DESIGN AND DEVELOPMENT OF DATA MANAGEMENT SYSTEM FOR MOTION DETECTION SYSTEM

LIM YU HUNG

A project report submitted in partial fulfilment of the
requirements for the award of the degree of
Bachelor of Engineering (Hons.) Mechatronics Engineering

Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2011

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : Lim Yu Hung

ID No. : 08UEB07291

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"DESIGN AND DEVELOPMENT OF DATA MANAGEMENT SYSTEM FOR MOTION DETECTION SYSTEM"** was prepared by **LIM YU HUNG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Mechatronics Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : Mr. Chuah Yea Dat

Date : _____

DESIGN AND DEVELOPMENT OF DATA MANAGEMENT SYSTEM FOR MOTION DETECTION SYSTEM

ABSTRACT

Fall can be perceived as an abnormal motion which will cause people especially elderly to suffer from pain and more seriously can affect one's health. Age population could not sustain any risk from falling that a normal youngster does especially at home when everyone is working away from home. There are some existing fall detection products on the market to assist elderly so that immediate response can be taken to save a precious life. In this report, a user friendly graphic user interface is developed to obtain real time data and monitor the motion of user. Graphs showing the digital code for acceleration in X, Y and Z-axis will be plotted and the data obtained will be saved in a text file for offline analysis purpose. A tri-axis accelerometer is used to detect the motion of body and the analogue data from this sensor is converted to digital value by using analogue and digital converter in microcontroller. The microcontroller also functions to communicate with personal computer through wireless module and perform the fall detection algorithm. Motion is being monitored in real time to detect if there is a fall happens. Simple fall algorithm using threshold value is developed based on experimental data collected from back falling, right-side falling and left-side falling. Experiments have also been carried out for normal daily activities like walking, sitting, squatting and standing back again and jumping. Buzzer on the hardware will sound and a message will prompt out on the graphic user interface to alert people nearby so that necessary action can be taken.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

a_T	total acceleration of body in digital code
a_x	X-axis acceleration in digital code
a_y	Y-axis acceleration in digital code
a_z	Z-axis acceleration in digital code

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

Studies have shown that many countries are suffering from the increase of elderly population and this trend is expected to keep on increasing in the next years (Perolle, Fraisse, Mavros, and Etxeberria, 2006). Many of elderly people are left alone at home when their grown up children go out work to earn some living. Accidents might occur during this time and these home accidents are identified to be falls, fire and flames, poisoning, choking and suffocation and others (Department of Health, Social Services and Public Safety UK, 2004). Among them, falls are predominant. The effect of falls happened on elderly cannot be neglected since it might cause them to paralyse at old age or even take away their lives if no immediate treatments are given to them (Tarik Al-ani, Quynh Trang Le Ba and Eric Monacelli, 2007).

As for the reason mentioned, it is good to have a system that can help detect and even monitor the motion of elderly at home with the convenience brought by technologies nowadays. This project relates Motion Detection System (MDS) used in home basis. Motion detection is one particular field in Body Area Network (BAN) for monitoring the daily activity of a person. In MDS, vital sign sensors, which are tri-axial accelerometers, are placed on a person's body with high correlation to motion. The tasks of MDS are to detect several motions like walking, sitting down, getting up, and especially the falling types of motions. The vital signals collected from the two accelerometers are sent to personal computer through wireless modules and the acceleration-to-time graph will be plotted out automatically in real time. This information is ready to be analysed by medical expert to identify a fall. Emergency

signal will be given out to alert the person's family and immediate action can be taken by medical centre to save a precious life at home.

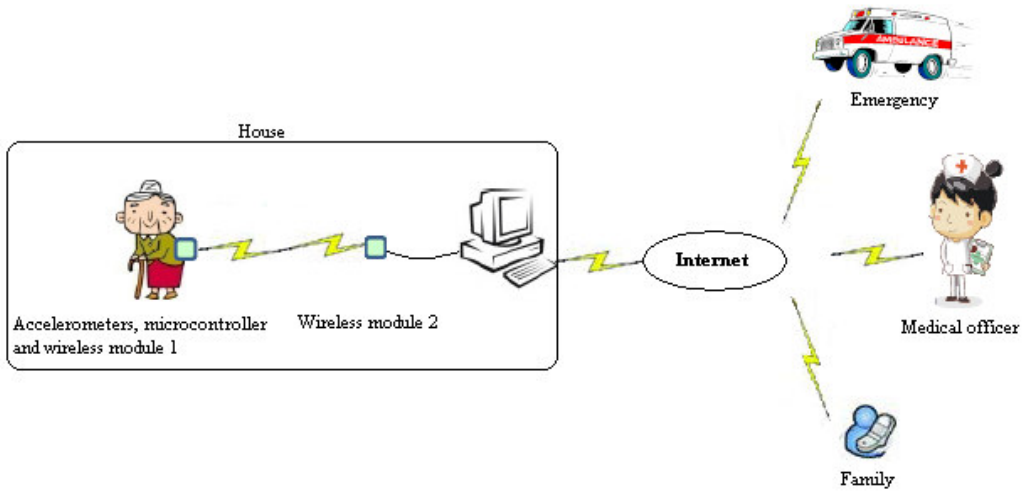


Figure 1.1: Motion Detection System (MDS)

The main reason to carry out this project is to reduce the number of accidental deaths and injuries of elderly caused by fall in home at Malaysia. Other reasons that lead to the conduction of this project are to reduce the Malaysian medical expenditure, to solve the problem of lack of rooms in hospitals, and to optimize the quality of life and independence of elderly.

In this project, only the house stage is of our concern. There are few challenges in designing and developing a MDS. The most critical part is the reliability of the system. It is important for the system to retrieve accurate and precise data from the accelerometer with minimum noise introduced and small time delay as well (ideally no time delay). Data should not be lost also during the data transferred from sensors to computer. The plotted graph should clear enough for proper interpretation. The second challenge is to design ergonomic hardware which does not intrude the daily life of elderly. It would be ideal to make the hardware not detectable by others. Besides, data transfer from sensors to computer must not provide movement restriction to them in their house, wireless modules are thus used in our system. The other problems would be the respect of privacy and personal data and the ease of use of the system to users.

1.1 Background

In Europe countries, there are many different home telemedicine systems developed ranging from very simple medicine-taking reminder, glucose and blood pressure monitoring system, to complex multi-users interface with Artificial Intelligent in decision making (Blount et al., 2007). MDS is perceived as a sub-brunch of Home Telehealth. It is classified as active telemonitoring in Tarik Al-ani et al. (2007) paper since sensors are attached to a person and alert will be generated when there is a fall. There are four main groups of technologies used to detect fall: worn device with immediate detection, worn device to detect unusual behaviour, environment sensing with immediate sensing and lastly environment sensing to detect unusual behaviour (Perolle et al., 2006).

In Dai Jiangpeng, Bai Xiaole, Yang Zhimin, Shen Zhaohui and Xuan Dong (2009), fall detection techniques are classified into three categories by academic researchers: acceleration based detection, databases based motion detection and lastly image processing based detection. Acceleration based detection is mostly based on thresholds and this would be the focus in our project. Databases based motion detection is a more powerful system which store detected user behaviour for different activities. Fall can be detected by classifying these activities and the motion of users are being monitored at the same time. Finally, image processing based detection is utilising video camera to detect fall. This system sacrifices users' privacy and the detection area is somehow restricted as there are "blind spots" in the house.

1.2 Aim and Objectives

The aim of this project is to design and develop a data management system for real time MDS at home in Malaysia which can be modified to suit the users' and developers' requirements as compare to existing commercial products. The data

management is divided into to two levels, low and high levels. Low level is responsible for data acquisition from accelerometers' input signals to computer and high level is converting these raw data into useful information and display on Graphic User Interface (GUI) with clear and proper indications. Simple fall detection algorithm will also be developed in this level.

There are several objectives to be achieved along with the aim throughout this project. Among them are:

- To develop an effective and accurate data acquisition system.
- To develop a simple and user-friendly Graphic User Interface.
- To design a real time data acquisition system.
- To design a durable MDS for appropriate working hours.

This report is organised as follows: Chapter 2 Literature Review, review on six related papers with comments; Chapter 3 Methodology, steps taken to complete this project; Chapter 4 Results and Discussions, design of GUI, source code explanations, experimental results, fall detection algorithm and problem encountered; Chapter 5 Conclusion and Recommendations, conclude the overall design and propose improvement future development.

CHAPTER 2

LITERATURE REVIEW

2.1 Dynamic Fall Detection and Pace Measurement in Walking Sticks

Almeida Oscar, Zhang Ming and Liu Jyh-Charn (2007) proposed dynamic fall detection system and pace measurement into walking sticks and canes using a gyroscope and Atmel EB63 evaluation board. This system detects fall by using a simple threshold method with defined stick's stability. Alarm will be given out when the stick's stability equal or larger than the falling threshold and the reset button is not pressed by the user in specific time. The user's walking pace is also monitored by the detection system, such that the user will be warned when travelling at paces above his or her normal speed. This system is able to preserve energy with different polling frequency levels.

The overall system is good but the major concern is on the walking stick and processing board. As the data collected from the gyroscope is transmitted through wire as shown in the figure, it might restrict the movement of elderly. It might even cause elderly to fall down if the wire is not being placed properly. Besides that, the size of the processing board is considerably large for elderly to carry even its weight might be low. This design is somehow intrusive to elderly as other people can notice the device easily. The propose solution will be making the data transfer between the stick and processing board wirelessly since no one would carry the stick for whole day and waste their effort taking off the system from their body if they just want to leave the stick for a moment. The idea is similar to the Bluetooth earphone used nowadays.

On the idea of fall detection algorithm, it is worth for referencing as it is based on simple threshold value to detect a fall. However, different falling type might result in different threshold values. For instance, the threshold value might different if the elderly is holding the walking stick while falling down compare to if he or she let the walking stick to fall freely. False positive (fall is detected but it did not occur in actual case) might happen if the walking stick is lifted and rotated from vertical to horizontal orientation by the users. Perhaps a pressure sensor is required to install at the base of the stick in this situation to ensure no alarm will be given out to prevent false detection.

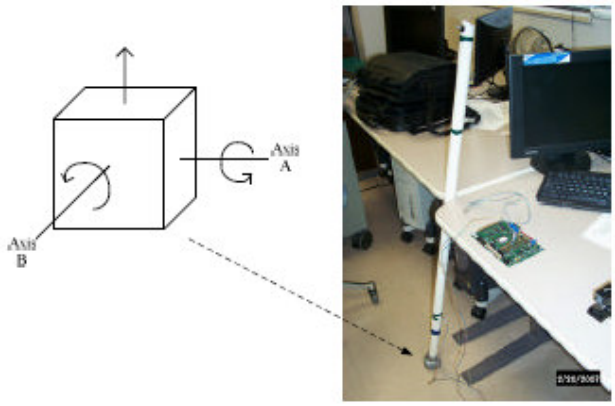


Figure 2.1: Walking Stick Prototype with Gyro and Atmel EB63 Evaluation Board. Orientation of the Gyro, Located at the Base of the Stick (Almeida et al., 2007)

2.2 PerFallD: A Pervasive Fall Detection System Using Mobile Phones

Dai et al. (2009) proposed a fall detection system using mobile phone with integrated accelerometer. Since this paper is using mobile phone as the detection devices, the major focus is on fall detection algorithm, alerting function and user interfaces design. The fall algorithm is based on the amplitude of total acceleration of phone body $|A_T| = \sqrt{|A_x|^2 + |A_y|^2 + |A_z|^2}$ and the amplitude of acceleration at the absolute

vertical direction $|A_v| = |A_x \sin\theta_z + A_y \sin\theta_y + A_z \cos\theta_y \cos\theta_z|$. If both parameters exceed the thresholds set within a period of time, alarm will be triggered. Users can change the setting of the system through the designed interface also.

Using mobile phone platform to detect fall is a new and good system as one can get a mobile phone with multi-functions at affordable price. The major issue in this system is that will using only one accelerometer integrated in the mobile phone enough for detecting a real fall. Noises might be generated and if the system solely depends on one accelerometer might not be that reliable. Moreover, as stated in the paper that there are three possible positions that a user can place the mobile phone, so the threshold set in fall algorithm might not be the optimum one. The area of sensing should be fixed in order to have more accurate fall detection rate.

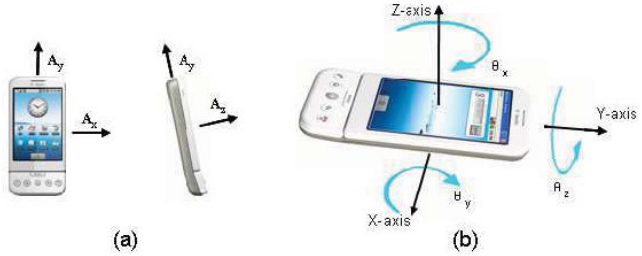


Figure 2.2: (a) Acceleration Readings in Directions of X-, Y-, and Z-Axis that are Associated with and Fixed Regard to the Body of the Mobile Phone. (b) Mobile Phone Orientation can be Decided by Yaw (Θ_x), Pitch (Θ_y) and Roll (Θ_z) (Dai Et Al., 2009)

2.3 Fall Detection Sensor System

In centre suisse d'electronique et de microtechnique [csem] (2004) paper, the detection system is integrated in the wrist watch. This device consists of a microprocessor (MSP430) with two Micro-Electromechanical System (MEMS) sensors which function as ADXL321 accelerometer. The user interface is LCD screen and an integrated vibrator for alert the user that a fall has been detected and alarm will soon be sent. Data can be transmitted over short distance using Bluetooth

protocol but this function is for future development. The acceleration signals are recorded and store in flash memory. No real time monitoring is mentioned in this paper currently. The system is tested offline using MATLAB. The algorithm used in this paper is similar to that in Dai et al. (2009) without the use of the amplitude of absolute vertical direction. It uses the time dependent acceleration vector with selected threshold to discriminate between two classes: fall and no fall.

The main advantage of placing fall detector at wrist is the possibility of wearing the device at night. The major drawback is the signal processing challenge of estimating a fall from wrist acceleration data, due to the strong accelerations experienced by the forearm during daily-life activities. This system is only suitable to detect intentional fall but not unintentional fall which contributes the most in the motion of a person.

Figure 2.3: Wrist Fall Detection System (centre suisse d'electronique et de microtechnique, 2004)

2.4 Accurate, Fast Fall Detection Using Gyroscopes and Accelerometer-Derived Posture Information

In this paper (Li et al., 2009), a fall detection system using both accelerometers and gyroscopes is proposed. Human activities are divided into two categories: static postures and dynamic transitions. By using two tri-axial accelerometers at separate body locations, the system can recognize four kinds of static postures: standing, bending, sitting, and lying. Motions between these static postures are considered as dynamic transitions. Linear acceleration and angular velocity are measured to

determine whether motion transitions are intentional. If the transition before a lying posture is not intentional, a fall event is detected. In addition, this solution features low computational cost and real-time response.

Although this system gives high fall detection rate, the results obtained did not reflect the reliability since data are only collected from three male people. There are several thresholds set in the algorithm based on the data collected from these people and these thresholds might not applicable to other users especially elderly in detecting real fall. Quality and quantity of the data sets are important and cannot be neglected. It is advised to widen the experiment area in order to obtain more accurate data and thresholds. Besides, this system also has difficulties in differentiating jumping into bed and falling against wall with a seated posture.

