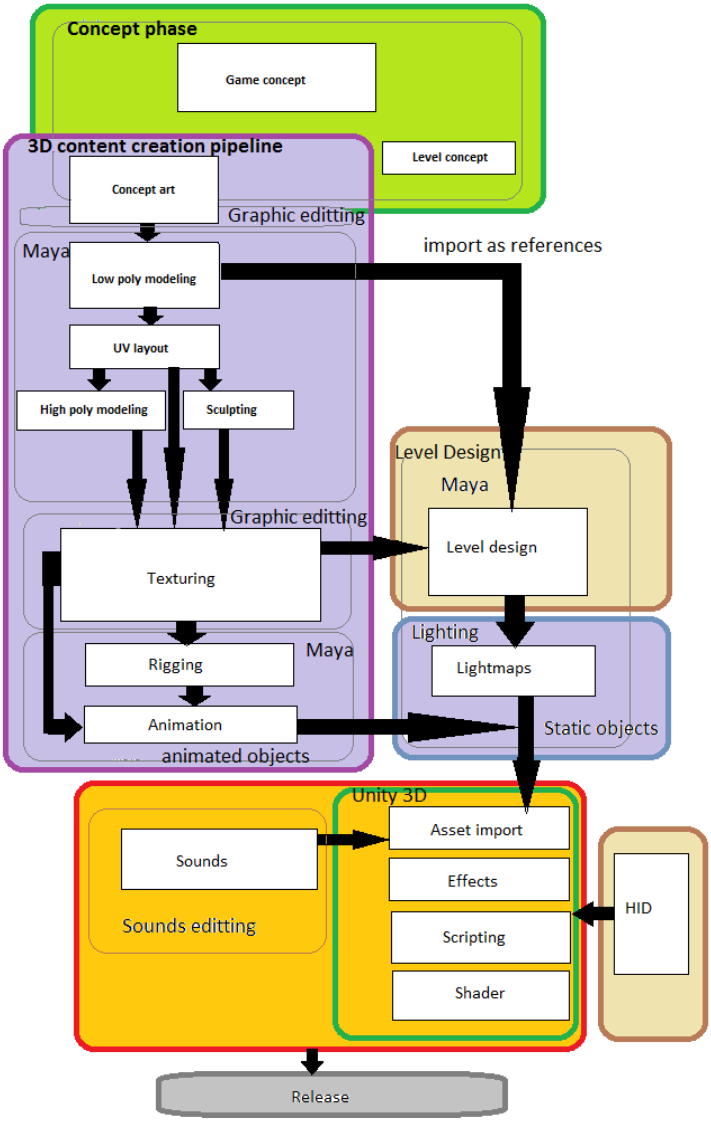


Figure 43: Game Structure

'Taichi' is an educational interactive game. Player is able to learn 'Taichi' by following the moves in the game with the motion sensing controller wearing on their wrist. Unity 3D and Autodesk Maya 2011 were used in the creation of 'Taichi'. The development team included people with varying skills from game designers, artists and game programmers. Each member was given specific roles according to their skills, such as Concept artists, Art Direction, Technical Direction and Project Manager.

In the following, we will introduce the tools that were used, workflow of the game development process and lastly explain some problems faced during the developing of ‘Taichi’, as well as the solutions. Below is one of the ‘Taichi’ in-game screenshot.

Autodesk Maya 2011 was used for modelling, texturing, animating and rendering throughout the project and Unity 3D was used for implementation of the art assets produced from Maya. Besides Maya and Unity 3D, video, audio and image editing software such as Adobe Photoshop and Adobe After Effect were also used during the project.

Autodesk Maya was use for modelling, texturing, animating and rendering in this project. Artists firstly modelled the characters and environments of the game out, unwrapped it, arrange the UV layout of these models, and texture them with Adobe Photoshop. After the models were done, the animators will take over the characters and start the rigging process, after rigging is done, animators will use the controllers created to animate the characters.

Unity3D is an integrated authoring tool for creating 3D video games or other interactive content such as architectural visualizations or real-time 3D animations, focusing on clear interplay of features and functionality. Unity is similar to Director, Blender game engine, Virtools, Torque Game Builder or Gamestudio in the sense that an integrated graphical environment is the primary method of development.

In this Game, the free version of the game engine Unity 3.4 was chosen for the production of ’Dynamite Pete’ as well as the GUI (Graphical User Interface). Unity 3.4 allows you to target all platforms from a single tool. Within a single project you

have complete control over delivery to all platforms. (e.g.: Web Player, PC and Mac Standalone, iOs, Android, Xbox 360, PS3), which eases the development for different consoles or devices.

4.1. General Game Production Pipeline

Our concept is just a simple idea of remaking Tai chi, a 600-year-old dancelike exercise derived from the martial arts — clears the mind, relaxes the body, and contributes to health and longevity into a game [39].

Figure 44: Sketch up for concept design

This production pipeline is basically a concept of workflow management for use in the game development process. The phases of are revised until the release of a video game.

Some of these tasks require to be carried out sequentially as presented in the following. We have to organize ways to carry out the deliverable to allow everyone to work at the same time. Since this very project, it was necessary to follow a pipeline

approach without iterations. The scope of this project was to successfully integrate a working hardware into the game engine.

4.1.1. Concept Phase

During the concept phase a small team designed the character, the setting and the game mechanics. The game's environment was chosen to be a Chinese Temple. A painterly style was preferred over a photorealistic style and the goal of the game was to learn Taichi. After this stage, we began to look at how we can integrate the hardware into the game.

Figure 45: Sketch up for concept design

4.1.2. Concept Art

To cut down time, we took a model for the internet to make the main character. After searching very extensively for it, we made some slight modification to it and removed the original texture, some artists moved on to graphics editing tools,

4.1.3. Low Polygon Modeling

As we wanted the game to run quickly we were not able to put a lot of polygons into our work so strategic planning is required. The character is put in a neutral position. To save the computing power, we only add detail to the subject where it will be needed to help deform the mesh. More attention is paid to where your points are and how they will deform. Points should follow the flow of the body, and no less than five spans per joint.

4.1.4. UV Layout

Before texturing or high polygon modeling could start, UV layouts had to be done either by UV layout artists or by the modeling artists themselves. UV set in Maya consists of a single UV layout. An object with multiple texture types can have multiple UV layouts

Use other projection tools to map out areas that are used in high detail areas. Start picking the larger areas first by selecting the polygonal faces of the body. Open up the tool options for Planar Projection. By selecting the Camera option, Maya will unwrap UVs according to whichever window you have activated. I activated the side view by clicking on the empty portion of the side viewport menu.

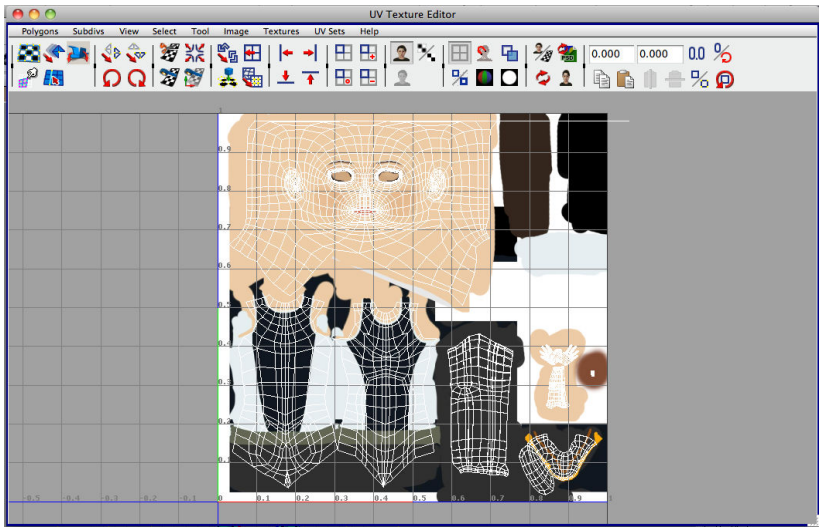


Figure 46: UV menu in Mava

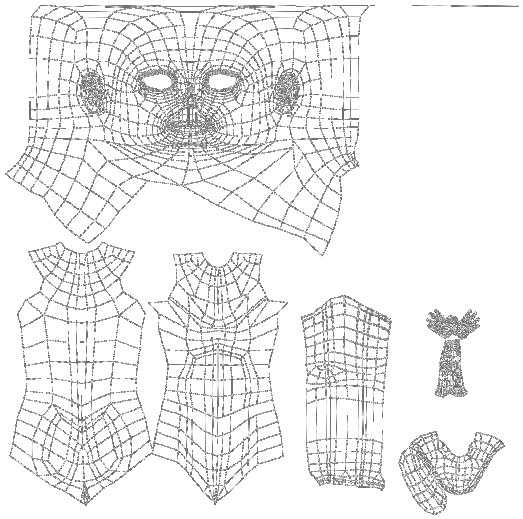


Figure 47: Exported file

. For this project a maximum of three UV sets per object was defined:

- Color map UV set: For color textures. Faces can reuse texture space, meaning that a certain space in texture space can be used by multiple faces.
- Normal map UV set (optional): Only used if a separate normal map was required.