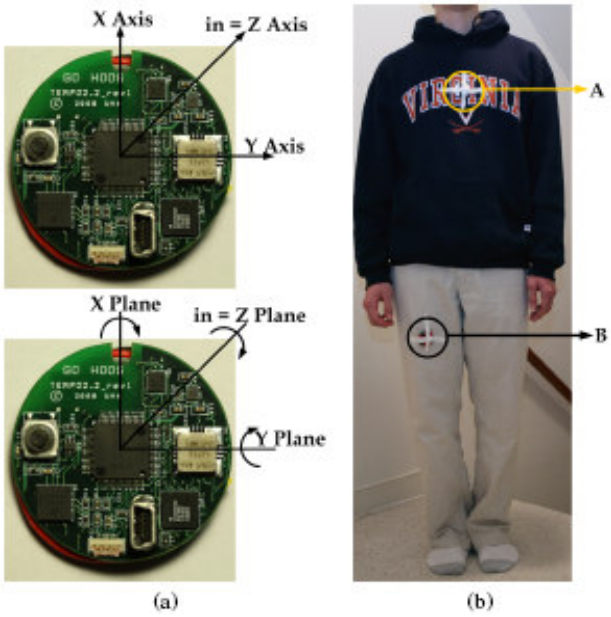


Figure 2.4: (a) The TEMPO 3.0 Sensor Node; (b) The Placement of Two TEMPO 3.0 Nodes (Li et al., 2009)

2.5 On-line Automatic Detection of Human Activity in Home Using Wavelet and Hidden Markov Models Scilab Toolkits

In Tarik Al-ani et al. (2007) paper, one ADXL202E bi-axial accelerometer is connected to a PIC16F876 microcontroller to gather the raw data and sends the acceleration-to-time information to a personal computer. This information is further processed by the computer using MATLAB and SCILAB software. The whole system is trained and tested using Wavelet and Hidden Markov Model, an Artificial Intelligent (AI) approach to distinguish four different human activity events: walk, fall during walking, fall starting from static position and sitting down-getting up.

This system is using algorithm that is more complicated than other previously discussed. It requires more computational power and memory storage for storing training data sets. This system is ideal for motion monitoring but as for simple fall detection, using this system is not that suitable and appropriate. It could be taken as a good reference for future development.

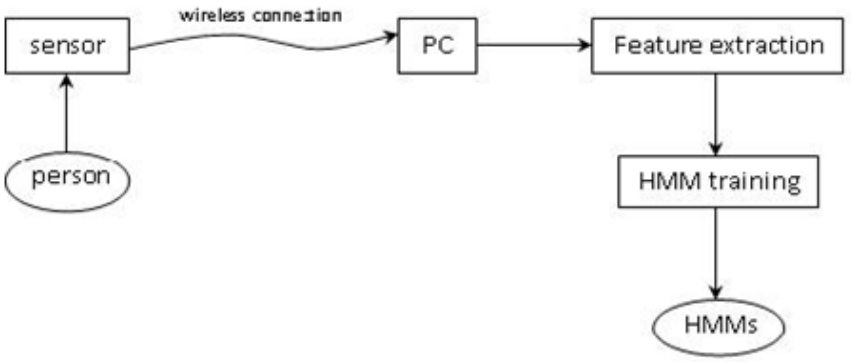


Figure 2.5: Functional Diagram of Automatic Detection System (Tarik Al-ani et al., 2007)

2.6 Fall Detection System for Bather Using Ultrasound Sensors

This fall detection system is using two ultrasound sensors (KEYENCE CORPORATION, UD-330) located at the roof of the bathroom, (Dobashi Hiroki, Tajima Takuya, Abe Takehiko, Kimura Haruhiko, 2008). This paper describes detection of bather's fall in bathroom at Japan which is related to their custom to take hot bath and soak in bathtub. The principle of fall detection in this system is by

calculating the behaviour identification rate using the height of bather head. The height of bather when he or she is sitting is taken as the threshold for determining whether the bather is standing or fell down. When the identification rate is high, this indicates that an accident has been occurred.

The issue to be concerned is whether the sensors are sensing the bather or other objects in the bathroom. When using this system, the area of concern must be well defined to make sure that the sensors are sensing the bather, not other things. Hence, everything inside the bathroom must be identified so that the unwanted data will be filtered before the further analysis is carried out.

This system is very practical to detect fall of bather since the bather cannot wear any device when he or she is taking bath. However, it might not suitable to be implemented throughout the entire area of house. It would be a very expensive system to implement because many sensors will be used due to the sensing range for these ultrasound sensors is only from 400mm to 3000mm as stated in the paper. The idea of using ultrasound to detect fall is worth to take as a reference.

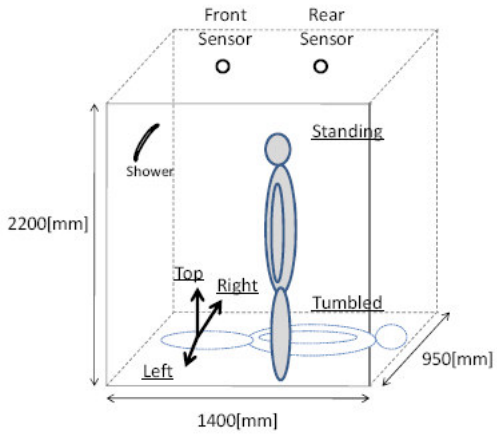


Figure 2.6: System Composition (Dobashi et al., 2008)

2.7 Review of Other Works and Existing Commercial Products

In Miaou Shaou-Gang, Sung Pei-Hsu, and Huang Chia-Yuan (2006) paper, they propose fall detection system using omni-camera. Image of a person is captured and this image will go through image processing with personal information of each individual taken into consideration to detect fall. Although the camera used is capable to capture 360˚ scene, several cameras will be needed to install in the entire house since there are different partitions in a house (rooms, toilet, kitchen and etc.). This is a costly system to be implemented and privacy will need to be intruded. Besides that, Fu, Z., Culurciello, E., Lichtsteiner, P., Delbruck, T. (2008); Sixsmith, A. and Johnson, N. (2004); also propose capturing images of people to detect fall based on image processing techniques. Such approaches have the same limitations mentioned above with moderate fall detection rate.

There are some commercial health monitoring products such as Philips' Lifeline uses a help button to issue medial alerts when the user falls. However, when a really serious fall happens, people may be in unconscious state and not able to push the button. Experiments have been carried out in Dai et al. (2009) paper on the commercial product provided by Brickhouse. The results show that this system has high false negative (29.9%) in backward falls. Meanwhile, the false positive is also high (21.9%). This product not only incapable to detect real fall, but it is bringing inconvenience to elderly as they need to reset the product every time a false alarm is triggered.

2.8 Overall Review

Table 2.1: Overall Review on Six Papers

Paper	Method	Outcome
Dynamic Fall Detection and Pace Measurement in Walking Sticks	• Walking sticks and canes with a gyroscope and Atmel EB63 evaluation board to detect fall and measure pace. • Fall is identified when the velocity	Able to identify a fall even the stick does not totally lie on the floor horizontally.

	exit the defined threshold.	
PerFallD: A Pervasive Fall Detection System Using Mobile Phones	• Mobile phone with integrated accelerometer to detect fall and send emergency signal • Fall algorithm is based on the amplitude of total acceleration of phone body and the acceleration at the absolute vertical direction. Fall is identified if both parameters exceed the thresholds defined.	Able to detect forward, lateral, backward fall with different accuracy.
Fall Detection Sensor System	• Wrist watch to detect fall. • Detect fall offline based on the acceleration of wrist watch.	Only able to detect intentional fall.
Accurate, Fast Fall Detection Using Gyroscopes and Accelerometer-Derived Posture Information	• Detect fall by placing accelerometers and gyroscopes at chest and thigh of a person. • Threshold-based fall detection.	Able to detect different fall type except falling against wall and jumping into bed.
On-line Automatic Detection of Human Activity in Home Using Wavelet and Hidden Markov Models Scilab Toolkits	• Bi-axial accelerometer and microcontroller to collect data and detect fall using Wavelet and Hidden Markov Models Scilab Toolkits. • Fall is detected base on classification.	Able to distinguish walk, fall during walk, fall start from static position and sitting down-getting up.
Fall Detection System for Bather Using Ultrasound Sensors	• Ultrasound installed in bathroom to detect fall. • Threshold-based fall detection using the sitting position as the threshold.	Detect fall in a place installed with sensor.

CHAPTER 3

METHODOLOGY

3.1 Project Work Scope

MDS is only focusing on real time motion detection, fall, in a house. Wireless communication modules, SKXbee with Xbee, which range up to 30m is used for data transfer between microcontroller PIC16F877A and personal computer. This range is far enough for a common size of house of 40 feet x 80 feet. Raw data from accelerometers is transmitted to PIC16F877A via Port A and Port E and the coding is done using MPLAB in C language. The data received by PIC will be sent through one set of SKXbee 1 wirelessly to another SKXbee 2. Finally, data received on SKXbee 2 is transmitted to computer via Universal Serial Bus (USB). Data will be processed, analysed and displayed on GUI using Visual Basic. Fall is detected by simple algorithm written in C. This paper only focuses on software part. Hardware parts will only be explained in brief.

3.1.1 Feasibility Study

The first step will be studying the past and most recent articles regarding the reliability, requirements and implementation of the system. Different way of detecting fall like using ultrasound, accelerometer, gyroscope, video camera or others will be identified. Besides that, algorithms for fall detection are also studied. After studying works from others, practical surveys will be carried out on medical officers

such as doctors and nurses and elderly people who will be the user to decide the detailed design of the system.

3.1.2 Preliminary Prototype Construction

Price and specification of the components in the design are considered before making purchase. Then, the circuit will be built on the breadboard for testing before soldering it.

3.1.3 Writing Source Codes

Based on the hardware built, source codes will be written in C language because it is easy to use, understand and user-friendly. Visual Basic is used to create the GUI by importing the Visual C into it. The source code will allow the system to collect real time data and plot out the corresponding graph with the given interval, i.e. every second or every minute.

The devices that required programming are:

- Microcontroller which converts the sensors analogue output to digital input before transmitting to transceiver module.
- Reading the data transmitted to PC through USB by certain interval.
- Algorithm for detecting fall.

3.1.4 Testing and Debugging

Each stage of programming will need to be tested to see whether the output data is correct by verification process. For instant, a mass is dropped from specific height to obtain its velocities and also its acceleration. The obtained experimental values are

compared with the designed software value and check. If problem encountered, troubleshooting will be performed.

3.1.5 Data Collection and Fall Detection

After the entire source codes are verified to be working correctly, data with different motions are collected for human to analyze. The data is represented in a graph form and might be compared with other existing sources. Data will be collected repeatedly as to see the system persistency and consistency. When the data collected is good enough, fall simple detection algorithm will be developed. Before that, the data might need to be smoothened to further reduce noise generated.

3.1.6 Finalised Hardware Construction

The overall hardware including the circuits and casing will be built out after the code written has been tested and debugged. Circuit path will be drawn out using Altium and all the components will then be soldered on printed circuit board (PCB). The arrangement of components is organised nicely in order to make the hardware as small as possible to comply with non-intrusive hardware design.

3.1.7 Improvement and Development

More user-friendly and systematic code as well as GUI will be developed. In the software part, artificial intelligent (AI) might be developed also if all the data transmission and retrieval are accomplished. This AI is important and it serves as to detect and identify which raw data obtained represent the patient falls and alarming signal such as text message to their family as well as the hospital to take immediate action.

3.2 Milestone Organisation

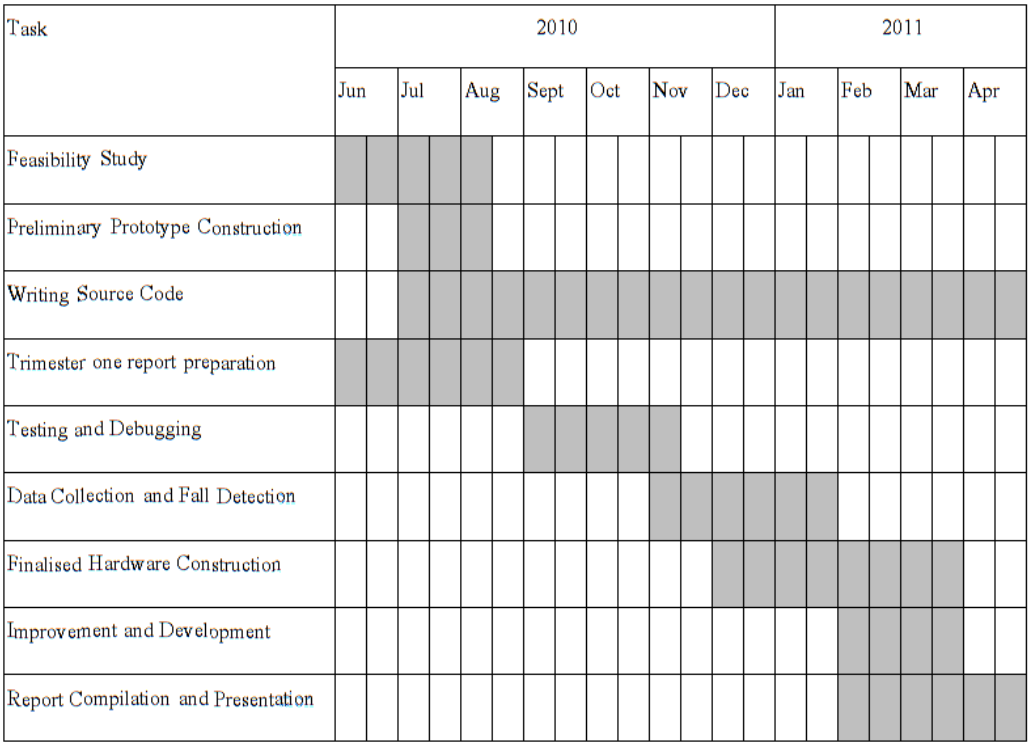


Figure 3.1: Gantt Chart

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Installing FTDI Driver

It is important to install this driver on PC so that the hardware can communicate with PC. Since SKXbee is using FTDI chip as serial port to USB converter, PC must be equipped with this driver in order to create a virtual COM port for communication among them. This driver is free and can be downloaded from a website (http://www.**ftdi**chip.com/FTDrivers.htm) and the driver installation guide also provided in the website.

4.2 Graphic User Interface (GUI) of MDS

In order to make the user interface of the MDS to be as user friendly as possible, there is not much features in the interface so that user does not need to key in many things for the configurations. Figure 4.1 shows the GUI when user double clicks on the .exe extension.

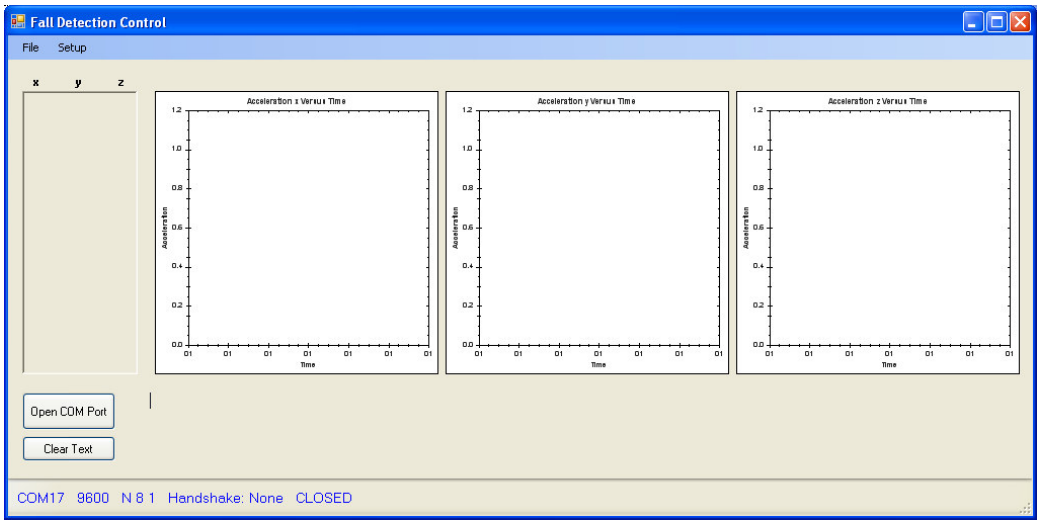


Figure 4.1: GUI of MDS

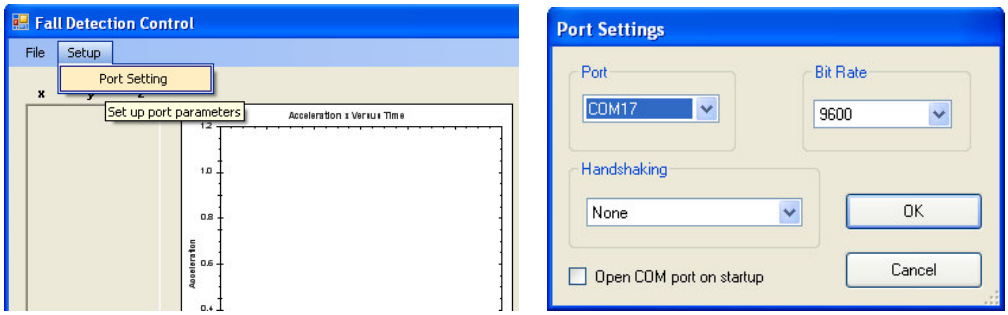
On the left, it is a text box showing the data received from the three axes accelerometer through microcontroller. There are three graphs shown in Figure 4.1, each of them representing the X, Y and Z-axis acceleration of a human body respectively. The data plotted on the graph is the direct conversion of the analogue-to-digital values representing the output voltage of the accelerometer. In order to obtain the actual acceleration values, it can be done by using the Excel file, "Acceleration Calculation", provided. The approximate acceleration value is almost linear to the output voltage produced as given in the data sheet.

Parameter	Conditions	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range		±3	±3.6		*g*
Nonlinearity	% of full scale		±0.3		%
Package Alignment Error			±1		Degrees
Interaxis Alignment Error			±0.1		Degrees
Cross-Axis Sensitivity[1]			±1		%

Figure 4.2: Accelerometer Non-linearity

At the bottom of the program, there is a tool strip status label showing the selected port which has been connected, baud rate, parity bit, number of data bits, number of stop bit, handshake and comport status (open or closed). All of these parameters can be changed except parity bit, number of data bits and number of stop

bit through setup menu strip. User can choose which port he/she would like to connect to if there is more than one com ports. User can also set the baud rate so that it synchronizes with the baud rate of SKXbee module. This interface has been specified that no parity bit is required, eight data bit to be sent and read and one stop bit. Figure 4.3 shows Port Setting user interface when setup menu is clicked.



a) b)

Figure 4.3: Port Setting User Interface when Setup is Clicked

Lastly, there are two command buttons and error display box. The functions of both command buttons are to open the selected COM port and clear text in the text box respectively. Error display box will display exceptions that are encountered in the software so that user understands what causes the error to be happened. Figure 4.4 shows that the COM port has been disconnected when the program is still running.

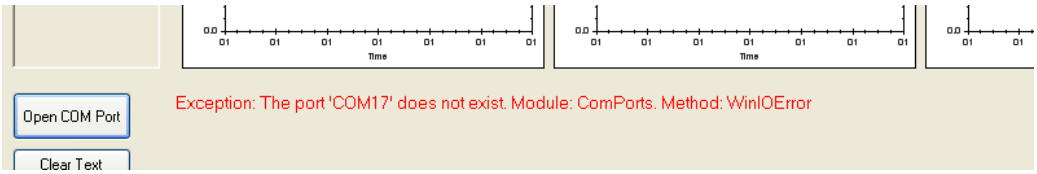


Figure 4.4: Message Showing Error Occurs

4.3 Get Started With MDS GUI

To start with the MDS, user will have to wear the processing board, which is the PIC microcontroller together with the SKXbee wireless module, on the waist as shown in the Figure 4.5. Sensor is specified to be placed at middle of the breast of user for detecting accurate motion acceleration. After putting on the hardware, user has to turn on the system by pushing the on off button on the board. When the board is powered up, user will have to double click on the software and the Fall Detection Control GUI will appear. User is required to do simple setting on the software as mentioned previously. The bit rate and handshaking option will always remain at "9600" bit rate and "None" respectively since the microcontroller and wireless module is preset into this condition. User should select which port to be opened while setting the configuration. Usually the port to be connected will display on the first item in the list box.

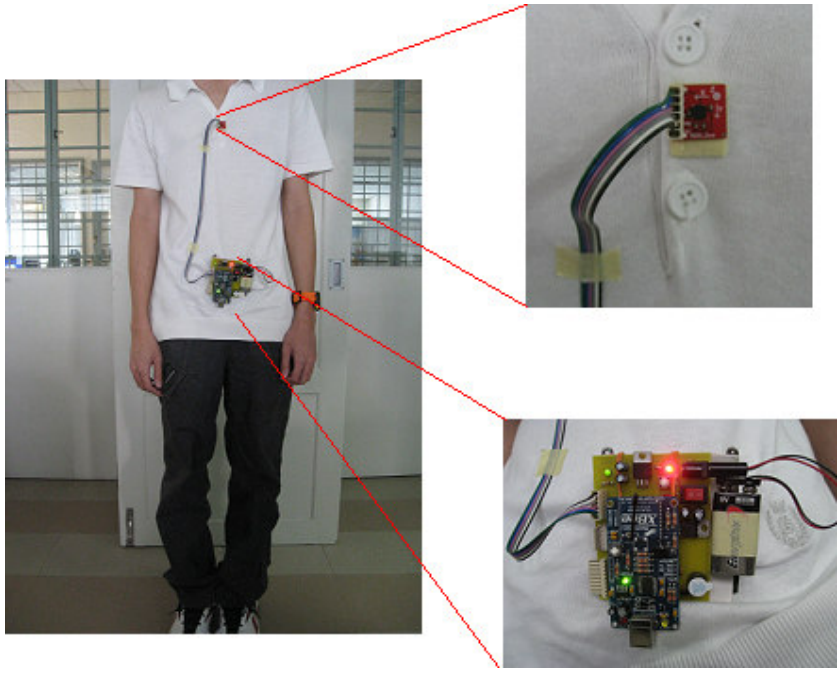


Figure 4.5: The Designated Location for Wearing the Hardware

After doing the necessary setup, user needs to click on the "Open COM Port" button and text box will be enabled at the same time. User must key in "ok" in the text box and microcontroller will starts to collect data from sensor and transmit the data to PC wirelessly.

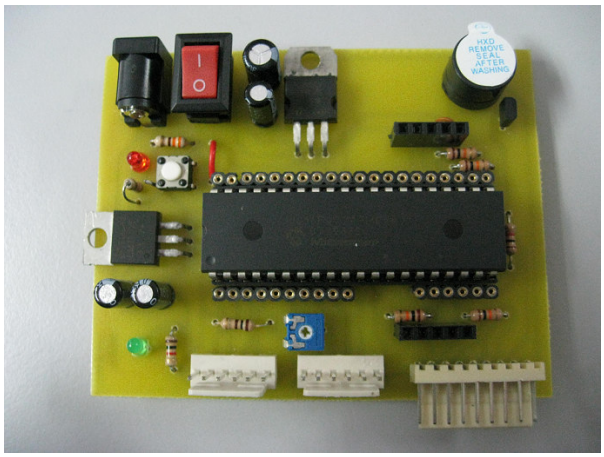


Figure 4.6: Hardware Configurations

Figure 4.7 shows the GUI when microcontroller is communicating with PC in real time. The graph plotted in GUI in real time is in discrete state and hence the points in between are not joined. In this report, the data obtained has not gone through data reconstruction state in digital signal processing.

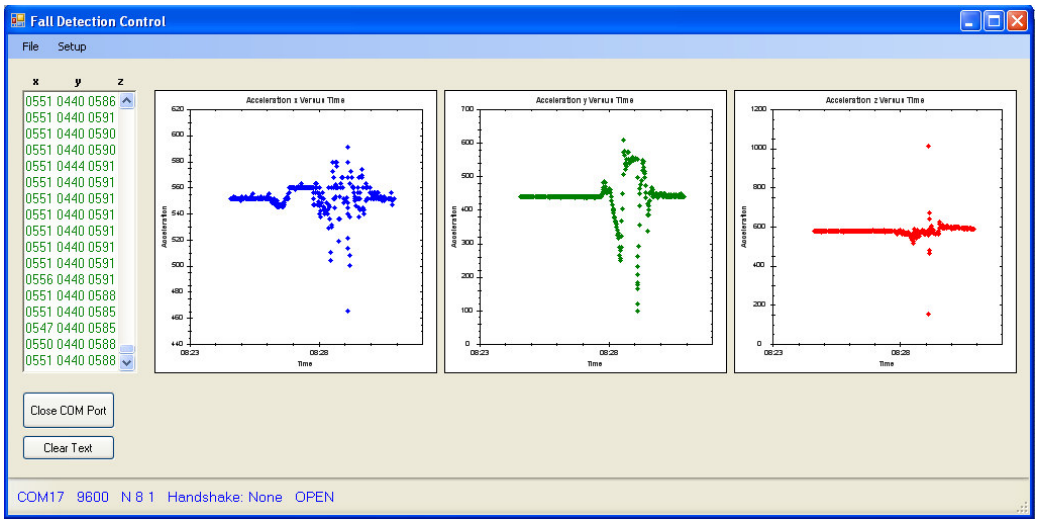


Figure 4.7: GUI Displaying Jumping Motion When the Software is Started to Communicate in Real Time

User can use any Excel package for plotting the graph offline and analyze it. User might need to do some setting before the graph can be generated in the Excel file. Below shows the step for drawing the graph:

- Start any Excel package
- Browse to save data in C:\ drive. Office button -> Open. Change File of type to All Files and select the corresponding "…Fall Detection Data Log"
- Select Delimited and click Next
- Select Delimited as Space and click Next
- Click Finish. User should see the collected data in an Excel sheet
- Highlight the X-axis column (column A), Y-axis column (column B) and Z-axis column (column C).
- Click Insert in top menu -> Line Chart -> 2D line
- Right click on the graph -> Select Data -> Click on Series 1 -> Click Edit -> Enter the series name as x
- Repeat for y and z-axes
- The graph showing the acceleration of three axes can be plotted in one as in Figure 4.8 if user would like to combine the graphs

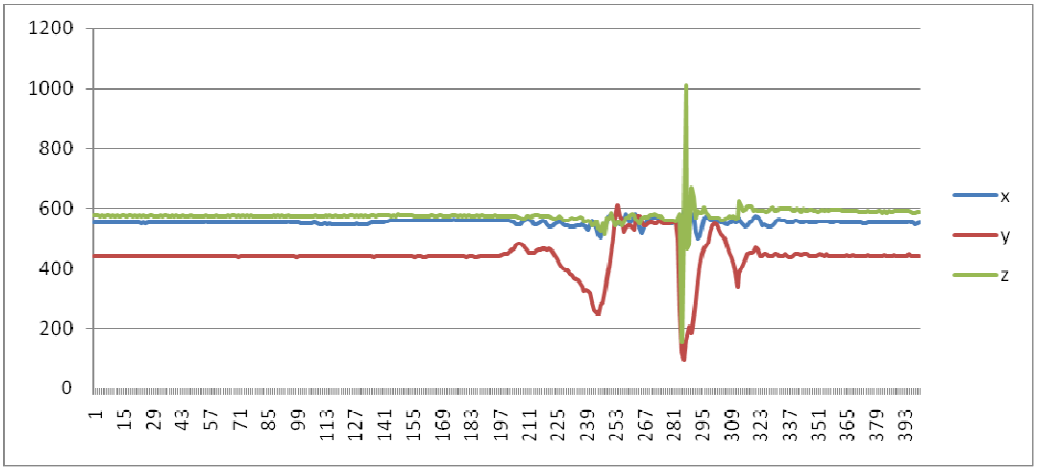


Figure 4.8: Offline Analysis of Data Collected on Jumping Motion

User can let the system to run on its own after doing the setup and MDS will monitoring the motion in real time. If fall is detected, a beep will be sounded and a message will be prompted to alert people in the house (Figure 4.9). Buzzer on the hardware will also be sounded continuously to alarm family members in the house that the elderly has fallen down. Buzzer will sound until the reset button is pressed to

disable the alarm. Besides disabling the alarm, user also has to press the reset button if he/she wishes to stop the whole system instead of only closing the software on PC. If user would like to continue with the monitoring, user has to clear the text box using the “Clear Text” button before keying in “ok” to start monitoring.

Figure 4.9: Message Prompts when there is Fall Identified

4.4 Visual Basic 2010 Source Code Explanation

SKXbee module is connected to PC via USB for more user friendly solution to ease user to explore the possible development application. There is a USB to Universal Asynchronous Receiver Transmitter (UART) converter chip on SKXbee wireless module, FTDI FT232RL, which offer easy yet reliable communication. This converter will communicate with PC by generating a virtual COM port on PC. Thus, communicating with SKXbee wireless module with PC is actually communicating through virtual COM port instead of USB.

MDS GUI is developed by using Visual Basic 2010 which is able to communicate with the wireless module via COM port driver on PC that used by FTDI FT232RL. The concept is same as communicating SKXbee with PC using Windows HyperTerminal. There are several parts to take note in the development of the GUI such as COM port communication, text to integer conversion, graph plotting and data logging. Fall detection algorithm is developed on the PIC microcontroller instead of this GUI to detect fall and send alert. The fall algorithm will be discussed in later section.

4.4.1 Loading Main Form

Once the program is executed, Form1 which is the Main Form will be loaded and the GUI will be showed. It will go through a routine for finding and accessing COM port. “MyPortSettingDialog” is defined to Class which will provide a dialog box for viewing and selecting COM ports and parameters as shown in Figure 4.3 b). A timer is used to check for any changes on the COM port at every 1000 milliseconds interval. Timer is turned off at the beginning and will be started in “SetInitialPortParameters” routine if there is no COM port saved previously.

“InitialDisplayElements” initialized the text colour in text box as shown in Figure 4.1 to be red for transmitting data. The initial port parameters are set based on the COM port being detected on the PC. COM port displayed on the GUI will always be the one lastly used by user. If the saved COM port is not detected, this program will search for other COM port on the PC and display the corresponding parameters. “No COM ports found. Please attach a COM-port device.” message will be displayed to prompt the user that there no any port being connected.