- Light map UV set: Cannot include overlapping faces in texture space. Since each face can have different lighting information, each face has to have its own amount of space in the UV layout.

4.1.5. Texturing

Texturing started right after the UV layout was done. Its is done in Photoshop with a tablet. Each mesh in Maya has to have one (and one only) Maya shader applied to it. But you can apply the same shader to any number of meshes. When you export a file (CGF or CHR), one (and only one) material file (.mtl) is created. This material file holds the list of Maya shaders that were applied to your meshes. So if you export a CGF file made of several meshes but all using the same Maya shader, one .mtl file will be created, containing only one material.

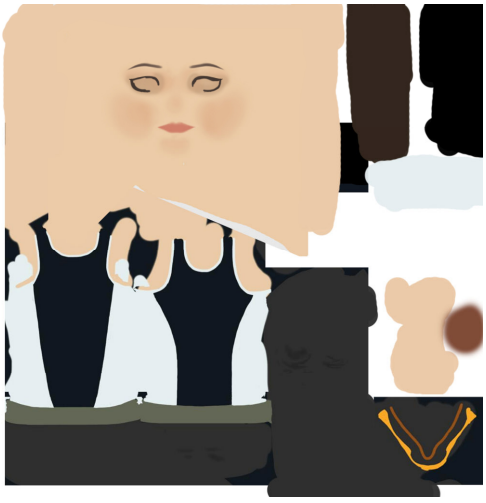


Figure 48: Sketch up for concept design

4.1.6. Rigging and Animation

To enable artists to easily create naturally looking animations of characters, a technique called ’rigging’ is used, which creates a virtual bone structure for each character. This structure can then be used to control the movements of the characters for animation in a natural way For rigging, it is is important to: -create basic skeletal structure hierarchy.

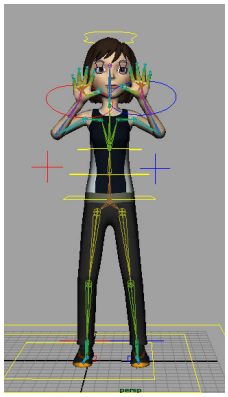
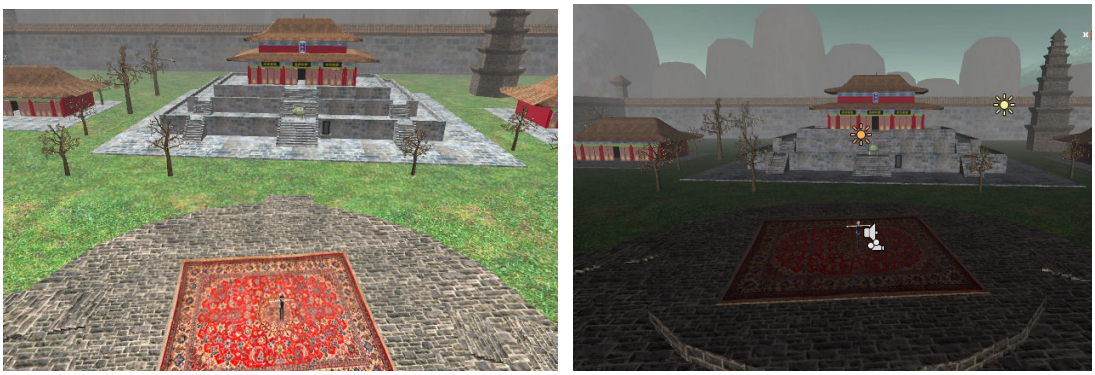


Figure 49: Character

The animation artists then produced 13 sets of key-frame based animations for some movements for the character

4.1.7. Lighting in this project

Our game scene setting is supposed to be during the sunset. We have added 3 directional lighting to the scene. The first was to emulate the sunset ambience, with color set according to the real sunset color. The other 2 light is used to improve the lighting quality and the shadow effect of the character and the environment. The game should be bright enough for the player to be able to view the environment around it.



With Lighting Without Lighting

Figure 50: Lighting

4.1.8. Concept and Presentation Rendering

In addition to the scene's lighting, rendering was also used for many different tasks during the development process. We have to render the environment concept using Maya first to look at the overall feel of the environment. Rendering in Maya does take a lot of time, and the end result in Unity will not be the same in Maya, as rendering using Maya setting in real-time will cause the game to lag. Thus, we have to remove some of the unneeded details and polygon to improve the overall performance of the game. We have improved the rendering in Unity using the image processing available in Unity.

4.2. Implementation

As our artist uses Maya 2011 to model and animate the character, we have to export our Maya project to *.fbx file. As Unity3D uses the *.fbx format as its main 3d model format. Unity3D allows the import of Maya files, however, Maya program has to be installed to the computer and it takes longer time than exporting it directly from Maya.

Converting the fbx file ourselves is better than using Unity to convert the file, as sometimes Unity conversion might give animation error.

4.2.1. Plug-in

Since the control for the game is using the customized HID device that is developed in Session 3, a C#/C++ plug-in is created in order for unity to be able to communicate or access the data of the hid device. The plug-in is exported in *.dll format, it has to be located outside unity project asset folder. When the game is build out for release, the plug-in has to be in the same folder as the *.exe file in order for it to be able to detect the plug-in.

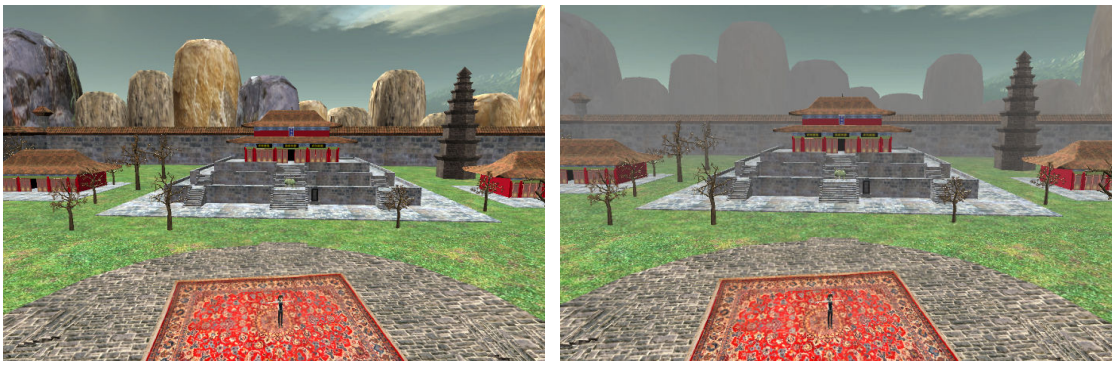
4.2.2. Object Integration in Unity

In order to save time in importing and editing of the 3D models, we have to separate the models into layers, such as building, character and the object. Doing so improve our overall time management, if we were to combine all the models into 1 layer, we have to edit the whole thing whenever there are changes. And object such as trees can be positioned in Unity thus, the artist could spend more time on designing the environment than editing the models.

4.2.3. Effects, Shader Programming

After implementing all the major features of the game, we feel that we have to touch up more on the graphic presentation of the game. We included some shader feature that was in-built within Unity3D engine. Such as particle effect, lens flare, fog and

some image processing shader. For the particle effect, we use the Particle Emitter from Unity3D. We use Unity3D linear fog system to hide the background of the environment. The fog did enhance the feel of the environment feel of mountainous area. We tried on some of the image processing effect in Unity3D but, it does not seem to fit into our current game design, thus we did not include it.



Without Fog With fog

Figure 51: Shader

4.2.4. Sound

The final feature that we have to include in the game is the sound effect and music. With the audio added to the game, the game will be more complete and it improved the overall quality of the game. The sound effect give a audio feedback for the game while the music set the game ambience according to the scene. Unity3D allow the user to set the type of sound and the properties the sound should have. The editor mode help to simplify the way the sound should be handled. Adding a sound to the game will need 2 things, ‘Audio Source’ and ‘Audio Listener’. The audio source is

used set the properties of audio clip and its properties, such as 2D or 3D sound. The audio listener is sound receiver; usually it is attached within the game camera.

4.2.5. Motion Data Library and Gesture Control User Interface

4.2.5.1. Motion Data Library for Game Scoring Mechanism

The whole Taichi game is separated into 13 forms, starting Form 1 to Form13. The motion data is recorded from a Taichi master and saved in a txt file for comparison with the player's input. Each form is saved in a separated motion data sequence as shown in the table bellow.

Time	x	y	z
Timer : 98	1140.164	1094.5	1022.881
Timer : 98.25	1144.59	1088.5	1019.831
Timer : 98.5	1141.639	1097.5	1016.78
Timer : 98.75	1140.164	1099	1021.356
Timer : 99	1143.115	1088.5	1019.831
Timer : 99.25	1143.115	1088.5	1019.831
Timer : 99.5	893.7705	910	874.9153
Timer : 99.75	1149.016	1081	1025.932
Timer : 100	1138.688	1085.5	1022.881
Timer : 100.25	1138.688	1072	1024.407
Timer : 100.5	1146.066	1057	1000
Timer : 100.75	1129.836	1019.5	1140.339
Timer : 101	1112.131	1033	1163.22
Timer : 101.25	1098.852	1021	1158.644
Timer : 101.5	1092.951	1031.5	1144.915
Timer : 101.75	1088.525	1007.5	1120.508
Timer : 102	1090	1013.5	1088.475
Timer : 102.25	1084.098	1031.5	1041.186
Timer : 102.5	1094.426	1058.5	1019.831
Timer : 102.75	1100.328	1072	1010.678
Timer : 103	1113.607	1093	1007.627
Timer : 103.25	1131.312	1076.5	1006.102
Timer : 103.5	1140.164	1069	1015.254
Timer : 103.75	1151.967	1069	1022.881

Timer : 104	1149.016	1073.5	1028.983
Timer : 104.25	1162.295	1070.5	1022.881
Timer : 104.5	1162.295	1070.5	1041.186
Timer : 104.75	1160.82	1052.5	1030.508
Timer : 105	1154.918	1052.5	1025.932
Timer : 105.25	1141.639	1039	1024.407
Timer : 105.5	1125.41	1069	1024.407
Timer : 105.75	1109.18	1072	1021.356
Timer : 106	1101.803	1072	1018.305
Timer : 106.25	1103.279	1076.5	1009.153
Timer : 106.5	1098.852	1087	1009.153
Timer : 106.75	1103.279	1090	1009.153
Timer : 107	1104.754	1091.5	1006.102
Timer : 107.25	1116.557	1090	1010.678
Timer : 107.4393	1116.557	1087	1006.102

Figure 52: Form 3 Motion data recorded from Taichi master

The game mechanism is designed in the way that at each sampled time by the timer, the user's motion data is compared with the standard motion data library. A difference is calculated and an accumulative summation is made. The bigger the difference is, the lower score the player gets. The game scoring mechanism code is attached in Annex E of the thesis.

4.2.6. Game Testing and Result Discussion

The Taichi Game developed in this project was tested in a total sample size of 30 players from 1000 Taichi players ranging from Taichi master to starter. The final score was recorded to investigate on the game scoring mechanism. The motion library is built from Taichi Master denoted as A. And the players are allowed to play the game twice consecutively to track the game score.

Figure 53: Game Testing of Interactive Taichi Game

The score of the 20 players in two plays are recorded as bellow:

Player	Level of Skills in Taichi	Play 1 Score	Play 2 Score
1	Master	9	8
2	Master	7	9
3	Master	6	7
4	Master	8	7
5	Master	4	6
6	Master	7	9
7	Master	8	8
8	Master	7	6
9	Master	8	7
10	Master	8	6
11	Starter	2	3
12	Starter	4	3
13	Starter	2	3
14	Starter	4	5
15	Starter	4	3
16	Starter	5	6
17	Starter	3	4
18	Starter	4	6
19	Starter	3	5
20	Starter	3	4

Figure 54: Game Score of 20 Testers

The average score of the Taichi Master is 7.2 and the average score for the starter is 3.4. It can be concluded that the level of skills is reflected by the score mechanism in the Taichi game developed in above session. However, it can be noted that No.5 Master No.5 only score 4 and 6 in two plays, yet it has been told that the Master plays quite well traditional Taichi. The reason of such unexpected low score is because of No.5 Taichi master is playing different styling of Taichi namely Wu Style [40]

whereas the standard motion library for assessment in this game is recorded from a Taichi master in Yang Style [41] which is more common than Wu Style. However this data has given more collective engineering knowledge in improving the future version of the Taichi game such that it will allow different styles of Taichi.

Chapter 5. Conclusion and Future Research

The research has studied the issues and challenges in making an improved human computer interface. A workable prototype of Wearable Robotics System for Interactive Digital Media is developed. Three major contributions are made including developing a simplified motion capture compared to existing inertial system on market by using accelerometer only, developing a Zigbee RF2.4G wireless sensor network and a USB plug and play device in order for users to use the Wearable Robotics System without installing driver onto their computers.

The prototype was evaluated and experiments are made to prove the mathematical model that this thesis proposed. Simulations are made on a tester to wear the sensor and control the virtual avatar inside a 3D game. Lastly, an interactive Taichi game was developed to prove the value of the system. The Taichi game played by the Wearable Robotics System illustrated to players that they can use natural body motion as input to play the game interactively instead of using keyboard. The game results were also collected to investigate on the game scoring mechanism through the sensor output from the Wearable Robotic System.

The future research work will include integrating low cost gyroscopes and compass to enable more precise motion tracking in complex environment. The work on investigating on the influence of the interactive Taichi game application on rehabilitation will also be proposed and carried out in future.

Bibliography

[1] Engleberg, Isa N. Working in Groups: Communication Principles and Strategies. My Communication Kit Series, 2006

[2] Jane F Thomsen1, Fred Gerr2 and Isam Atroshi3, Carpal tunnel syndrome and the use of computer mouse and keyboard, Department of Occupational Medicine, Copenhagen University Hospital in Glostrup

[3] Frey, W., Off-the-shelf, real-time, "human body motion capture for synthetic environments" Tech. Rep. (1996) NPSCS-96-003, Naval Postgraduate School, Monterey, California

[4] Hightower, J., and Borriello, G., "Location systems for ubiquitous computing", Computer (2001), 34, 8, 57–66

[5] Meyer, K., Applewhite, H. L., and Biocca, F. A., "A survey of position-trackers", Presence (1992), 1, 2, 173–200

[6] Miller, N., Jenkins, O. C., Kallmann, M., and Matric, M. J., "Motion capture from inertial sensing for untethered humanoid teleoperation", International Conference of Humanoid Robotics (2004), 547–565

[7] S.S. Ge, A.P. Loh and F. Guan, "Robust Sound Localization Using Lower Number of Microphones", International Journal of Information Acquisition, Vol. 2, no.1, pp.1-22, March 2005.

[8] S.S. Ge and F.L. Lewis, Editors, Autonomous Mobile Robots: Sensing, Control, Decision-Making, and Applications, CRC Press, Taylor and Francis Group, Boca Raton, FL, 2006.

[9] F. Guan, A.P. Loh and S. S. Ge, "3D Sound Localization Using Movable Microphones Sets," Proceedings of Fourth International Conference on Industrial Automation, Montreal, Canada, June 9-11, 2003.

[10] S. S. Ge, A. P. Loh, and F. Guan, "Sound Localization Based on Mask Diffraction," Proceedings of IEEE International Conference on Robotics and Automation, pp. 1972-1977, September 14-19, 2003, Taipei, Taiwan.

[11] A. P. Loh, S. S. Ge and F. Guan, "Sound Source Tracking Using Movable Microphone Sets," Proceedings of the 2nd International conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2003), Singapore, 15-18 December, 2003.

[12] A. P. Loh, F. Guan, S. S. Ge, "Motion Estimation Using Audio and Video Fusion," Proceedings of the 8th International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 1569-1574, Kunming, China, 6 – 9 December 2004.

[13] S. S. Ge, F. Guan, A.P. Loh and C.H. Fua, "Feature Representation Based on Intrinsic Structure Discovery in High Dimensional Space," Proceedings of the 2006 IEEE International Conference on Robotics and Automation, p.3399-3404, Orlando, Florida, USA, May 15-19, 2006

[14] S. S. Ge, Y. Yang and T.H. Lee, “Hand Gesture Recognition and Tracking based on Distributed Locally Linear Embedding,” Proceedings of IEEE International Conference on Robotics, Automation and Mechatronics, pp. 567-572, Bangkok, Thailand, 7 - 9 June, 2006.

[15] T. S. Chua, S. S. Ge and Y. Yang, “Expression Recognition and Tracking based on Distributed Locally Linear Embedding and Expression Motion Energy”, Workshop on Human Emotions in Voice and Body: Approached from affective sciences and virtual reality, Organized by the Swiss House Singapore, in cooperation with the Swiss Centre for Affective Sciences and MIRALab, University of Geneva, Singapore, 14-15 December 2006.

[16] Ward, J. A., Lukowicz, P., and Troster, G., Gesture spotting using wrist worn microphone and 3-axis accelerometer, Joint Conference on Smart Objects and Ambient Intelligence (2005), 99–104

[17] Bachmann, E. R., Inertial and Magnetic Tracking of Limb Segment Orientation for Inserting Humans Into Synthetic Environments, PhD thesis (2000), Naval Postgraduate School, Monterey, California.

[18] S.S. Ge, T.H. Lee and C.J. Harris, Adaptive Neural Network Control of Robotic Manipulators, World Scientific, London, December 1998.

[19] RF2.4G, http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01

[20] Unity3d Engine: http://unity3d.com/

[21] M. Fitzpatrick, L. Harding, Using the Wii for Vestibular Rehabilitation, Vestibular Disorder Association

[22] Foxlin, E., Pedestrian tracking with shoe-mounted inertial sensors, Computer Graphics and Applications (2005), 25, 6, 38–46

[23] Ward, A., Jones, A., and Hopper, A., A new location echnique for the active office. Personal Communications (1997) 4, 5, 42–47

[24] Hazas, M., and Ward, A., A novel broadband ultrasonic location system, International Conference on Ubiquitous Computing (2002), 264–280

[25] A. Y. Benbasat. An inertial measurement unit for user interfaces. Master's thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, September 2000.