User can specify whether he/she wants to open the COM port on start up as shown in the Figure 4.10. Upon loading the Main Form, the Open COM port on start up check box will also be checked. The COM port will be open once the program is started and user does not need to click on the “Open COM Port” button. Lastly, three graphs will be initialized as well to set the desired form of appearance as well as the axes labelling when the form is loaded.

```
Private Sub Form1_Load(ByVal sender As Object, _
                       ByVal e As System.EventArgs) Handles Me.Load

    Show()

    UserPort1 = New ComPorts

    MyPortSettingsDialog = New PortSettingsDialog

    tmrLookForPortChanges.Interval = 1000
    tmrLookForPortChanges.Stop()

    InitializeDisplayElements()
```

```
    SetInitialPortParameters()

    If ComPorts.comPortExists Then
        UserPort1.SelectedPort.PortName = _
        ComPorts.myPortNames(MyPortSettingsDialog.cmbPort.SelectedIndex)

        If MyPortSettingsDialog.chkOpenComPortOnStartup.Checked Then

            UserPort1.PortOpen = UserPort1.OpenComPort()
            rtbMonitor.Enabled = True
            AccessForm("DisplayCurrentSettings", "", Color.Black)
            AccessForm("DisplayStatus", "", Color.Black)

        Else
            DisplayCurrentSettings()
        End If
    End If

    AddHandler ComPorts.UserInterfaceData, AddressOf AccessFormMarshal
    AddHandler PortSettingsDialog.UserInterfaceData, _
    AddressOf AccessFormMarshal
    AddHandler PortSettingsDialog.UserInterfacePortSettings, _
    AddressOf SetPortParameters

    CreateGraph(zg1, zg2, zg3)
End Sub
```

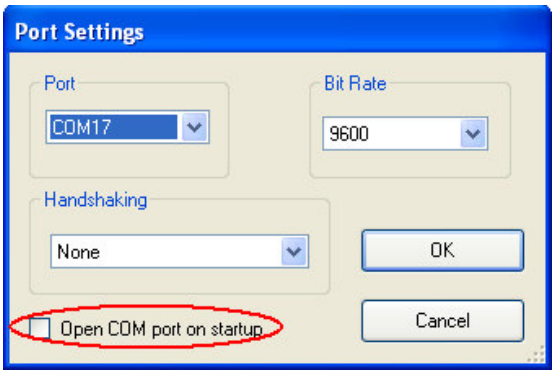


Figure 4.10: Open COM Port on Start Up on Port Setting Dialog Box

4.4.2 Input Processing

```
Private Sub rtbMonitor_TextChanged(ByVal sender As System.Object, _
                                   ByVal e As System.EventArgs) _
                                   Handles rtbMonitor.TextChanged

    ProcessTextboxInput()
End Sub
```

```
Private Sub ProcessTextboxInput()

    Dim ar As IAsyncResult
    Dim msg As String
    Dim textLength As Integer
    Dim userInput As String

    If ((rtbMonitor.Text.Length > userInputIndex + _
         UserPort1.ReceivedDataLength) And ComPorts.comPortExists) Then

        userInput = rtbMonitor.Text

        textLength = userInput.Length - userInputIndex

        userInput = rtbMonitor.Text.Substring(userInputIndex, textLength)

        msg = DateTime.Now.ToString

        ar = UserPort1.WriteToComPortDelegate1.BeginInvoke _
        (userInput, New AsyncCallback _
        (AddressOf UserPort1.WriteCompleted), msg)

        AccessForm("UpdateStatusLabel", "", Color.Black)

    Else
        UserPort1.ReceivedDataLength = 0

    End If

    userInputIndex = rtbMonitor.Text.Length
End Sub
```

4.4.3 Receiving Data

When there is data received from SKXbee via COM port, the above routine will be executed. Try...Catch statement has been used to provide a way to handle some or all possible errors that may occur in a given block of code, while still running code. If there is no error, the “RaiseEvent” will trigger the event of displaying new received data on the text box and plotting the graphs on GUI. Error message which is known as exception will be displayed in red text as shown in Figure 4.4.

```
Friend Sub DataReceived(ByVal sender As Object, _
                        ByVal e As SerialDataReceivedEventArgs)

    Dim newReceivedData As String

    Try
```

```
            newReceivedData = selectedPort.ReadExisting

            receivedDataLength += newReceivedData.Length

            RaiseEvent UserInterfaceData("AppendToMonitorTextBox", _
                                         newReceivedData, Color.Black)

        Catch ex As Exception
            DisplayException(ModuleName, ex)
        End Try
    End Sub
```

4.4.4 Appending Data Received, Plotting Graphs and Logging Data Collected

Once the event has been raised, there are three possible cases or actions can be taken to perform some defined tasks. For case "AppendToMonitorTextBox", "rtbMonitor.AppendText(formText)" functions to append the data received from SKXbee on the text box named rtbMonitor. The data collected is also being saved in a rich text file for offline analysis purpose. The graphs can be plotted out using Excel same as doing offline analysis using SD card explained in section 4.2. Figure 4.11 shows the raw data saved on C:\ drive. First, second and third column corresponds to X-axis, Y-axis and Z-axis respectively.

```
0542 0655 0546
0545 0655 0547
0547 0655 0547
0547 0655 0546
0544 0655 0545
0540 0654 0545
0547 0655 0545
0547 0655 0547
0547 0655 0547
0542 0654 0547
0544 0654 0545
0547 0655 0547
0547 0655 0547
0546 0655 0545
0542 0655 0547
0548 0655 0547
0545 0655 0545
0547 0655 0550
0543 0655 0545
0544 0655 0547
0547 0655 0547
```

Figure 4.11: Example of Data Collected and Saved in Rich Text File

The approximate acceleration of each axis can be computed by using the Acceleration Calculation Excel file provided as mentioned in section 4.1. Figure below show a value of 542 corresponds to an acceleration of 2.11m/s^2.

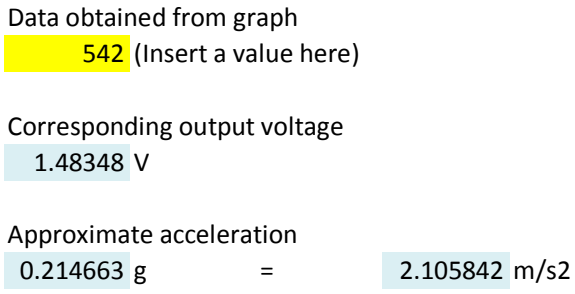


Figure 4.12: Excel File for Acceleration Calculation

The While...End While statement used in this program code session is to check whether there is a fall detected. The detection algorithm is done by the PIC microcontroller and when there is a fall detected, a character "e" will be sent out to this GUI. Once this GUI detects character "e", a beep will be sounded and a message box will prompt out alerting others that there is a fall been detected. On the hardware side, there is a buzzer attach on the board and the buzzer will sound to alert other family members in the house that the people wearing this MDS has fell down and immediate action must be taken.

"Convert1(formText.Length, formText)" converts the data received from microcontroller, which is in character form into integer so that real time graphs can be plotted in the GUI.

For case "DisplayStatus", status will always be updated through the rtbStatus in Figure 4.4. For case "DisplayCurrentSettings", port's configuration and the parameters will be displayed on ToolStripStatusLabel at the bottom of the GUI as shown in Figure 4.13.

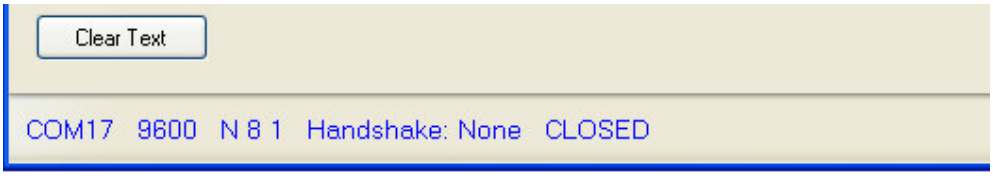


Figure 4.13: Port Setting Displayed on the Toolstripstatuslabel

```
Private Sub AccessForm(ByVal action As String, _
                       ByVal formText As String, ByVal textColor As Color)

    Select Case action

        Case "AppendToMonitorTextBox"

            rtbMonitor.SelectionColor = colorReceive
            rtbMonitor.AppendText(formText)

            SavetoRichTextFile(formText)

            While count < formText.Length And fall = False
                If formText.Substring(count) = "e" Then
                    Beep()
                    MsgBox("Fall has been detected!" + _
                           a + "Immediate action must be taken!",_
                           CType(MessageBoxButtons.OK, MsgBoxStyle), _
                           "Fall Detected!!!")
                    fall = True
                End If
                count += 1
            End While

            count = 0
            fall = False

            Convert1(formText.Length, formText)

            rtbMonitor.SelectionColor = colorTransmit

        Case "DisplayStatus"

            DisplayStatus(formText, textColor)

        Case "DisplayCurrentSettings"

            DisplayCurrentSettings()

        Case Else

    End Select
End Sub
```

4.4.5 Saving Data to Rich Text File

When doing real time detection in Continuous Mode, the data collected will be saved in a rich text file format automatically so that the data can be analyzed offline after one whole day of system running. This data is useful and crucial for analysing the fall signal for different falling down patterns and writing a reliable fall algorithm to detect fall accurately and precisely.

"StreamWriter" is a class in .NET framework library to create and write texts into a defined file. "objWriter" is declared as one type of "StreamWriter", and a valid path for the file that wants to write to is declared. "Write" is a method to write all the text into specific file. Data will be saved in C:\ drive and the file name includes the date of the data collect with "Fall Detection Data Log".

```
Private Sub SavetoRichTextFile(ByVal content As String)

    Dim fileName As String = "C:\" + DateString + _
          " Fall Detection Data Log.rtf"
    Dim objWriter As New System.IO.StreamWriter(fileName, True)
    objWriter.Write(content)
    objWriter.Close()
End Sub
```

4.4.6 Converting Data Collected in Character Form to Integer

The data transmitted from microcontroller is in the form of character as the communication between PIC and SKXbee is through UART. Microcontroller is unable to transmit the integer value of more than 8 bits and the data received could not be understood by the PC in direct integer form. What the PC can interpret is only ASCII code. Thus, microcontroller has to process analogue-to-digital data from accelerometer. The integer values are split in to four characters before PIC microcontroller sends these data to PC. As a result, conversion subroutine is needed in GUI in order to be able to plot the graphs.

“i = Convert.ToInt16(data.Substring(i2, 4))” converts four characters to one integer value. For instance, integer “0542” displayed on the text box in GUI represents characters “0”, “5”, “4”, “2” and must be converted into “real” integer value so that graphs can be plotted. After conversion, each X, Y and Z-axis acceleration graphs will be plotted with respect to time. GenerateGraph() subroutine is used to draw the graph out using ZedGraphControl.

```
Private Sub Convert1(ByRef length As Integer, ByRef data As String)

    data = storage & data
    length = length + length2

    While acc <= length And c = 1
        If length - acc = 0 Then
            c = 0
        ElseIf length - acc >= 5 Then
            i = Convert.ToInt16(data.Substring(i2, 4))
            d = d + 1
            If d = 1 Then
                GenerateGraph(zg1)
            ElseIf d = 2 Then
                GenerateGraph(zg2)
            ElseIf d = 3 Then
                GenerateGraph(zg3)
                d = 0
            End If

            i2 = i2 + 5
            acc = acc + 5
            c = 1

        Else
            acc = acc + 5
            c = 0
            i2 = 0
        End If
    End While

    If length - acc < 0 Then
        length2 = length - acc + 5
        storage = data.Substring(acc - 5, length2)
        i2 = 1
        acc = 0
        c = 1
    Else
        storage = ""
        length2 = 0
        i2 = 1
        acc = 0
        c = 1
    End If
End Sub
```

4.4.7 Plotting Graph Using ZedGraphControl

ZedGraph is an additional control in Visual Studio .NET. To add this control in Visual Basic control toolbox, zedgraph.dll and zedgraph.Web.dll files must be downloaded and added in the project. These files are available online and are a free source. Once the zedgraph.dll is added, ZedGraphControl will be seen in toolbox.

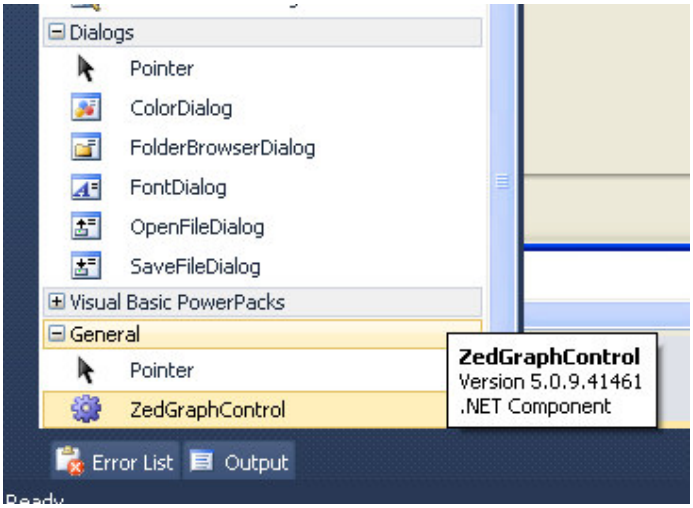


Figure 4.14: ZedGraphControl in Visual Studio Toolbox

As mentioned in section 4.3.1, graphs have to be initialized before using. Each graph is labelled properly with Acceleration, Time and the corresponding graph Title. Note that the Time-axis type must be changed to "AxisType.Date" in order to allow the GUI to show correct format of time displayed on the graph.

```
Private Sub CreateGraph(ByVal zgc1 As ZedGraphControl, _
                        ByVal zgc2 As ZedGraphControl, _
                        ByVal zgc3 As ZedGraphControl)
    Dim myPane1 As GraphPane = zgc1.GraphPane
    Dim myPane2 As GraphPane = zgc2.GraphPane
    Dim myPane3 As GraphPane = zgc3.GraphPane

    myPane1.Title.Text = "Acceleration x Versus Time"
    myPane2.Title.Text = "Acceleration y Versus Time"
    myPane1.XAxis.Title.Text = "Time"
    myPane1.YAxis.Title.Text = "Acceleration"
    myPane1.XAxis.Type = AxisType.Date
    myPane2.XAxis.Title.Text = "Time"
    myPane2.YAxis.Title.Text = "Acceleration"
```

```
        myPane2.XAxis.Type = AxisType.Date
        myPane3.Title.Text = "Acceleration z Versus Time"
        myPane3.XAxis.Title.Text = "Time"
        myPane3.YAxis.Title.Text = "Acceleration"
        myPane3.XAxis.Type = AxisType.Date
    End Sub
```

A list of point pairs are generated in plotting graph. "list.Add(DateTime.Now.ToOADate, Convert.ToDouble(i))" add new pair of points into the list. The time and acceleration values are horizontal and vertical axis respectively. X, Y and Z-axis acceleration are plotted accordingly and the appearance of each curve is defined. X-axis acceleration will have blue colour curve, Y-axis and Z-axis acceleration will have green and red curve respectively. The graph will be updated each time this routine is called to plot real time graph.

```
    Private Sub GenerateGraph(ByVal zgcvalue As ZedGraphControl)
        Dim myPane As GraphPane = zgcvalue.GraphPane
        Dim list As PointPairList = New PointPairList

        list.Add(DateTime.Now.ToOADate, Convert.ToDouble(i))


        If d = 1 Then
            Dim myCurve As LineItem = _
                myPane.AddCurve("", list, Color.Blue, SymbolType.Circle)
            myCurve.Symbol.Fill = New Fill(Color.Blue)
        ElseIf d = 2 Then
            Dim myCurve As LineItem = _
                myPane.AddCurve("", list, Color.Green, SymbolType.Circle)
            myCurve.Symbol.Fill = New Fill(Color.Green)
        ElseIf d = 3 Then
            Dim myCurve As LineItem = _
                myPane.AddCurve("", list, Color.Red, SymbolType.Circle)
            myCurve.Symbol.Fill = New Fill(Color.Red)
        End If

        zgcvalue.AxisChange()
        zgcvalue.Invalidate()
    End Sub
```

4.5 MPLAB Source Code Explanation

PIC microcontroller needs to be programmed in order to sample the analogue data from accelerometer to digital form and communicate with SKXbee through UART. MPLAB IDE is used to write the program in C and HI-TECH C Compiler is used to compile the program and to generate the Hex file. It is important to know how PIC

microcontroller is programmed so that communication between GUI and PIC can be fully understood.

4.5.1 Receiving Command to Start Communicate with GUI

PIC microcontroller will only start to communicate with GUI when “ok” characters are entered by user. This is done by using While statement as shown.

```
unsigned char a;

      while(1)
      {
            a = receive();

            if (a == 'o')
            {
                  a = receive();
                  if (a == 'k') break;
            }
      }
```

4.5.2 Convert Analogue Data from Accelerometer into Digital Data

Referring to PIC16F877A datasheet, there are total of eight modules (RA0, 1, 2, 3, 5, RE0, 1, 2) of analogue to digital (A/D) converter. In the hardware design, RA3 is connected to a positive 2.8V to provide reference voltage for all the A/D modules due to the reason that accelerometer only produces output voltage range from 0 – 2.8V. RA0, 1, 2 is connected to the X, Y and Z-axis of the accelerometer respectively. RE0, 1, 2 have been reserved for second accelerometer in the future development.