[26] Frey, W., Off-the-shelf, real-time, "human body motion capture for synthetic environments" Tech. Rep. (1996) NPSCS-96-003, Naval Postgraduate School, Monterey, California

[27] Hightower, J., and Borriello, G., "Location systems for ubiquitous computing", Computer (2001), 34, 8, 57–66

[28] Meyer, K., Applewhite, H. L., and Biocca, F. A., "A survey of position-trackers", Presence (1992), 1, 2, 173–200

[29] Bishop, T. G., Self-Tracker: "A Smart Optical Sensor on Silicon", PhD thesis (1984), University of North Carolina at Chapel Hill

[30] The introduction of Altium Designer http://www.altium.com/ products/ altium-designer

[31] Foxlin, E., and Harrington, M., Weartrack: A self-referenced head and hand tracker for wearable computers and portable VR, International Symposium on Wearable Computers (2000), 155–162

[32] Foxlin, E., Pedestrian tracking with shoe-mounted inertial sensors, Computer Graphics and Applications (2005), 25, 6, 38–46

[33] Foxlin, E., Harrington, M., and Pfeifer, G., Constellation: A wide-range wireless motion-tracking system for augmented reality and virtual set applications, Computer Graphics, Annual Conference Series (1998), 371–378

[34] Ward, J. A., Lukowicz, P., and Troster, G., Gesture spotting using wrist worn microphone and 3-axis accelerometer, Joint Conference on Smart Objects and Ambient Intelligence (2005), 99–104

[35] Bachmann, E. R., Inertial and Magnetic Tracking of Limb Segment Orientation for Inserting Humans Into Synthetic Environments, PhD thesis (2000), Naval Postgraduate School, Monterey, California.

[36] MEMS, http://en.wikipedia.org/wiki/Microelectromechanical_systems

[37] HID, http://en.wikipedia.org/wiki/Human_interface_device

[38] Maya, http://usa.autodesk.com/maya/

[39] Taichi Chuan, http://en.wikipedia.org/wiki/T'ai_chi_ch'uan

[40] Wu Style Taichi, http://en.wikipedia.org/wiki/Wu_style_tai_chi_chuan

[41] Yang Style Taichi, http://www.fushengyuan-taichi.com.au/

Annex A: Microcontroller Firmware

```
/*****************************include files*******************************/
#include <p18cxxx.h>
#include <p18f45j10.h>
/*****************************include files*******************************/
#define HARDWARE_TRANSMITTER
//#define HARDWARE_RECEIVER
#define ID 1
/*****************************include files*******************************/
#include "GenericTypeDefs.h"
#include "MRF24J40MA.h"
#include "KEYBOARD.h"
#include "GyroDataProcessing.h"
#include "ChipConfig.h"
/*****************************include files*******************************/


#pragma config XINST    = OFF      // Extended instruction set
#pragma config STVREN   = ON       // Stack overflow reset
//#pragma config PLLDIV   = 5      // (20 MHz crystal used on this board)
#pragma config WDTEN    = OFF      // Watch Dog Timer (WDT)
#pragma config CP0      = OFF      // Code protect
//#pragma config CPUDIV   = OSC1   // OSC1 = divide by 1 mode
#pragma config IESO     = OFF      // Internal External (clock) Switchover
#pragma config FCMEN    = OFF      // Fail Safe Clock Monitor
#pragma config FOSC     = HSPLL    // Firmware must also set OSCTUNE<PLLEN> to start PLL!
#pragma config WDTPS    = 32768
//#pragma config WAIT     = OFF         // Commented choices are
//#pragma config BW       = 16         // only available on the
//#pragma config MODE     = MM         // 80 pin devices in the
//#pragma config EASHFT   = OFF         // family.
//#pragma config MSSPMSK  = MSK5
//#pragma config PMPMX    = DEFAULT
//#pragma config ECCPMX   = DEFAULT
#pragma config CCP2MX   = DEFAULT


/*****************************Function declaration*************************/
void Load_ID(void);
void Load_Gyro_Data(void);
void Load_ACC_Data(void);
void Load_KB_Data(void);
```

```
/******************************Function declaration*********************************/
void YourHighPriorityISRCode();
void YourLowPriorityISRCode();
/** Important VECTOR REMAPPING ***********************************************/
#if defined(__18CXX)
//On PIC18 devices, addresses 0x00, 0x08, and 0x18 are used for
//the reset, high priority interrupt, and low priority interrupt
//vectors.  However, the current Microchip USB bootloader
//examples are intended to occupy addresses 0x00-0x7FF or
//0x00-0xFFF depending on which bootloader is used.  Therefore,
//the bootloader code remaps these vectors to new locations
//as indicated below.  This remapping is only necessary if you
//wish to program the hex file generated from this project with
//the USB bootloader.  If no bootloader is used, edit the
//usb_config.h file and comment out the following defines:
//#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER
//#define PROGRAMMABLE_WITH_USB_LEGACY_CUSTOM_CLASS_BOOTLOADER

#if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)
#define REMAPPED_RESET_VECTOR_ADDRESS         0x1000
#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS    0x1008
#define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS    0x1018
#elif defined(PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)
#define REMAPPED_RESET_VECTOR_ADDRESS         0x800
#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS    0x808
#define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS    0x818
#else
#define REMAPPED_RESET_VECTOR_ADDRESS         0x00
#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS    0x08
#define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS    0x18
#endif

#if
defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOA
DER)
extern void _startup (void);        // See c018i.c in your C18 compiler dir
#pragma code REMAPPED_RESET_VECTOR = REMAPPED_RESET_VECTOR_ADDRESS
void _reset (void)
{
_asm goto _startup _endasm
}
#endif
#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR = REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
void Remapped_High_ISR (void)
{
```

```
_asm goto YourHighPriorityISRCode _endasm
}
#pragma code REMAPPED_LOW_INTERRUPT_VECTOR = REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
void Remapped_Low_ISR (void)
{
_asm goto YourLowPriorityISRCode _endasm
}

#if
defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOA
DER)

#pragma code HIGH_INTERRUPT_VECTOR = 0x08
void High_ISR (void)
{
_asm goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS _endasm
}
#pragma code LOW_INTERRUPT_VECTOR = 0x18
void Low_ISR (void)
{
_asm goto REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS _endasm
}
#endif   //end of "#if
defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(PROGRAMMABLE_WITH_USB_LEGACY_CUSTOM_C
LASS_BOOTLOADER)"

#pragma code


//These are your actual interrupt handling routines.
#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode() //adc code
{
     if(INTCONbits.INT0IF)
    {
          INTCONbits.INT0IF=0;
          INTCONbits.INT0IE=0;
    }
     //***TMR0***
     if(INTCONbits.TMR0IF) //MRF call
     {
          INTCONbits.TMR0IE = 0;//disable timer0
          INTCONbits.TMR0IF = 0; //clear flag
/*********************************************************************/
          Load_ID();//0
```

```
            Load_Gyro_Data();//123
            Load_ACC_Data();//456
            Load_KB_Data();//78910
            PORTDbits.RD1=1;
            MRF24J40_Transmit(11,RFTxBuffer);
            PORTDbits.RD1=0;
/*********************************************************************/
            TMR0H=0xfd;
            TMR0L=0x00;
            INTCONbits.TMR0IE=1;
      }
}

#pragma interruptlow YourLowPriorityISRCode
void YourLowPriorityISRCode()
{
}

#elif defined(__C30__)
#if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)

#endif
#endif
//******END interrupt Service Routine******************


//development board code
void main(void)
{
     unsigned char i = 0;

    InitialPorts();

     PORTDbits.RD1=0;

     for(i=0;i<TX_PLOAD_WIDTH;i++)
          RFTxBuffer[i]=0;

     while(1);
}
void Load_ID(void)
{
     RFTxBuffer[0]=ID;
}
void Load_Gyro_Data(void)
```

```
{
    UINT8 GYRO1,GYRO2;
    UINT8 MOUSE1,MOUSE2;
    RFTxBuffer[3]=0;
}
void Load_ACC_Data(void)
{
    RFTxBuffer[4]=AD_CONVERSION(0x00);//AN0
    RFTxBuffer[6]=AD_CONVERSION(0x03);//AN3
}
void Load_KB_Data(void)
{
    UINT8 joy1,joy2;
    if(RFTxBuffer[5]>190)
        RFTxBuffer[8]=KEY_J;
    else if(joy1>170)
        RFTxBuffer[8]=KEY_RIGHT;
    else if(joy1<80)
        RFTxBuffer[8]=KEY_LEFT;
    else if(joy2>170)
        RFTxBuffer[8]=KEY_UP;
    else if(joy2<80)
        RFTxBuffer[8]=KEY_DOWN;
    else if(PORTDbits.RD4)
        RFTxBuffer[8]=KEY_ENTER;
}
```

Annex B: HID firmware in USB

```
#ifndef __USB_DESCRIPTORS_C
#define __USB_DESCRIPTORS_C

/** INCLUDES *******************************************************/
#include "GenericTypeDefs.h"
#include "Compiler.h"
#include "usb_config.h"

#include "usb_device.h"          //haojie
#include "usb_function_hid.h"    //haojie key
#include "usb_function_hid_m.h"  //haojie mouse
#include "usb_function_hid_h.h"  //haojie hid

/** CONSTANTS ******************************************************/
#if defined(__18CXX)
```

```
#pragma romdata
#endif

/* Device Descriptor */
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
    0x12,                           // Size of this descriptor in bytes
    USB_DESCRIPTOR_DEVICE,          // DEVICE descriptor type
    0x0200,                         // USB Spec Release Number in BCD format
    0x00,                           // Class Code
    0x00,                           // Subclass code
    0x00,                           // Protocol code
    USB_EP0_BUFF_SIZE,          // Max packet size for EP0, see usb_config.h
    MY_VID,                         // Vendor ID
    MY_PID,                         // Product ID: Mouse in a circle fw demo
    0x0001,                         // Device release number in BCD format
    0x01,                           // Manufacturer string index
    0x02,                           // Product string index
    0x00,                           // Device serial number string index
    0x01                            // Number of possible configurations
};

/* Configuration 1 Descriptor */
ROM BYTE configDescriptor1[]={
    /* Configuration Descriptor */
    0x09,//sizeof(USB_CFG_DSC),      // Size of this descriptor in bytes
    USB_DESCRIPTOR_CONFIGURATION,   // CONFIGURATION descriptor type
    DESC_CONFIG_WORD(0x005B),    //haojie Total length of data for this cfg
    3,                              //haojie // Number of interfaces in this cfg
    1,                              // Index value of this configuration
    0,                              // Configuration string index
    _DEFAULT | _SELF,               // Attributes, see usb_device.h
    50,                             // Max power consumption (2X mA)

    /* Interface Descriptor */
    0x09,//sizeof(USB_INTF_DSC),     // Size of this descriptor in bytes
    USB_DESCRIPTOR_INTERFACE,       // INTERFACE descriptor type
    0,                              //hoajie Interface Number
    0,                              // Alternate Setting Number
    1,                              //haojie Number of endpoints in this intf
    HID_INTF,                       // Class code
    BOOT_INTF_SUBCLASS,             // Subclass code
    HID_PROTOCOL_KEYBOARD,          // Protocol code
    0,                              // Interface string index
```

```
    /* HID Class-Specific Descriptor */
    0x09,//sizeof(USB_HID_DSC)+3,    // Size of this descriptor in bytes RRoj hack
    DSC_HID,                          // HID descriptor type
    DESC_CONFIG_WORD(0x0111),        // HID Spec Release Number in BCD format (1.11)
    0x00,                             // Country Code (0x00 for Not supported)
    HID_NUM_OF_DSC,                   //haojie Number of class descriptors, see usbcfg.h
    DSC_RPT,                          // Report descriptor type
    DESC_CONFIG_WORD(HID_RPT01_SIZE),//hoajie sizeof(hid_rpt01),      // Size of the report
descriptor

    /* Endpoint Descriptor */
    0x07,/*sizeof(USB_EP_DSC)*/
    USB_DESCRIPTOR_ENDPOINT,     //Endpoint Descriptor
    HID_EP | _EP_IN,             //EndpointAddress
    _INTERRUPT,                     //Attributes
    DESC_CONFIG_WORD(8),         //size
    0x01,                             //haojie  //Interval

     /*----------mouse----------*/
    /* Interface Descriptor */
    0x09,//sizeof(USB_INTF_DSC),      // Size of this descriptor in bytes
    USB_DESCRIPTOR_INTERFACE,        // INTERFACE descriptor type
    1,                                //haojie // Interface Number
    0,                                // Alternate Setting Number
    1,                                // Number of endpoints in this intf
    HID_INTF,                         // Class code
    BOOT_INTF_SUBCLASS,               // Subclass code
    HID_PROTOCOL_MOUSE,               // Protocol code
    0,                                // Interface string index

    /* HID Class-Specific Descriptor */
    0x09,//sizeof(USB_HID_DSC)+3,    // Size of this descriptor in bytes RRoj hack
    DSC_HID,                          // HID descriptor type
    DESC_CONFIG_WORD(0x0111),        // HID Spec Release Number in BCD format (1.11)
    0x00,                             // Country Code (0x00 for Not supported)
    HID_NUM_OF_DSC_M,                 //haojie  // Number of class descriptors, see usbcfg.h
    DSC_RPT,                          // Report descriptor type
    DESC_CONFIG_WORD(HID_RPT01_SIZE_M),   //haojie

    /* Endpoint Descriptor */
    0x07,/*sizeof(USB_EP_DSC)*/
    USB_DESCRIPTOR_ENDPOINT,     //Endpoint Descriptor
    HID_EP_M | _EP_IN,           //haojie  //EndpointAddress
    _INTERRUPT,                     //Attributes
    DESC_CONFIG_WORD(3),            //size
```

```
    0x01,                           //haojie Interval

    /*----------hid----------*/
    /* Interface Descriptor */
    0x09,//sizeof(USB_INTF_DSC),     // Size of this descriptor in bytes
    USB_DESCRIPTOR_INTERFACE,       // INTERFACE descriptor type
    2,                              //haojie Interface Number
    0,                              // Alternate Setting Number
    2,                              //haojie Number of endpoints in this intf
    HID_INTF,                       // Class code
    0,                                  // Subclass code
    0,                                  // Protocol code
    0,                              // Interface string index

    /* HID Class-Specific Descriptor */
    0x09,//sizeof(USB_HID_DSC)+3,    // Size of this descriptor in bytes
    DSC_HID,                        // HID descriptor type
    0x11,0x01,                      // HID Spec Release Number in BCD format (1.11)
    0x00,                           // Country Code (0x00 for Not supported)
    HID_NUM_OF_DSC_H,               //haojie Number of class descriptors, see usbcfg.h
    DSC_RPT,                        // Report descriptor type
    DESC_CONFIG_WORD(HID_RPT01_SIZE_H),   // haojie sizeof(hid_rpt01),      // Size of the
report descriptor

    /* Endpoint Descriptor */
    0x07,/*sizeof(USB_EP_DSC)*/
    USB_DESCRIPTOR_ENDPOINT,    //Endpoint Descriptor
    HID_EP_H | _EP_IN,              //haojie //EndpointAddress
    _INTERRUPT,                     //Attributes
    0x40,0x00,                  //size
    0x01,                           //Interval

    /* Endpoint Descriptor */
    0x07,/*sizeof(USB_EP_DSC)*/
    USB_DESCRIPTOR_ENDPOINT,    //Endpoint Descriptor
    HID_EP_H | _EP_OUT,             //haojie EndpointAddress
    _INTERRUPT,                     //Attributes
    0x40,0x00,                  //size
    0x01                        //Interval
};


//Language code string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[1];}sd000={
sizeof(sd000),USB_DESCRIPTOR_STRING,{0x0409
```

```
}};

//Manufacturer string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[25];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{'M','i','c','r','o','c','h','i','p',' ',
'T','e','c','h','n','o','l','o','g','y',' ','I','n','c','.'
}};

//Product string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[22];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{'K','e','y','b','o','a','r','d',' ','D','e','m','o'
}};

//Class specific descriptor - HID keyboard
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
    {    0x05, 0x01,                    // USAGE_PAGE (Generic Desktop)
    0x09, 0x06,                    // USAGE (Keyboard)
    0xa1, 0x01,                    // COLLECTION (Application)
    0x05, 0x07,                    //   USAGE_PAGE (Keyboard)
    0x19, 0xe0,                    //   USAGE_MINIMUM (Keyboard LeftControl)
    0x29, 0xe7,                    //   USAGE_MAXIMUM (Keyboard Right GUI)
    0x15, 0x00,                    //   LOGICAL_MINIMUM (0)
    0x25, 0x01,                    //   LOGICAL_MAXIMUM (1)
    0x75, 0x01,                    //   REPORT_SIZE (1)
    0x95, 0x08,                    //   REPORT_COUNT (8)
    0x81, 0x02,                    //   INPUT (Data,Var,Abs)
    0x95, 0x01,                    //   REPORT_COUNT (1)
    0x75, 0x08,                    //   REPORT_SIZE (8)
    0x81, 0x03,                    //   INPUT (Cnst,Var,Abs)
    0x95, 0x05,                    //   REPORT_COUNT (5)
    0x75, 0x01,                    //   REPORT_SIZE (1)
    0x05, 0x08,                    //   USAGE_PAGE (LEDs)
    0x19, 0x01,                    //   USAGE_MINIMUM (Num Lock)
    0x29, 0x05,                    //   USAGE_MAXIMUM (Kana)
    0x91, 0x02,                    //   OUTPUT (Data,Var,Abs)
    0x95, 0x01,                    //   REPORT_COUNT (1)
    0x75, 0x03,                    //   REPORT_SIZE (3)
    0x91, 0x03,                    //   OUTPUT (Cnst,Var,Abs)
    0x95, 0x06,                    //   REPORT_COUNT (6)
    0x75, 0x08,                    //   REPORT_SIZE (8)
    0x15, 0x00,                    //   LOGICAL_MINIMUM (0)
    0x25, 0x65,                    //   LOGICAL_MAXIMUM (101)
    0x05, 0x07,                    //   USAGE_PAGE (Keyboard)
```

```
    0x19, 0x00,                    //   USAGE_MINIMUM (Reserved (no event indicated))
    0x29, 0x65,                    //   USAGE_MAXIMUM (Keyboard Application)
    0x81, 0x00,                    //   INPUT (Data,Ary,Abs)
    0xc0      }
};/* End Collection,End Collection            */

//Class specific descriptor - HID mouse
ROM struct{BYTE report[HID_RPT01_SIZE_M];}hid_rpt01_m={ //haojie
    {0x05, 0x01, /* Usage Page (Generic Desktop)             */
    0x09, 0x02, /* Usage (Mouse)                            */
    0xA1, 0x01, /* Collection (Application)                  */
    0x09, 0x01, /*  Usage (Pointer)                         */
    0xA1, 0x00, /*  Collection (Physical)                   */
    0x05, 0x09, /*      Usage Page (Buttons)                */
    0x19, 0x01, /*      Usage Minimum (01)                  */
    0x29, 0x03, /*      Usage Maximum (03)                  */
    0x15, 0x00, /*      Logical Minimum (0)                 */
    0x25, 0x01, /*      Logical Maximum (0)                 */
    0x95, 0x03, /*      Report Count (3)                    */
    0x75, 0x01, /*      Report Size (1)                     */
    0x81, 0x02, /*      Input (Data, Variable, Absolute)    */
    0x95, 0x01, /*      Report Count (1)                    */
    0x75, 0x05, /*      Report Size (5)                     */
    0x81, 0x01, /*      Input (Constant)    ;5 bit padding  */
    0x05, 0x01, /*      Usage Page (Generic Desktop)        */
    0x09, 0x30, /*      Usage (X)                           */
    0x09, 0x31, /*      Usage (Y)                           */
    0x15, 0x81, /*      Logical Minimum (-127)              */
    0x25, 0x7F, /*      Logical Maximum (127)               */
    0x75, 0x08, /*      Report Size (8)                     */
    0x95, 0x02, /*      Report Count (2)                    */
    0x81, 0x06, /*      Input (Data, Variable, Relative)    */
    0xC0, 0xC0}
};/* End Collection,End Collection            */

//Class specific descriptor - HID
ROM struct{BYTE report[HID_RPT01_SIZE_H];}hid_rpt01_h={
{
    0x06, 0x00, 0xFF,        // Usage Page = 0xFFFF (Vendor Defined)
    0x09, 0x01,              // Usage
    0xA1, 0x01,              // Collection (Application, probably not important because vendor
defined usage)
    0x19, 0x01,              //      Usage Minimum (Vendor Usage = 0) (minimum bytes the device
should send is 0)
```

```
    0x29, 0x04,             //haojie      Usage Maximum (Vendor Usage = 64) (maximum bytes the
device should send is 64)
    0x15, 0x00,             //      Logical Minimum (Vendor Usage = 0)
    0x26, 0xFF, 0x00,       //      Logical Maximum (Vendor Usage = 255)
    0x75, 0x08,             //      Report Size 8 bits (one full byte) for each report.
    0x95, 0x04,             //haojie      Report Count 64 bytes in a full report.
    0x81, 0x02,             //      Input (Data, Var, Abs)
    0x19, 0x01,             //      Usage Minimum (Vendor Usage = 0)
    0x29, 0x04,             //haojie      Usage Maximum (Vendor Usage = 64)
    0x91, 0x02,             //      Output (Data, Var, Ads)
    0xC0}
};                  // End Collection

//Array of configuration descriptors
ROM BYTE *ROM USB_CD_Ptr[]=
{
    (ROM BYTE *ROM)&configDescriptor1
};

//Array of string descriptors
ROM BYTE *ROM USB_SD_Ptr[]=
{
    (ROM BYTE *ROM)&sd000,
    (ROM BYTE *ROM)&sd001,
    (ROM BYTE *ROM)&sd002
};

/** EOF usb_descriptors.c ***************************************************/

#endif
```