Data is sampled for X, Y and Z-axis accordingly and are stored in an integer array called “SData[]”. Delay is necessary as to allow PIC has sufficient acquisition time and conversion time. To access RA0, the combination of CHS2, CHS1 and

CHS0 is “000”. A/D module will start the conversion once ADGO is set to 1. After the conversion time, converted data will be stored in ADRESL and ADRESH registers. Data has been configured to be right-justified, making six most significant bits of ADRESH read as “0” (Figure 4.15). Combination to access RA1 and RA2 is “001” and “010” respectively (Please refer to PIC datasheet).

```
unsigned int temp;
CHS2=0;CHS1=0;CHS0=0;
DelayUs(20);
ADGO=1;
DelayUs(20);
if(ADIF)
{
      SData[0]=ADRESL;
      temp=ADRESH*0b100000000;
      SData[0]+=temp;
}
```

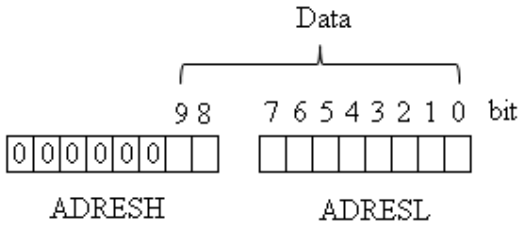


Figure 4.15: Visualization on Data Stored in ADRESL and ADRESH

4.5.3 Conversion from Characters to Integer and Sending Characters to GUI to Display on Text Box

To convert integer to characters, the first thing to do is to split the integer value into one, tenth, hundredth, and thousandth. This can be done by dividing the integer by 10, 100, 1000 and taking the corresponding remainder values. Below shows an example of splitting integer value of 1023.

$$\frac{1023}{10} = 102 \text{ remainder } 3 \quad \text{(one)}$$

$$\frac{102}{10} = 10 \text{ remainder } 2 \quad \text{(tenth)}$$

$$\frac{10}{10} = 1 \text{ remainder } 0 \quad \text{(hundredth)}$$

$$\frac{1}{10} = 0 \text{ remainder } 1 \quad \text{(thousandth)}$$

This simple logic is written in code below. SData[0] is an integer array holding the X-axis data, the data is stored in a temporary register named "Data" in performing the splitting process. To convert the split integer value into ASCII character, it is as simple as adding the value by hexadecimal value of 30h. "display()" function in the code below sends the character to PC and displays it on text box. This routine is repeated for Y and Z-axis data (SData[1] and SData[2] respectively) and this process will be repeated again and again until there is fall detected, then only the PIC will stop getting data from accelerometer.

```
unsigned int one;
unsigned int ten;
unsigned int hundreds;
unsigned int thousands;
unsigned int Data;

      Data=SData[0]/10;
      one=SData[0]%10;
      ten=Data%10;
      Data=Data/10;
      hundreds=Data%10;
      Data=Data/10;
      thousands=Data%10;
      display(0x30+thousands);
      display(0x30+hundreds);
      display(0x30+ten);
      display(0x30+one);
```

Dec	Hx	Oct	Html	Chr
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9

Figure 4.16: Part of ASCII Lookup Table Showing the ASCCI Code for Character "0-9"

4.5.4 Fall Algorithm

Fall can be identified if all three axes acceleration values have exceeded the specified threshold values. Thresholds are set based on the experimental results being carried out and it will be discussed in the next section. Square root function has been used to calculate the magnitude of the total acceleration of body and math.h header must be included in order to use this function.

```
#include <math.h>

            unsigned int Threshold = 1112;
            unsigned long x;
            unsigned long x2;
            unsigned long y;
            unsigned long y2;
            unsigned long z;
            unsigned long z2;
            unsigned int calculated_acceleration;

                  x=SData[0];
                  x2=x*x;
                  y=SData[1];
                  y2=y*y;
                  z=SData[2];
                  z2=z*z;
                  calculated_acceleration=(unsigned
                              int)round(sqrt((x2+y2+z2)));
                  if(calculated_acceleration >= Threshold)
                  {
                        display(0x65);
                        while(1)
                        {
                              RD1=1;
                        }
                  }
```

4.6 Experimental Results

It is hard and impossible to carry out the experiments on elderly since “near to real fall” actions have to be done in order to obtain accurate and precise data. Hence,

experiments have been carried out on two young persons, Lim Yu Hung (author) and Koo Yee Lan. Both are in the age of 22 to 23. The number of experimental results obtained is small due to time constraint which has been spent on developing the whole system till collecting data.

Data have been collected for daily activities such as walking, sitting down, squatting down and standing up back and jumping (refer to Appendix). The paramount is to obtain falling data. Suitable thresholds that will indicate a fall will then be obtained. Thresholds finding and setting will be discussed in brief in this report. Three types of falling patterns, back falling, left-side falling and right-side falling, have been simulated. Each falling patterns are repeated three times to obtain the average thresholds. Front falling has not been simulated due to reason that the hardware is to be wear at front position (Figure 4.17) and it might damage the hardware. If front falling were to be simulated, proper casing would have to be designed so that it will protect the hardware and at the same time not hurting the user who is wearing it during fall.

Figure 4.17: Left-Side and Right-Side Falling Motion

The threshold value can based on total acceleration of body according to Dai et al. (2009) paper.

$$|a_T| = \sqrt{|a_x|^2 + |a_y|^2 + |a_z|^2} \tag{4.1}$$

Where

a_T = total acceleration of body in digital code

a_x = X-axis acceleration in digital code

a_y = Y-axis acceleration in digital code

a_z = Z-axis acceleration in digital code

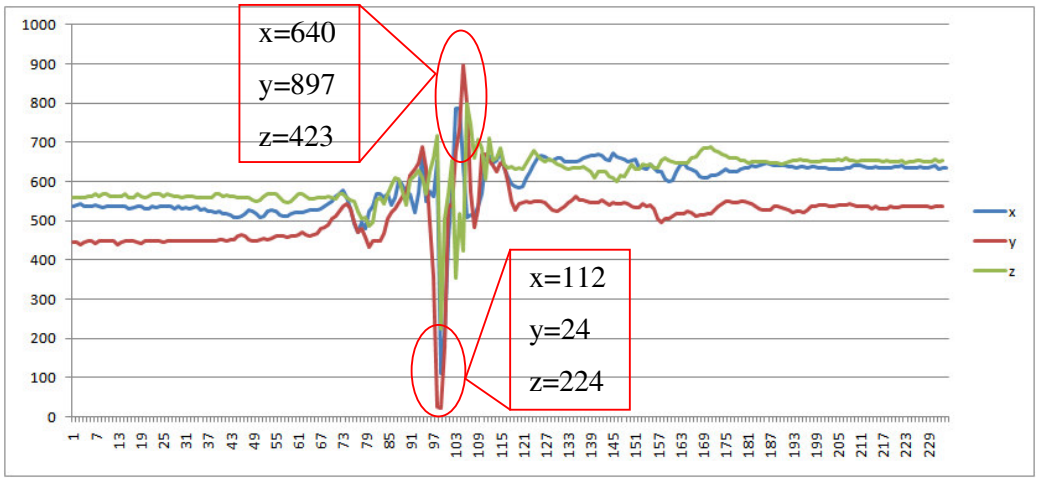
4.6.1 Back Falling Result

$$a_{1(\max)} = \sqrt{640^2 + 897^2 + 423^2} = 1180.31 \approx 1180$$
$$a_{1(\min)} = \sqrt{112^2 + 24^2 + 224^2} = 251.59 \approx 252$$

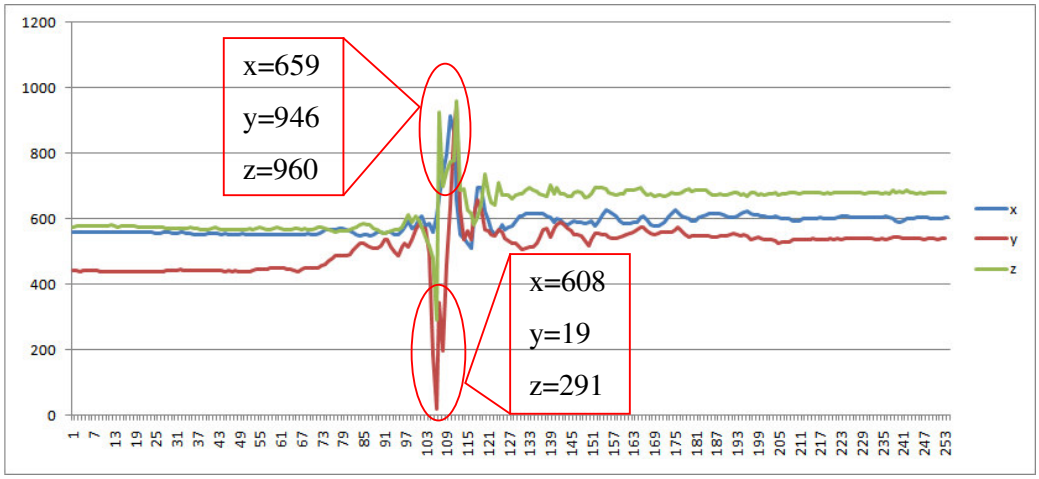
$$a_{2(\max)} = \sqrt{659^2 + 946^2 + 960^2} = 1500.27 \approx 1500$$
$$a_{2(\min)} = \sqrt{608^2 + 19^2 + 291^2} = 674.32 \approx 674$$

$$a_{3(\max)} = \sqrt{744^2 + 496^2 + 783^2} = 1188.55 \approx 1189$$
$$a_{3(\min)} = \sqrt{504^2 + 28^2 + 206^2} = 545.19 \approx 545$$

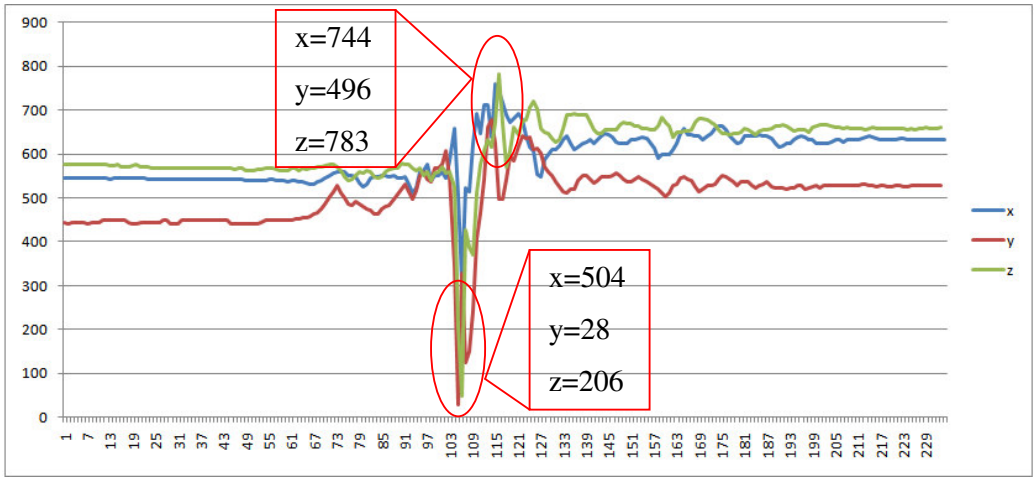
$$a_{average(\max)} = \frac{1180 + 1500 + 1189}{3} = 1289.67 = 1290$$
$$a_{average(\min)} = \frac{252 + 674 + 545}{3} = 490.33 \approx 490$$


x=640
y=897
z=423
x=112
y=24
z=224
x
y
z

a)


x=659
y=946
z=960
x=608
y=19
z=291
x
y
z

b)


x=744
y=496
z=783
x=504
y=28
z=206
x
y
z

c)

Figure 4.18: Three Experimental Results for Back Falling with X, Y, Z-Axis

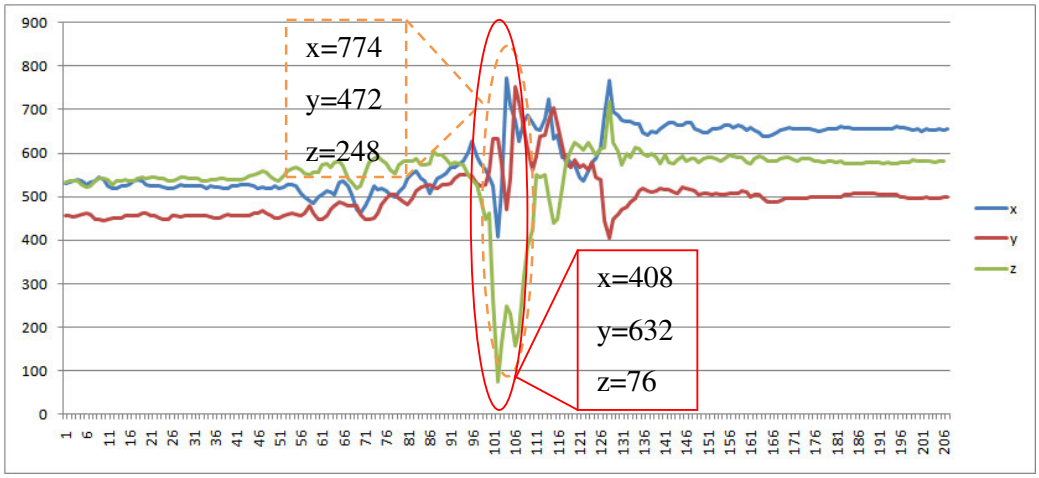
4.6.2 Left Falling Result

$$a_{1(\max)} = \sqrt{774^2 + 472^2 + 248^2} = 939.87 \approx 940$$
$$a_{1(\min)} = \sqrt{408^2 + 632^2 + 76^2} = 756.08 \approx 756$$

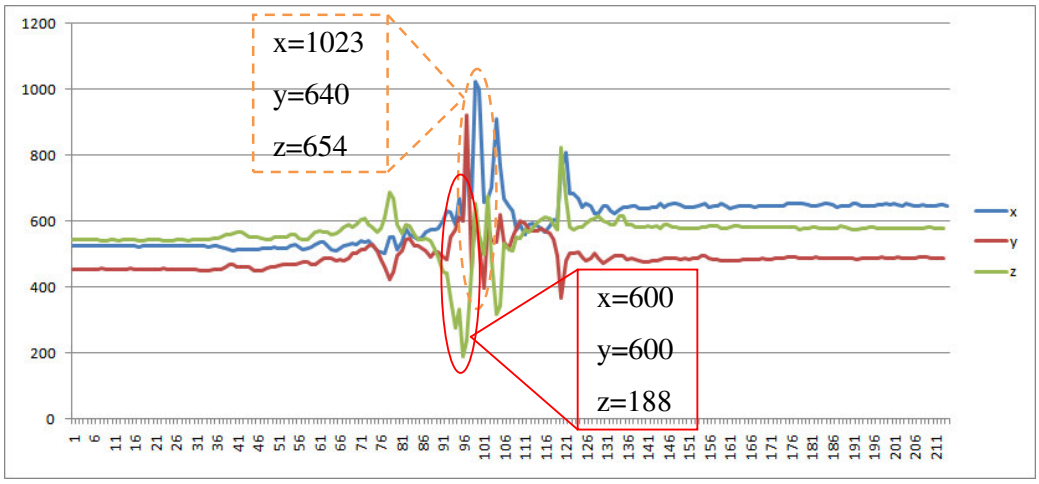
$$a_{2(\max)} = \sqrt{1023^2 + 640^2 + 654^2} = 1372.53 \approx 1373$$
$$a_{2(\min)} = \sqrt{600^2 + 600^2 + 188^2} = 869.11 \approx 869$$

$$a_{3(\max)} = \sqrt{896^2 + 487^2 + 284^2} = 1058.60 \approx 1059$$
$$a_{3(\min)} = \sqrt{608^2 + 611^2 + 252^2} = 898.05 \approx 898$$

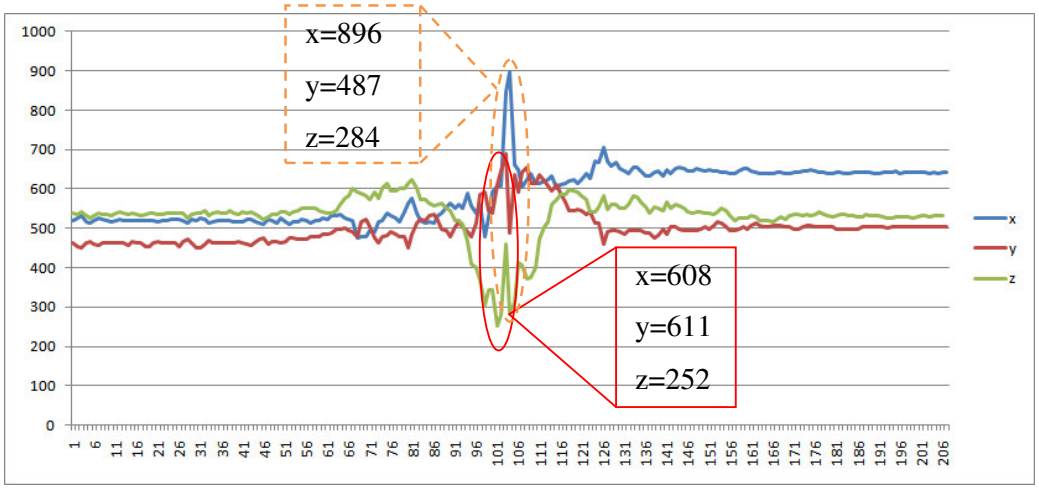
$$a_{average(\max)} = \frac{940 + 1373 + 1059}{3} = 1124$$
$$a_{average(\min)} = \frac{756 + 869 + 898}{3} = 841$$



a)



b)



c)

Figure 4.19: Three Experimental Results for Left-Side Falling with X, Y, Z-Axis

4.6.3 Right Falling Result

$$a_{1(\max)} = \sqrt{417^2 + 586^2 + 819^2} = 1089.98 \approx 1090$$
$$a_{1(\min)} = \sqrt{372^2 + 192^2 + 527^2} = 673.04 \approx 673$$

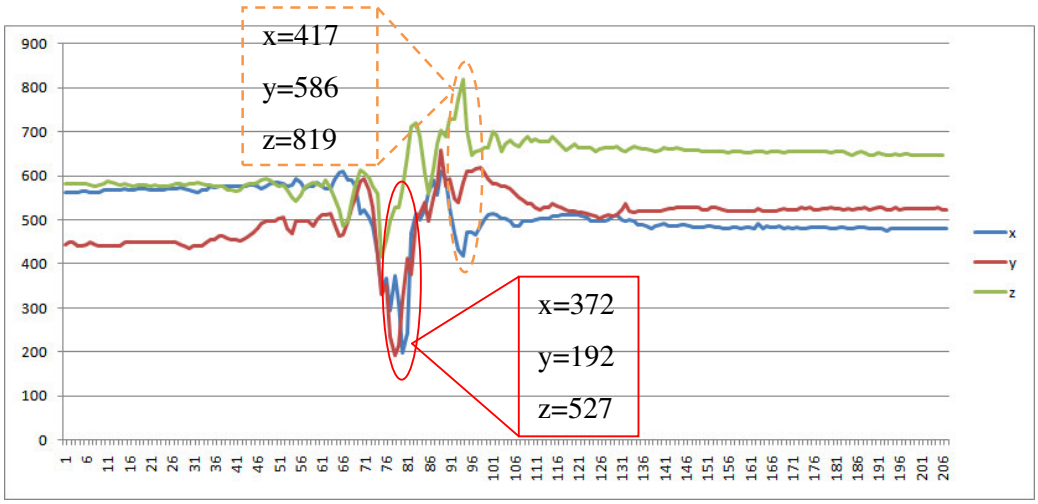
$$a_{2(\max)} = \sqrt{433^2 + 432^2 + 1016^2} = 1185.90 \approx 1185$$
$$a_{2(\min)} = \sqrt{327^2 + 204^2 + 792^2} = 880.80 \approx 881$$

$$a_{3(\max)} = \sqrt{236^2 + 159^2 + 1022^2} = 1060.88 \approx 1061$$

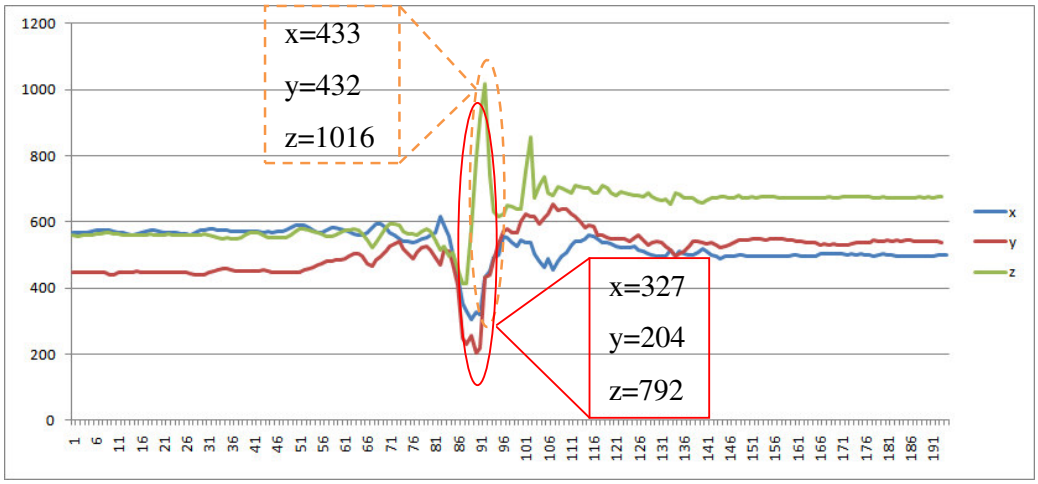
$$a_{3(\min)} = \sqrt{312^2 + 112^2 + 568^2} = 657.66 \approx 658$$

$$a_{average(\max)} = \frac{1090 + 1185 + 1061}{3} = 1112$$

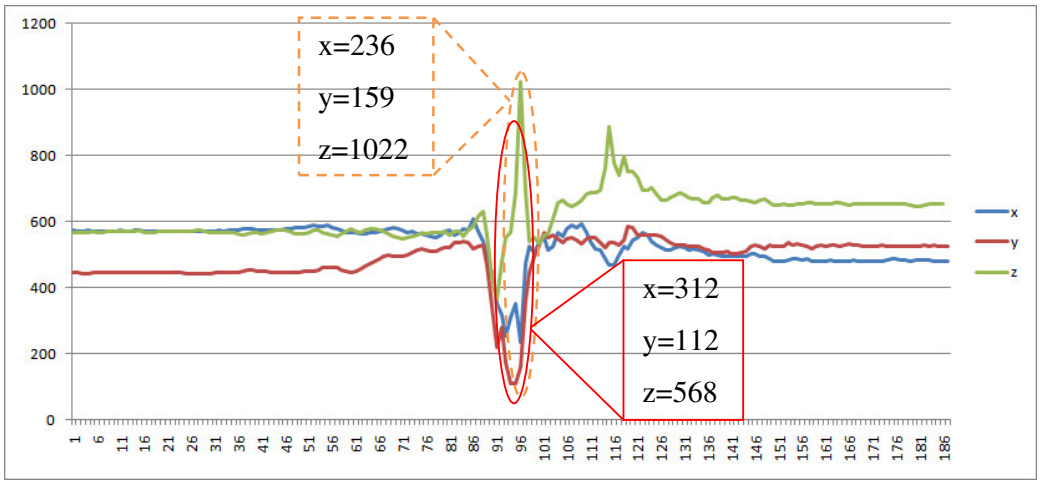
$$a_{average(\min)} = \frac{673 + 881 + 658}{3} = 737.33 \approx 737$$



a)



b)



c)

Figure 4.20: Three Experimental Results for Right-Side Falling with X, Y, Z-Axis

4.6.4 Determining Threshold Value

After analysing a series of data obtained as in previous section, the appropriate threshold value can be determined. From the processed data, it can be seen that the maximum total acceleration values for three types of falling are very close as compare to the minimum total acceleration values. Thus, the threshold is set based on the maximum for simplicity without sacrificing the accuracy. The smallest average maximum total acceleration value is set as the threshold in the hardware so that the system is able to detect three falling types. Average values have been taken instead of the minimum value from the experiments as to compensate for allowable tolerance in the real environment.

$$\text{Threshold} = 1112$$

4.7 Problem Encountered

Throughout the construction of the hardware and software design, there are many problems faced. Corrective steps have been taken to solve and improve the overall system.

4.7.1 Conversion from Characters to Integer

One of the problems is the conversion from character to integer algorithm on GUI. The first approach used to perform conversion is by reading the raw data obtained and displayed on the text box and convert four characters into one integer by incrementing the text box length of the text box.

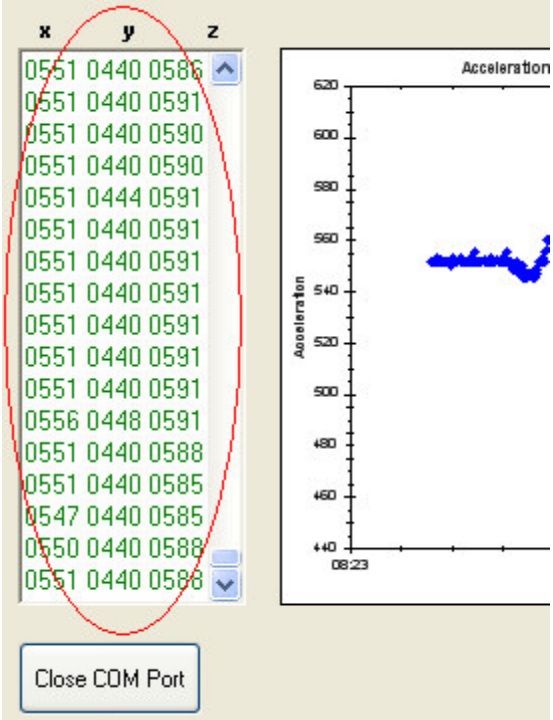


Figure 4.21: Using Data in Text Box to Perform Character to Integer Conversion

Text box length is being assigned by a long variable as shown below, which can carry up to 64-bit signed integer value and this value will eventually reach its

maximum value (9223372036854775807) and overflow will occur. When overflow occurs, no more conversion can be done and the graphs would not be plotted out as integer values are needed for the graphs plotting.

```
Dim length2 As Long = 0
                .
                .
                .
            i = Convert.ToInt16(rtbMonitor.Substring(i2, 4))
```

As shown in the code above, the conversion is based on the data displayed on rtbMonitor text box and these values will be used to plot graphs. To solve the overflow problem, conversion routine is modified to convert characters received immediately from the COM port instead of the characters displayed on text box. This method is proved to be working since the length of data (one packet) that transmitted to COM port each time is relatively small, below ten characters. Hence, the length of the packet will not exceed the maximum amount of neither long nor integer variables and each time the data is transmitted, the length of the string will be reset instead of keep on increasing as in the case in text box.

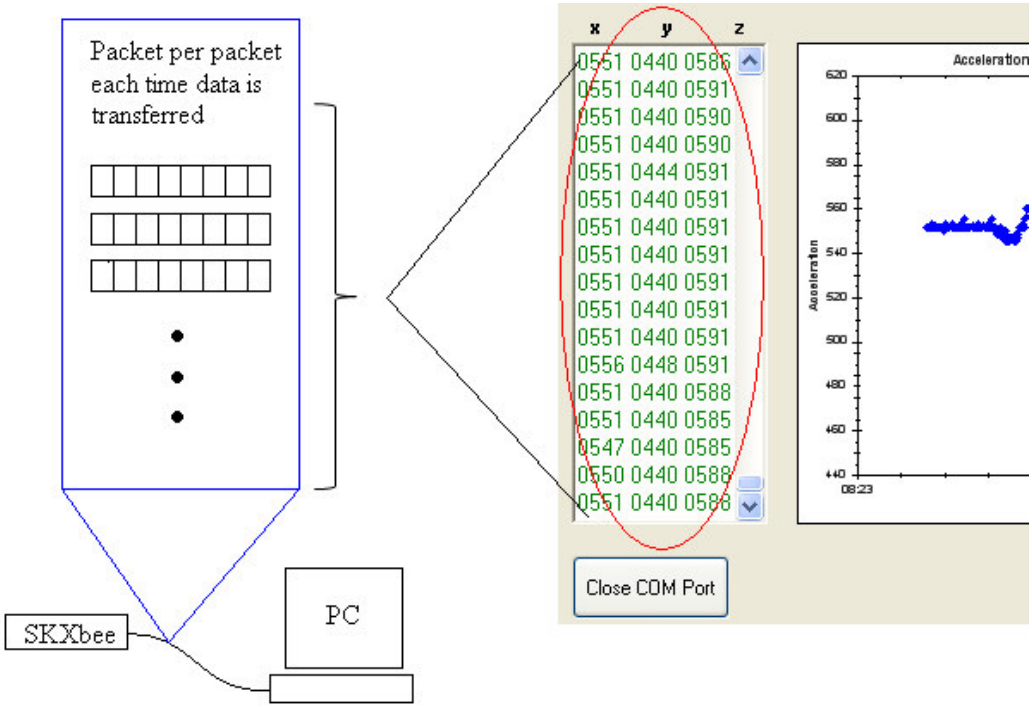


Figure 4.22: Data Transferred in Packet

```
Dim length2 As Integer = 0
                    .
                    .
                    .
              i = Convert.ToInt16(data.Substring(i2, 4))
```

The length of the data received in one packet can be represented by integer instead of long variable. This will not only solve the problem of overflow but also improving the processing power required by the program.

4.7.2 Signal Noises

There are appearances of unknown noise and spike at the very beginning of data collection (Figure 4.23). This directly influenced accuracy of the system. Therefore, several steps are taken in order to find out the reason which may cause by wireless module or microcontroller.

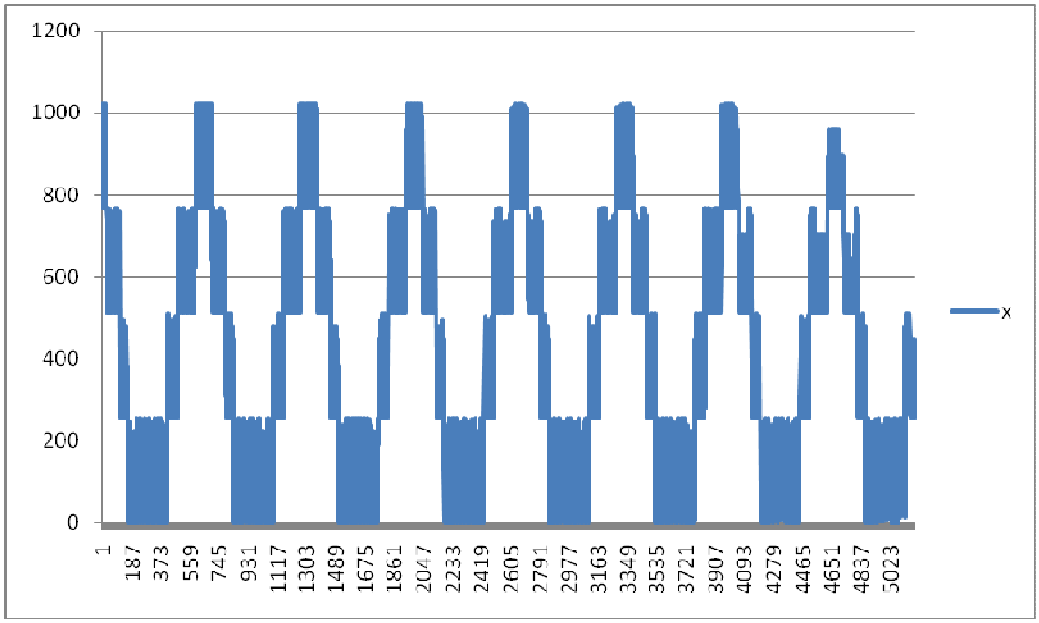


Figure 4.23: Inaccurate Data Obtained at the Beginning

First and foremost, accelerometer is replaced by signal generator to diagnosis wireless module. However, the obtained waveform got lots of noise and the output values somehow were different from input values. So, microcontroller part is suspected to be the source of problem.