Annex C: Win32 Class driver to read WRS data

```
BOOL CUsbhidiocDlg::DeviceNameMatch(LPARAM lParam)
    {

    // Compare the device path name of a device recently attached or removed
    // with the device path name of the device we want to communicate with.

    PDEV_BROADCAST_HDR lpdb = (PDEV_BROADCAST_HDR)lParam;

    DisplayData("MyDevicePathName = " + MyDevicePathName);
```

```
    if (lpdb->dbch_devicetype == DBT_DEVTYP_DEVICEINTERFACE)
    {

        PDEV_BROADCAST_DEVICEINTERFACE lpdbi = (PDEV_BROADCAST_DEVICEINTERFACE)lParam;


        CString DeviceNameString;

        //The dbch_devicetype parameter indicates that the event applies to a device interface.
        //So the structure in LParam is actually a DEV_BROADCAST_INTERFACE structure,
        //which begins with a DEV_BROADCAST_HDR.

        //The dbcc_name parameter of DevBroadcastDeviceInterface contains the device name.

        //Compare the name of the newly attached device with the name of the device
        //the application is accessing (myDevicePathName).

        DeviceNameString = lpdbi->dbcc_name;

        DisplayData("DeviceNameString = " + DeviceNameString);


        if ((DeviceNameString.CompareNoCase(MyDevicePathName)) == 0)

        {
            //The name matches.

          return true;
        }
        else
        {
          //It's a different device.

          return false;
    }

    }
        else
        {
            return false;
        }
}


void CUsbhidiocDlg::DisplayCurrentTime()
```

```
{
    //Get the current time and date and display them in the log List Box.

    CTime curTime = CTime::GetCurrentTime();
    CString CurrentTime = curTime.Format( "%H:%M:%S, %B %d, %Y" );
    DisplayData(CurrentTime);
}


void CUsbhidiocDlg::DisplayData(CString cstrDataToDisplay)
{
    //Display data in the log List Box

    USHORT    Index;
    Index=m_ResultsList.InsertString(-1, (LPCTSTR)cstrDataToDisplay);
    ScrollToBottomOfListBox(Index);
}


void CUsbhidiocDlg::DisplayFeatureReport()
{
    USHORT    ByteNumber;
    CHAR ReceivedByte;

    //Display the received data in the log and the Bytes Received List boxes.
    //Start at the top of the List Box.

    m_BytesReceived.ResetContent();

    //Step through the received bytes and display each.

    for (ByteNumber=0; ByteNumber < Capabilities.FeatureReportByteLength; ByteNumber++)
    {
        //Get a byte.

        ReceivedByte = FeatureReport[ByteNumber];

        //Display it.

        DisplayReceivedData(ReceivedByte);
    }
}


unsigned char test22;
```

```
int inPosX = 600;
int inPosY = 300;
int XinPosX = 1000 - 50;
int YinPosX = 1100 - 50;
int ZinPosX = 1200 - 50;

void CUsbhidiocDlg::DisplayInputReport()
{
     USHORT   ByteNumber;
     unsigned char ReceivedByte;
     char ProcessData;
     //char ReceivedByte;
     CString  strs;


     test22++;
     //Display the received data in the log and the Bytes Received List boxes.
     //Start at the top of the List Box.

     m_BytesReceived.ResetContent();

     //Step through the received bytes and display each.

     for (ByteNumber=0; ByteNumber < Capabilities.InputReportByteLength; ByteNumber++)
     {
          //Get a byte.
          ReceivedByte = InputReport[ByteNumber];
          //   strs.Format("%02X", ReceivedByte);
          //   strs = strs.Right(2);



          if (ByteNumber==2) //testing, was 2
          {
               //write
               fprintf(fd1, "%d\t", ReceivedByte);
               m_output_X.MoveWindow(XinPosX,ReceivedByte+300,20,20,1);

          }


          if (ByteNumber==3) //testing, was 2
          {
               /*if(ReceivedByte >160)
                    inPosY-=20;
```

```
        if(inPosY <= 200)
            inPosY = 600;
        m_LeftFoot.MoveWindow(inPosX,inPosY,100,100,1);
        */
        //raw output to show
        m_output_Y.MoveWindow(YinPosX,ReceivedByte+300,20,20,1);



        //process data and integrate
        if(ReceivedByte >116&&ReceivedByte<136)
            ReceivedByte = 126;
        ProcessData = ReceivedByte - 126;

        if(ProcessData>=-10&&ProcessData<=10)
            ProcessData=0;

        inPosY-=ProcessData/10;
        if(inPosY <= 100||inPosY>=700)
            inPosY = 600;

        m_RightFoot.MoveWindow(inPosX+100,inPosY,100,100,1);

        fprintf(fd1, "%d\t", ReceivedByte);
    }

    if (ByteNumber==4)
    {
        //raw data to show
        m_output_Z.MoveWindow(200-ReceivedByte+1100,600,20,20,1);

        ////write
        //if(ReceivedByte >96&&ReceivedByte<116)
        //   ReceivedByte = 106;

        fprintf(fd1, "%d\t", ReceivedByte);
        //fprintf(fd1, "\n");
    }

    if (ByteNumber==5)
    {
        //raw data to show
        //m_output_Z.MoveWindow(200-ReceivedByte+1100,600,20,20,1);

        ////write
```

```
                //if(ReceivedByte >96&&ReceivedByte<116)
                //   ReceivedByte = 106;

                fprintf(fd1, "%d\t", ReceivedByte);
                fprintf(fd1, "\n");
            }

            //Display it.
            DisplayReceivedData(ReceivedByte);
        }

}
```