Nevertheless, the result was same as previous data obtained. No matter how the input frequency and the input amplitude are adjusted, data collected still not varied a lot. After a series of testing, the data collected was found to be shifted in between three axes acceleration values. By referring back to the PIC microcontroller sheet, it is found that the data conversion time specified was insufficient. Thus, longer delay time was introduced and eventually the output waveform was same with the input waveform but with appearance of noise.

Then, suggestion was given to check the grounding of each component because improper grounding might cause noise to output. So, all the unused analogue pins are being connected to ground. As a result, unwanted noise is reduced on the output waveform. The unused analogue pins can also be configured to become digital pins to give the same effect as hardware grounding.

Finally, the system was tested again with accelerometer and proper and accurate data was obtained (Figure 4.26, 4.27, 4.28). The following figures show the basic idea of troubleshooting process.

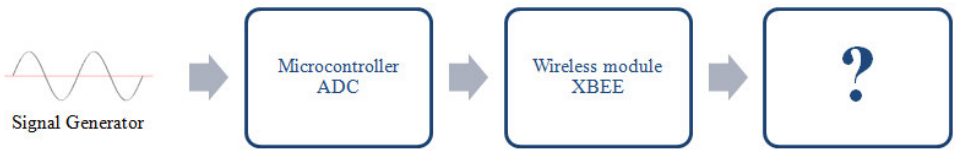


Figure 4.24: Wireless Module Troubleshooting

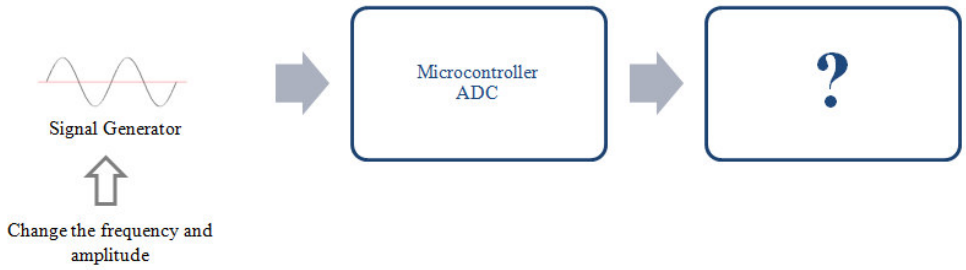


Figure 4.25: Microcontroller Troubleshooting

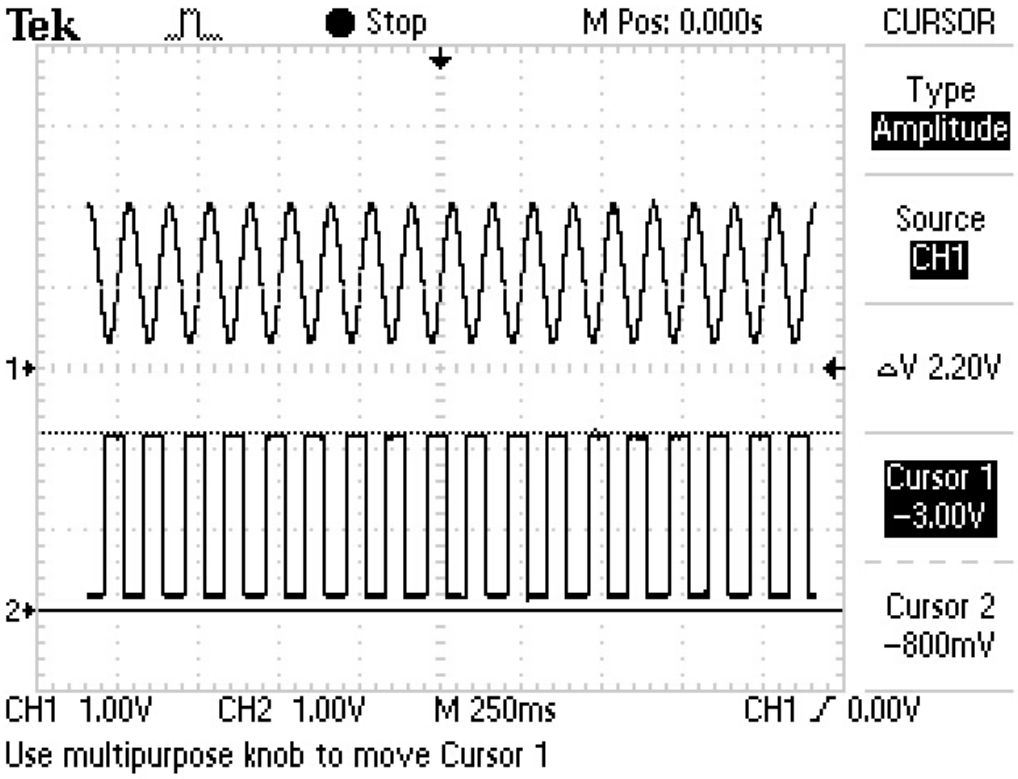
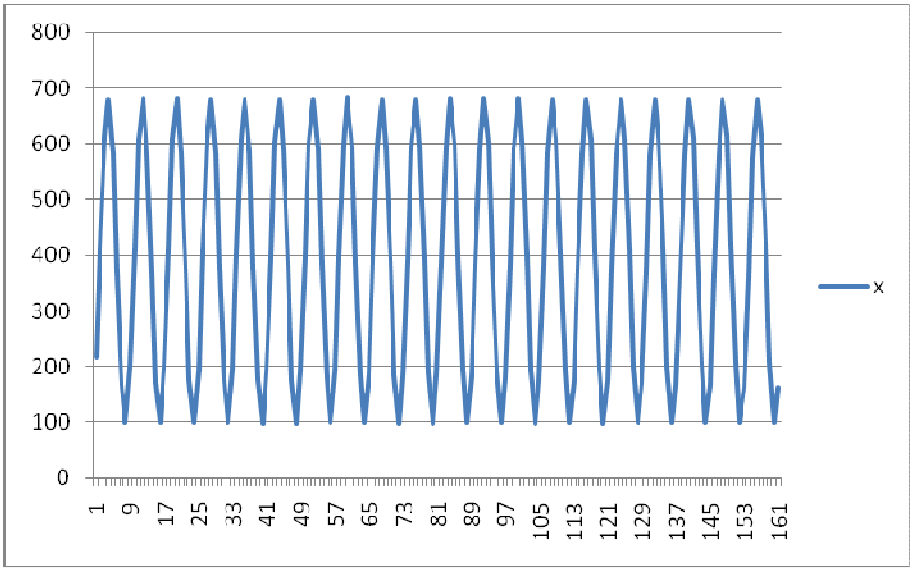


Figure 4.26: Data from Two Signal Generator with 1.76V_{pp} for Sine Wave and 2V_{pp} for Square Wave at 8Hz

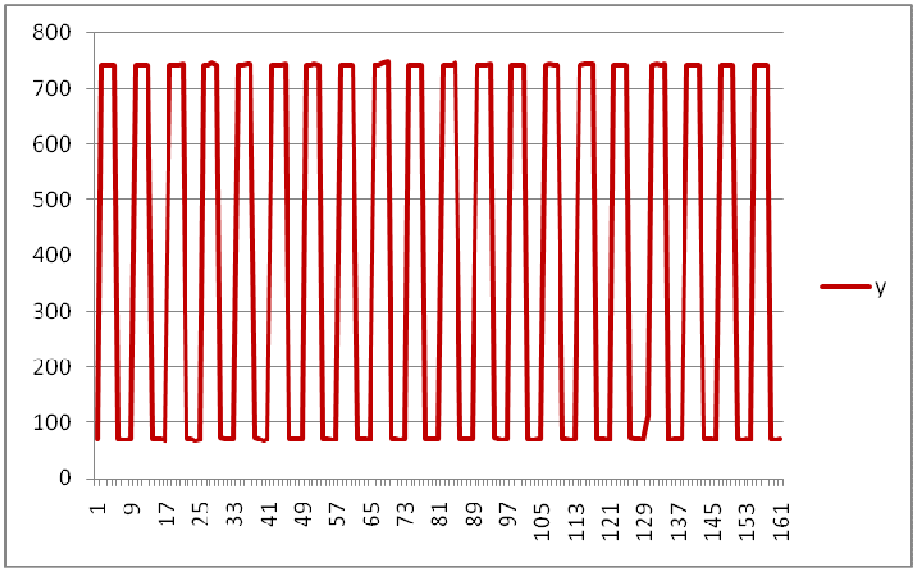


a)

	actual	calculated
x(min)	0.32	0.28418
x(max)	2.04	2.003906

b)

Figure 4.27: a) Data (Sine Waveform) after A/D Conversion to PC Through Skxbee b) Data Comparison between the Actual (Signal Generator) with the Calculated (Skxbee)



a)

	actual	calculated
y(min)	0.16	0.196289
y(max)	2.2	2.191406

b)

Figure 4.28: a) Data (Square Waveform) after A/D Conversion to PC through Skxbee b) Data Comparison between the Actual (Signal Generator) With the Calculated (Skxbee)

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Recommendations

5.1.1 Reducing Power Consumption

Improvement should be done to allow user to choose to send the data continuously to the PC or only send the alert to the PC when there is a fall detected. This can be done by doing some setting on the hardware. To choose to send data continuously, user will only need to plug the connector to pin 1 and pin 9 counting from right top of the microcontroller. If user prefers to alert the PC when there is fall, unplug the whole connector from the microcontroller. Figure 5.1 shows how the connection would be.

The main reason why this system allows user to select different modes is to improve the power management and reduce power consumption on the hardware. When using continuous mode, part of the power is used by SKXbee wireless module and interfacing between wireless module with microcontroller in transmitting data. As state in the datasheet of SKXbee wireless module, the typical operating current consumed when transmitting data is 45mA and receiving data is 50mA. As for PIC microcontroller, the maximum current sourced by PORTC and PORTD is 200mA, averagely 100mA for each PORTC and PORTD. Since microcontroller is connected with wireless module through receive and transmit bit (PORTC pin 6 and 7), the maximum current consumed is also quite high. Based on the estimation above, Alert Mode is proposed to reduce power consumption and increase the battery lifespan.

However, the above statement is purely based on theoretical analysis. Experimental data shall be conducted in future.

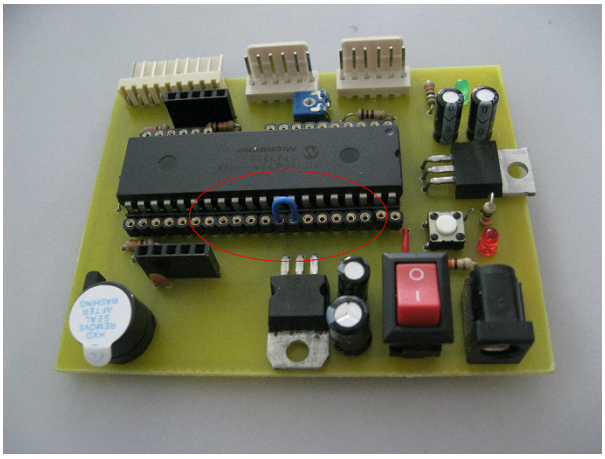


Figure 5.1: Continuous Mode Configuration if Alert Mode is Implemented

Besides that, PIC microcontroller should be configured in sleep mode when user is sitting down or did not move for some time (resting or taking nap). However, SKXbee can only communicate in Asynchronous mode as stated in the data sheet. In order to save more power, hardware selection would also be considered. Choosing other wireless module that could communicate in Synchronous mode allow PIC microcontroller to transfer or receive data in sleep mode. Funding is also an issue that directly affect the system power consumption because advance hardware will allow MDS to have lower power consumption. With limited funding, the best hardware that can suit for this application will be the one used in this project.

5.1.2 Storing Data in Secure Digital (SD) Card

When whole system was being designed and constructed, there are some issues and problem faced. One of the issues that will affect the reliability of MDS is data loss. Since data is transmitted through wireless module, some of the data will somehow loss due to environment disturbance and obstacles like wall that interfere the data

transmission. The indoor communication range of SKXbee that will cause no data loss is far more less than 30 metres as in datasheet due to the reason stated above.

Due to this problem, it is more reliable to save the data in a SD or even micro SD card that can be carried by user together the hardware. Effort has been put on adding SD card feature on the real hardware. However, the source code for creating a text file and storing data in SD card could not be finished due to limitation of time. Hardware port has been reserved in hardware as shown in Figure 5.2. By implementing this feature, it will not only help to solve data loss problem, it also save power and extend the battery life. Data stored in SD card can then be taken out to analyze on PC using the same method as mentioned in section 4.3. This is a very important feature that must be implemented in future development.

However, it is proposed to use PIC 18F family to perform this task as Writing to SD card is based on Serial Port Interface (SPI) which may consume quite a lot of the microcontroller Random Access Memory (RAM).

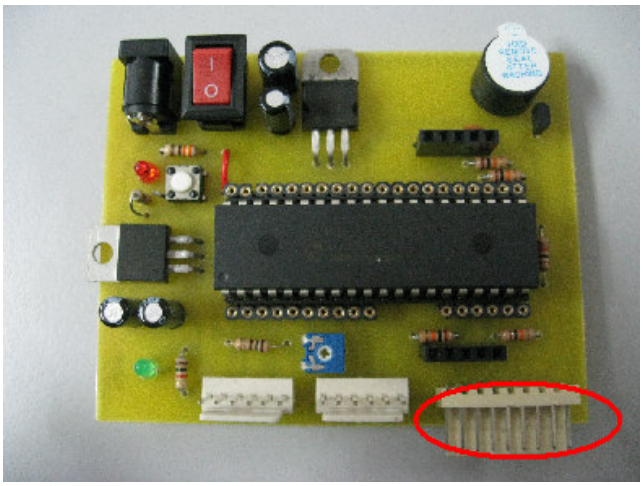


Figure 5.2: Reserved Port for Connecting to SD Card

5.1.3 Improve Fall Detection Accuracy Using Additional Gyroscope

There are some motions not in fall category but still triggers the fall detection alarm. One of the non-fall motions identified is jumping, which is very seldom to be performed by elderly. In order to increase the accuracy of fall, it is suggested to use another gyroscope combining with accelerometer to obtain the body orientation and velocity as proposed in many papers like Almeida et al. (2007) and Dai et al. (2009) papers. By using this additional information together with acceleration value, fall can be identified more accurately. However, more computation power will be needed to run the fall algorithm.