Annex D: Avatar Control

```
using UnityEngine;
using System.Collections;
using System;
using System.Collections.Generic;

using System.Text;
using System.Runtime.InteropServices;
using System.IO;
using System.Threading;

[System.Serializable]
public class DeviceCap
{
    public int maxX, minX,
               maxY, minY,
               maxZ, minZ;
};
public class HID
{
    protected static DeviceCap[] deviceCap;
     public static int[,] hidData;
    public static float[] timer;    //list of the device id last update
    //public static List<int> timeOutDevice = new List<int>();    //list of timeout device id
    public static bool[] deviceTimeOut;

    public static void SetDeviceCap(DeviceCap[] dev)
    {
        deviceCap = dev;
```

```
}

public static int GetReport(int deviceID, int dataID)
 {
    lock (hidData)
    {
        return hidData[deviceID, dataID];
    }
 }

public static float GetOrientation(int deviceId, int dataId)
{
    float  min = 0, max = 1, input;
    int index = dataId;

    switch (dataId)
    {
        case 0:
            {
                min = deviceCap[deviceId].minX;
                max = deviceCap[deviceId].maxX;
            } break;
        case 1:
            {
                min = deviceCap[deviceId].minY;
                max = deviceCap[deviceId].maxY;
            } break;
        case 2:
            {
                min = deviceCap[deviceId].minZ;
                max = deviceCap[deviceId].maxZ;
            } break;
    }

    input = (float)HID.GetReport(deviceId, index);

    //input = Mathf.Clamp(input, min, max);

    return (input - min) / (max - min) * 180.0f;
}

public static float[] GetOrientations(int deviceId)
{
    return new float[3] { GetOrientation(deviceId, 0),
                          GetOrientation(deviceId, 1),
```

```
                                GetOrientation(deviceId, 2) };
    }
    public static void DataSize(int id, int data)
    {
        hidData = new int[id, data];
        timer = new float[id];
        deviceTimeOut = new bool[id];
        for (int i = 0; i < id; ++i)
            deviceTimeOut[i] = true;
    }
};

public class PortegeHidPlugin : MonoBehaviour {

     [DllImport("HID_Plugin.dll")]
     protected static extern void UpdateHID();
     //
     [DllImport("HID_Plugin.dll")]
     protected static extern int inputReport(int n);

    [DllImport("HID_Plugin.dll")]
    protected static extern void ResetInput();

    [DllImport("HID_Plugin.dll")]
    protected static extern bool deviceFound();

    //[DllImport("HID_Plugin.dll")]
    //public static extern bool deviceFound();

     private StreamWriter output;
     //private float timer;

     private bool startRecording = false;
     private string form;

     //private int timer;
     private float recordStart = 0;

     public string performer;

     public int[] deviceID;
     public int[] dataArray;
    public Vector2[] drawPos;
    public Vector2 translateWarning;
    public DeviceCap[] deviceCap;
```

```
private float sX, sY;

private Thread thread = null;

private bool runThread = false;

public Texture2D[] leftDevTex;
public Texture2D[] rightDevTex;
public Texture2D[] captTex;

bool deviceWorking;
float countdown;

 // Use this for initialization
 void Start () {

      //timer = Time.timeSinceLevelLoad;
      Time.fixedDeltaTime = 0.25F;
     HID.DataSize(deviceID.Length, dataArray.Length);
     HID.SetDeviceCap(deviceCap);

     ResetInput();

     StartHID();
 }

void OnDisable()
{
     //usb.CleanUp();
     StopHID();
}

void OnApplicationQuit()
{
     StopHID();
}

 void OnGUI()
 {
      GUI.depth = -2;

     if (Event.current.type == EventType.Repaint)
     {
         int caption, right, left;
```

```
            left = HID.deviceTimeOut[0] ? 0 : 1;
            right = HID.deviceTimeOut[1] ? 0 : 1;
            caption = right == 0 || left == 0 ? 0 : 1;

            if (countdown > 0.0f)
            {
                Color defaultColor = GUI.color;

                GUI.color = new Color(defaultColor.r, defaultColor.g, defaultColor.b,
countdown);

                GUI.DrawTexture(new Rect(20 * sX, 20 * sY, leftDevTex[left].width * sX,
leftDevTex[left].height * sY), leftDevTex[left]);
                GUI.DrawTexture(new Rect((20 + leftDevTex[left].width) * sX, 20 * sY,
rightDevTex[left].width * sX, rightDevTex[left].height * sY), rightDevTex[right]);
                GUI.DrawTexture(new Rect((20 + leftDevTex[left].width -
(captTex[caption].width * 0.5f)) * sX, (20 + (leftDevTex[left].height * 0.5f) -
(captTex[caption].height * 0.5f)) * sY,
                                                captTex[caption].width * sX,
captTex[caption].height * sY), captTex[caption]);

                GUI.color = defaultColor;
            }
            if (caption == 1)   //device working
            {
                if (!deviceWorking)
                    deviceWorking = true;
            }
            else
            {
                deviceWorking = false;
                countdown = 1.0f;
            }
        }
    }

    void StartHID()
    {
        if (thread != null) return;

        runThread = true;
        try
        {
            thread = new Thread(new ThreadStart(RunHID));
```

```
            thread.Start();
        }
        catch (Exception e)
        {
            print(e.Message);
        }
        finally
        {
            //print("Thread Started");
        }
    }

    void StopHID()
    {
        if (thread == null) return;
        //print("Abborted");

        runThread = false;

        thread.Abort();
        thread = null;
    }

    void RunHID()
    {
        DateTime now = DateTime.Now;
        int fps = 0, tick = 0, id, dataPerSec = 0;
        int[] data;

        bool emptyData;
        while (runThread)
        {
            TimeSpan elapsedSpan = new TimeSpan(DateTime.Now.Ticks - now.Ticks);
            now = DateTime.Now;

            UpdateHID();

            //if (!deviceFound()) return;

            emptyData = true;

            id = inputReport(2);
            data = new int[dataArray.Length];

            for (int i = 0; i < dataArray.Length; ++i)
```

```
                data[i] = inputReport(dataArray[i]);



            if (id == 0)
            {
                foreach (int no in data)
                    if (no != 0)
                    {
                        emptyData = false;
                        break;
                    }
            }
            else emptyData = false;

            for (int i = 0; i < deviceID.Length; ++i)
            {
                HID.timer[i] += elapsedSpan.Milliseconds;
                if (HID.deviceTimeOut[i]) continue;    //the device id is already in the time
out list
                //the device id has not been updated for the pass 1 second
                //this might be due to the device error or no battery
                if (HID.timer[i] > 500)
                {
                    //print("HID REMOVE " + i);
                    for (int n = 0; n < dataArray.Length; ++n)
                        HID.hidData[i, n] = 0;
                    //HID.timeOutDevice.Add(i);
                    HID.deviceTimeOut[i] = true;
                }
            }

            if (!emptyData)
                for (int i = 0; i < deviceID.Length; ++i)
                    if (deviceID[i] == id)
                    {
                        //print(deviceID[i] +" " + id);
                        HID.timer[i] = 0;
                        //print("HID REMOVE " + i);
                        HID.deviceTimeOut[i] = false;
                        //HID.timeOutDevice.Remove(i);    //we'll remove the device id from the
time out list

                        //print(HID.GetOrientation(i, 0) + " " + HID.GetOrientation(i, 1) + "
" + HID.GetOrientation(i, 2));
```

```
                for (int n = 0; n < dataArray.Length; ++n)
                    HID.hidData[i, n] = inputReport(dataArray[n]);
                break;
            }

        if (!emptyData)
        {
            ++dataPerSec;
           /* print(inputReport(0) + "\t" +
                  inputReport(1) + "\t" +
                  inputReport(2) + "\t" +
                  inputReport(3) + "\t" +
                  inputReport(4) + "\t" +
                  inputReport(5) + "\t" +
                  inputReport(6) + "\t" +
                  inputReport(7));*/
        }

        ResetInput();

        tick += elapsedSpan.Milliseconds;
        ++fps;

        if (tick > 1000)
        {
            tick = 0;
           // print("Thread Fps : " + fps);
           // print("Data Per Sec : " + dataPerSec);
            fps = 0;
            dataPerSec = 0;
        }
    }
}
 // Update is called once per frame
void Update()
{
    sX = (float)Screen.width / 1024.0F;
    sY = (float)Screen.height / 768.0F;

    if(deviceWorking)
    {
        if (countdown > 0.0f)
            countdown -= Time.deltaTime;
        else
```

```
                countdown = 0.0f;
            }
        }

        void StartRecording(string move)
        {
            recordStart =  Time.timeSinceLevelLoad;
            form = move;
            startRecording = true;
            SaveInputTo("Takes/" + performer + form, Time.timeSinceLevelLoad - recordStart);
        }


        void EndOfClip()
        {

        }

        void SaveInputTo(string fileName, float currTime)
        {
            string file;

            for( int i = 0; i < deviceID.Length; ++i )
            {
                file = fileName + "_" + i + ".txt";

                if( !File.Exists(file) )
                     output = File.CreateText(file);
                else
                    output = File.AppendText(file);

                output.Write( "Timer : " + currTime );

                for( int k = 0; k < dataArray.Length; ++k )
                    output.Write( "\t" + HID.GetReport( i, k ) );

                output.WriteLine( "" );
                output.Close();

                print( "Recording Timer : " + currTime );
            }
        }
    }
```

Annex E: Game Scoring System for Taichi

```
void StartScoreAnimation()
{
    startScoreBoardAnim = true;

    if (rollAnim != 0) return;

    paintR = null;
    paintAnim = 0;

    taichiPoseR = null;
    poseAnim = 0;

    signAnim = -1;
    scrollRollR = scrollRoll[rollAnim];
    alpha = 0;
    watchFrame = -1;

    showProgress = finishProgress = showScore = showGrade = false;

    p1PercentBar = 0.95f;
    sfx.PlayOneShot(clips[0]);
    gradeAlpha = 0;
    Invoke("RollAnim", scrollSpeed);
}

void RollAnim()
{
    if (alpha < 1.0f)
        alpha += 0.075f;
    ++rollAnim;

    if (rollAnim == scrollRoll.Length)
    {
        rollAnim = 0;

        showProgress = true;

        float p = (float)p1Score / (float)totalScore;

        if (p > 0.85f)
            grade = 4;
        else if (p > 0.65f)
```

```
            grade = 3;
        else if (p > 0.5f)
            grade = 2;
        else if (p > 0.3f)
            grade = 1;
        else
            grade = 0;

        sfx.PlayOneShot(clips[4]);

        p1Percent = Mathf.InverseLerp(totalScore, 0, p1Score) * 0.8f + 0.1f;

        float time = 3 - ((p1Percent - 0.1f) / 0.8f); //0-3 seconds

        p1PercentAdd = (p1PercentBar - p1Percent) / (framePerSecond * - time);
        if (p1PercentAdd > 0 || p1Score == 0)
        {
            p1PercentAdd = 0;
            p1Percent = p1PercentBar;
        }

        return;
    }
    else if (rollAnim < scrollRoll.Length)
    {
        Invoke("RollAnim", scrollSpeed);
    }

    scrollRollR = scrollRoll[rollAnim];
}

void PoseAnim()
{
    ++poseAnim;

    if (poseAnim == taichiPose.Length)
    {
        poseAnim = 0;    //this is for debug purposes
        Invoke("SignAnim", signSpeed);
        return;
    }
    else if (poseAnim < taichiPose.Length)
        Invoke("PoseAnim", poseSpeed);

    taichiPoseR = taichiPose[poseAnim];
```

```
}

void SignAnim()
{
    ++signAnim;
    print(signAnim);
    if (signAnim == 2)
    {
        signAnim = 1;
        showGrade = true;
    }
    else if (signAnim < redDot.Length)
        Invoke("SignAnim", signSpeed);
}

void PaintAnim()
{
    ++paintAnim;
    if( paintAnim == 1 )
        sfx.PlayOneShot(clips[3]);
    if (paintAnim == paint.Length)
    {
        paintAnim = 0;    //this is for debug purposes
        showScore = true;
        taichiPoseR = taichiPose[poseAnim];
        Invoke("PoseAnim", painSpeed);
        return;
    }
    else if (paintAnim < paint.Length)
        Invoke("PaintAnim", painSpeed);

    paintR = paint[paintAnim];
}

void DrawWatch(bool fadeOut)
{
    if (Event.current.type == EventType.Repaint)
    {
        if (fadeOut)
        {
            if (!gamePause)
            {
                if (alpha < 1)
                    alpha += Time.deltaTime;
                else
```

```
                    {
                        watchFrame += Time.deltaTime * 30;
                        if (watchFrame > 24) watchFrame = 24;
                    }
                }
                GUI.color = new Color(1, 1, 1, alpha);
                GUI.DrawTexture(new Rect(0, 0, 1024 * screenRatioX, 768 * screenRatioY),
watchFade);
                GUI.color = new Color(1, 1, 1, 1);

                if (watchFrame > -1)
                    GUI.DrawTexture(new Rect(336 * screenRatioX, 5 * screenRatioY,
watch[(int)watchFrame].width * screenRatioX, watch[(int)watchFrame].height * screenRatioY),
watch[(int)watchFrame]);
            }
            else
            {
                if (!gamePause)
                {
                    if (alpha > 0)
                        alpha -= Time.deltaTime * 2;
                }
                GUI.color = new Color(1, 1, 1, alpha);
                GUI.DrawTexture(new Rect(0, 0, 1024 * screenRatioX, 768 * screenRatioY),
watchFade);

                if (watchFrame > -1)
                    GUI.DrawTexture(new Rect(336 * screenRatioX, 5 * screenRatioY,
watch[(int)watchFrame].width * screenRatioX, watch[(int)watchFrame].height * screenRatioY),
watch[(int)watchFrame]);
                GUI.color = new Color(1, 1, 1, 1);
            }
        }
    }

    void DrawScoreBoard()
    {
        //score
        Rect rect = new Rect(756 * screenRatioX, 606 * screenRatioY, scoreBoard.width *
screenRatioX, scoreBoard.height * screenRatioY);
        GUI.DrawTexture(rect, scoreBoard);

        GUI.Label(new Rect(830 * screenRatioX, 639 * screenRatioY, 110, 50), p1Score.ToString(),
scoreStyle);
```

```
        //GUI.Label(new Rect(100 * screenRatioX, 132 * screenRatioY, 110, 50),
p2Score.ToString(), scoreStyle);
    }
    void DrawProgressBar()
    {
        if (Event.current.type == EventType.Repaint)
        {
            if (totalScoreBar > totalScorePercent)
            {
                totalScoreBar -= totalScoreAdd;
            }
            if (totalScoreBar < totalScorePercent)
                totalScoreBar = totalScorePercent;
        }

        Rect rect;

        const float totalScoreBarX = 26, totalScoreBarY = 525;
        const float progBarX = 189, progBarY = 656;
        const float percentBarY = 663, barMax = 475;

        rect = new Rect(progBarX * screenRatioX, progBarY * screenRatioY,
progressBarTex[0].width * screenRatioX, progressBarTex[0].height * screenRatioY);
        GUI.DrawTexture(rect, progressBarTex[0]);   //prog bar base

        Color c1, c2;   //start color and end color
        float percent;  //the color lerp percentage

        if (p1PercentBar < 0.25f)   //0 to 25%
        {
            c1 = tryAgainColor;
            c2 = fairColor;
            percent = p1PercentBar * 4;
        }
        else if (p1PercentBar < 0.5f) //25 - 50%
        {
            c1 = fairColor;
            c2 = goodColor;
            percent = (p1PercentBar - 0.25f) * 4;
        }
        else if (p1PercentBar < 0.75f)  //50 - 75%
        {
            c1 = goodColor;
            c2 = excellentColor;
            percent = (p1PercentBar - 0.5f) * 4;
```

```
        }
        else    //75 - 100%
        {
            c1 = c2 = excellentColor;
            percent = 1;
        }

        GUI.color = Color.Lerp(c1, c2, percent);    //lerp the color

        float pBarW = barMax * p1PercentBar;

        rect = new Rect(progBarX * screenRatioX, percentBarY * screenRatioY, pBarW * screenRatioX,
progressBarTex[4].height * screenRatioY);
        GUI.DrawTexture(rect, progressBarTex[4]);   //this is the progress bar which will move

        float fadeBarX = progBarX + pBarW - (progressBarTex[5].width * 0.1f);
        float fadeBarMaxX = barMax - (progressBarTex[5].width * 0.75f);
        float fadeBarWidth = progressBarTex[5].width;

        if (pBarW > fadeBarMaxX)
        {
            float diff = barMax - pBarW + (progressBarTex[5].width * 0.25f);
            float diff2 = barMax - fadeBarMaxX + (progressBarTex[5].width * 0.25f);
            float p = diff / diff2;
            fadeBarWidth = (progressBarTex[5].width) * p;
        }

        rect = new Rect(fadeBarX * screenRatioX, percentBarY * screenRatioY, fadeBarWidth *
screenRatioX, progressBarTex[5].height * screenRatioY);
        GUI.DrawTexture(rect, progressBarTex[5]);   //this is the progress bar fade

        GUI.color = Color.white; //change back to default color
        rect = new Rect(totalScoreBarX * screenRatioX, totalScoreBarY * screenRatioY,
progressBarTex[1].width * screenRatioX, progressBarTex[1].height * screenRatioY);
        GUI.DrawTexture(rect, progressBarTex[1]);   //the grey color circle

        mat.SetFloat("_Cutoff", totalScoreBar);

        Graphics.DrawTexture(rect, mat.mainTexture, mat); //color is 147 200 172    this is the
bar itself

        GUI.DrawTexture(rect, progressBarTex[2]);   //move bar front image this is the tai chi
logo
    }
```