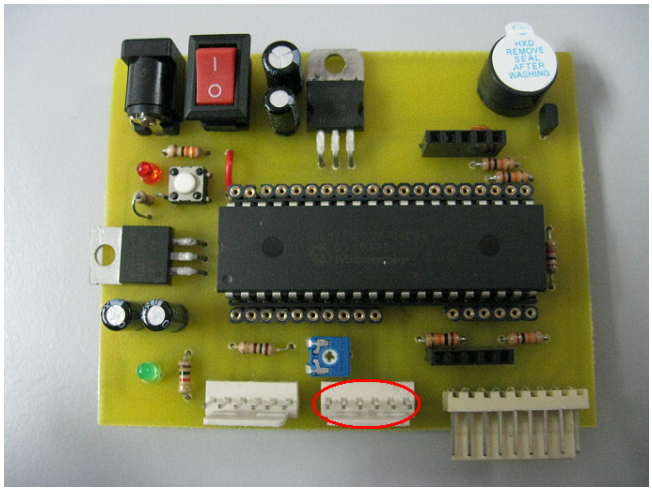


Figure 5.3: Reserved Port for Connecting to Gyroscope

5.1.4 Instantaneous Graph Plotting

There is an issue concerning the real time graph plotting using Zedgraph control in VB 2010. The graphs showing X, Y and Z-axis acceleration should display instantaneous graphs upon receiving data from the microcontroller according to the source code written. However, only the Z-axis acceleration graph being refreshed on real time while X and Y-axis did not shows the updated graphs despite data have been updated in the graph. This could be due to the VB program and Zedgraph control that could not be updated in real time regardless of the PC RAM.

5.2 Conclusion

In conclusion, the aim and objectives of this project have been achieved with some improvement to be done in future. User friendly and simple GUI using Visual Basic 2010 has been developed together will simple and effective fall algorithm (MPLAB) to detect real fall. Several improvements can be done as stated in the recommendations to make this MDS a more reliable system to user. The MDS hardware can be shrunk much smaller (coin size) size using surface mount components instead of through-hole to prevent intrusion to user.

The overall cost to develop MDS is only at about RM700 – RM800 (including circuit testing components). This cost could be reduced if this product is being mass produced to make it affordable to each family.

REFERENCES

Perolle, G., Fraisse, P., Mavros, M., Etxeberria, I. (2006) Automatic Fall Detection and Activity Monitoring for Elderly.

Department of Health, Social Services and Public Safety UK. (2004). Home Accident Prevention Strategy and Action Plan 2004-2009.

Tarik Al-ani, Quynh Trang Le Ba and Eric Monacelli. (2007). On-line Automatic Detection of Human Activity in Home Using Wavelet and Hidden Markov Models Scilab Toolkits.

Blount, M., Batra, V. M., Capella, A. N., Ebling, M. R., Jerome, W. F., Martin, S. M. et al. (2007). Remote health-care monitoring using Personal Care Connect.

Almeida Oscar, Zhang Ming, Liu Jyh-Charn. (2007). Dynamic Fall Detection and Pace Measurement in Walking Sticks.

Dobashi Hiroki, Tajima Takuya, Abe Takehiko, Kimura Haruhiko. (2008). Fall Detection System for Bather Using Ultrasound Sensors.

Dai Jiangpeng, Bai Xiaole, Yang Zhimin, Shen Zhaohui and Xuan Dong. (2009). PerFallD: A Pervasive Fall Detection System Using Mobile Phones.

Mars Lan, Ani Nahapetian, Alireza Vahdatpour, Lawrence Au, William Kaiser, Majid Sarrafzadeh. (2009). SmartFall: An Automatic Fall Detection System Based on Subsequence Matching for the SmartCane.

centre suisse d'electronique et de microtechnique [csem]. (2004). Fall Detection Sensor System.

Li Qiang, Stankovic, John A., Mark Hanson, Adam Barth, John Lach, Zhou Gang (2009). Accurate, Fast Fall Detection Using Gyroscopes and Accelerometer-Derived Posture Information.

Miaou Shaou-Gang, Sung Pei-Hsu, Huang Chia-Yuan. (2006). A Customized Human Fall Detection System Using Omni-Camera Images and Personal Information.

Fu, Z., Culurciello, E., Lichtsteiner, P. and Delbruck, T.. (2008). Fall Detection using an Address-Event Temporal Contrast Vision Sensor, in Proceedings of IEEE International Symposium on Circuits and Systems.

Sixsmith, A. and Johnson, N. (2004). A Smart Sensor to Detect the Falls of the Elderly, Pervasive Computing, Vol. 3, Issue 2, pp. 42-47.

APPENDICES

APPENDIX A: Visual Basic 2010 Source Code

MainForm.vb

```
Option Explicit On
Option Strict On

Imports System.IO.Ports
Imports Microsoft.Win32
Imports ZedGraph

Public Class MainForm

    Const ButtonTextOpenPort As String = "Open COM Port"
    Const ButtonTextClosePort As String = "Close COM Port"
    Const ModuleName As String = "Fall Detection Control"

    Const a As String = ControlChars.NewLine
    Dim i As Integer = 0
    Dim i2 As Integer = 1
    Dim acc As Integer = 0
    Dim b As Integer = 0
    Dim c As Integer = 1
    Dim d As Integer = 0
    Dim count As Integer = 0
    Dim fall As Boolean = False
    Dim storage As String = ""
    Dim length2 As Integer = 0
    Dim time As Double

    Friend MyMainForm As MainForm
    Friend MyPortSettingsDialog As PortSettingsDialog
    Friend UserPort1 As ComPorts

    Private Delegate Sub AccessFormMarshalDelegate(ByVal action As String,
ByVal textToAdd As String, _
                                                   ByVal textColor As Color)

    Private AccessFormMarshalDelegate1 As AccessFormMarshalDelegate
    Private colorReceive As Color = Color.Green
    Private colorTransmit As Color = Color.Red
    Private maximumTextBoxLength As Integer
```

```
    Private receiveBuffer As String
    Private savedOpenPortOnStartup As Boolean
    Private userInputIndex As Integer

    ''' <summary>
    ''' Perform functions on the application's form.
    ''' Used to access the form from a different thread.
    ''' See AccessFormMarshal().
    ''' </summary>
    '''
    ''' <param name="action"> a string that names the action to perform on the
form </param>
    ''' <param name="formText"> text that the form displays </param>
    ''' <param name="textColor"> a system color for displaying text </param>

    Private Sub AccessForm(ByVal action As String, ByVal formText As String,
ByVal textColor As Color)

        Select Case action

            ' Select an action to perform on the form.
            ' (Can add more actions as needed.)

            Case "AppendToMonitorTextBox"

                ' Append text to the rtbMonitor textbox using the color for
received data.

                rtbMonitor.SelectionColor = colorReceive
                rtbMonitor.AppendText(formText)

                SavetoRichTextFile(formText)

                While count < formText.Length And fall = False
                    If formText.Substring(count) = "e" Then
                        Beep()
                        MsgBox("Fall has been detected!" + a + "Immediate
action must be taken!", _
                               CType(MessageBoxButtons.OK, MsgBoxStyle), "Fall
Detected!!!")
                        fall = True
                    End If
                    count += 1
                End While

                count = 0
                fall = False

                Convert1(formText.Length, formText)

                'Conversion(formText.Length)

                'Return to the default color.

                rtbMonitor.SelectionColor = colorTransmit

                ' Trim the textbox's contents if needed. (Uncomment codes
below)

                'If rtbMonitor.TextLength > maximumTextBoxLength Then

                'TrimTextBoxContents()
```

```
                'End If

            Case "DisplayStatus"

                ' Add text to the rtbStatus textbox using the specified color.

                DisplayStatus(formText, textColor)

            Case "DisplayCurrentSettings"

                ' Display the current port settings in the
ToolStripStatusLabel.

                DisplayCurrentSettings()

            Case Else

        End Select
    End Sub

    ''' <summary>
    ''' Convert the digits into integers and generate the corresponding graphs
    ''' </summary>

    Private Sub Convert1(ByRef length As Integer, ByRef data As String)

        data = storage & data
        length = length + length2

        While acc <= length And c = 1
            If length - acc = 0 Then
                c = 0
            ElseIf length - acc >= 5 Then
                i = Convert.ToInt16(data.Substring(i2, 4))
                d = d + 1
                If d = 1 Then
                    GenerateGraph(zg1)
                ElseIf d = 2 Then
                    GenerateGraph(zg2)
                ElseIf d = 3 Then
                    GenerateGraph(zg3)
                    d = 0
                End If

                i2 = i2 + 5
                acc = acc + 5
                c = 1

            Else
                acc = acc + 5
                c = 0
                i2 = 0
            End If
        End While

        If length - acc < 0 Then
            length2 = length - acc + 5
            storage = data.Substring(acc - 5, length2)
            i2 = 1
            acc = 0
            c = 1
```

```
        Else
            storage = ""
            length2 = 0
            i2 = 1
            acc = 0
            c = 1
        End If
    End Sub

    ''' <summary>
    ''' Save the data into rich text format.
    ''' </summary>

    Private Sub SavetoRichTextFile(ByVal content As String)

        Dim fileName As String = "C:\" + DateString + " Fall Detection Data
Log.rtf"
        Dim objWriter As New System.IO.StreamWriter(fileName, True)
        objWriter.Write(content)
        objWriter.Close()
    End Sub

    ''' <summary>
    ''' Generate graph.
    ''' </summary>

    Private Sub GenerateGraph(ByVal zgcvalue As ZedGraphControl)
        Dim myPane As GraphPane = zgcvalue.GraphPane
        Dim list As PointPairList = New PointPairList

        list.Add(DateTime.Now.ToOADate, Convert.ToDouble(i))

        If d = 1 Then
            Dim myCurve As LineItem = myPane.AddCurve("", list, Color.Blue,
SymbolType.Circle)
            myCurve.Symbol.Fill = New Fill(Color.Blue)
        ElseIf d = 2 Then
            Dim myCurve As LineItem = myPane.AddCurve("", list, Color.Green,
SymbolType.Circle)
            myCurve.Symbol.Fill = New Fill(Color.Green)
        ElseIf d = 3 Then
            Dim myCurve As LineItem = myPane.AddCurve("", list, Color.Red,
SymbolType.Circle)
            myCurve.Symbol.Fill = New Fill(Color.Red)
        End If

        zgcvalue.AxisChange()
        zgcvalue.Invalidate()
    End Sub

    ''' <summary>
    ''' Enables accessing the form from another thread.
    ''' The parameters match those of AccessForm()
    ''' </summary>
    '''
    ''' <param name="action"> a string that names the action to perform on the
form </param>
    ''' <param name="formText"> text that the form displays </param>
    ''' <param name="textColor"> a system color for displaying text </param>

    Private Sub AccessFormMarshal(ByVal action As String, ByVal formText As
String, ByVal textColor As Color)
```

```
        AccessFormMarshalDelegate1 = New AccessFormMarshalDelegate(AddressOf
AccessForm)
        Dim args() As Object = {action, formText, textColor}

        ' Call AccessForm, passing the parameters in args.

        MyBase.Invoke(AccessFormMarshalDelegate1, args)
    End Sub

    ''' <summary>
    ''' Display the current port parameters on the form.
    ''' </summary>

    Private Sub DisplayCurrentSettings()

        Dim selectedPortState As String = ""

        If ComPorts.comPortExists Then

            If (Not (UserPort1.SelectedPort Is Nothing)) Then

                If UserPort1.SelectedPort.IsOpen Then
                    selectedPortState = "OPEN"
                    btnOpenOrClosePort.Text = ButtonTextClosePort
                Else
                    selectedPortState = "CLOSED"
                    btnOpenOrClosePort.Text = ButtonTextOpenPort
                End If
            End If

            UpdateStatusLabel _
            (CStr(MyPortSettingsDialog.cmbPort.SelectedItem) + "   " + _
            CStr(MyPortSettingsDialog.cmbBitRate.SelectedItem) + _
            "   N 8 1   Handshake: " + _
            MyPortSettingsDialog.cmbHandshaking.SelectedItem.ToString + _
            "   " + selectedPortState)

        Else
            DisplayStatus(ComPorts.noComPortsMessage, Color.Red)
            UpdateStatusLabel("")

        End If
    End Sub

    ''' <summary>
    ''' Provide a central mechanism for displaying exception information.
    ''' Display a message that describes the exception.
    ''' </summary>
    '''
    ''' <param name="moduleName"> the module where the exception
occurred.</param>
    ''' <param name="ex"> the exception </param>

    Private Sub DisplayException(ByVal moduleName As String, ByVal ex As
Exception)

        Dim errorMessage As String

        errorMessage = "Exception: " & ex.Message & _
        " Module: " & moduleName & _
        ". Method: " & ex.TargetSite.Name
```

```
            DisplayStatus(errorMessage, Color.Red)

            ' To display errors in a message box, uncomment this line:
            MessageBox.Show(errorMessage)
        End Sub

        ''' <summary>
        ''' Displays text in a richtextbox.
        ''' </summary>
        '''
        ''' <param name="status"> the text to display.</param>
        ''' <param name="textColor"> the text color. </param>

        Private Sub DisplayStatus(ByVal status As String, ByVal textColor As Color)

            ' Purpose    : Displays text in a richtextbox.

            ' Accepts    : status - the text to display.
            '            : textcolor - the text color.

            rtbStatus.ForeColor = textColor
            rtbStatus.Text = status
        End Sub

        ''' <summary>
        ''' Get user preferences for the COM port and parameters.
        ''' See SetPreferences for more information.
        ''' </summary>

        Private Sub GetPreferences()

            UserPort1.SavedPortName = Settings.Default.ComPort
            UserPort1.SavedBitRate = Settings.Default.BitRate
            UserPort1.SavedHandshake = Settings.Default.Handshaking
            savedOpenPortOnStartup = Settings.Default.OpenComPortOnStartup
        End Sub

        ''' <summary>
        ''' Initialize elements on the main form.
        ''' </summary>

        Private Sub InitializeDisplayElements()

            ' The TrimTextboxContents routine trims a richtextbox with more data
    than this:

            maximumTextBoxLength = 10000
            rtbMonitor.SelectionColor = colorTransmit
        End Sub

        ''' <summary>
        ''' Determine if the textbox's TextChanged event occurred due to new user
    input.
        ''' If yes, get the input and write it to the COM port.
        ''' </summary>

        Private Sub ProcessTextboxInput()

            Dim ar As IAsyncResult
            Dim msg As String
            Dim textLength As Integer
```

```
        Dim userInput As String

        ' Find out if the textbox contains new user input.
        ' If the new data is data received on the COM port or if no COM port
exists, do nothing.

        If ((rtbMonitor.Text.Length > userInputIndex +
UserPort1.ReceivedDataLength) And _
        ComPorts.comPortExists) Then

            ' Retrieve the contents of the textbox.

            userInput = rtbMonitor.Text

            ' Get the length of the new text.

            textLength = userInput.Length - userInputIndex

            ' Extract the unread input.

            userInput = rtbMonitor.Text.Substring(userInputIndex, textLength)

            ' Create a message to pass to the Write operation (optional).
            ' The callback routine can retrieve the message when the write
completes.

            msg = DateTime.Now.ToString

            ' Send the input to the COM port.
            ' Use a different thread so the main application doesn't have to
wait
            ' for the write operation to complete.

            ar = UserPort1.WriteToComPortDelegate1.BeginInvoke(userInput, New
AsyncCallback(AddressOf UserPort1.WriteCompleted), msg)

            ' To use the same thread for writes to the port,
            ' comment out the statement above and uncomment the statement
below.
            'UserPort1.WriteToComPort(userInput)

            AccessForm("UpdateStatusLabel", "", Color.Black)

        Else
            ' Received bytes displayed in the text box are ignored,
            ' but we need to reset the value that indicates
            ' the number of received but not processed bytes.

            UserPort1.ReceivedDataLength = 0

        End If

        'Uncomment codes below to trim text box contents

        'If rtbMonitor.TextLength > maximumTextBoxLength Then
        'TrimTextBoxContents()
        '   End If

        ' Update the value that indicates the last character processed.

        userInputIndex = rtbMonitor.Text.Length
    End Sub
```

```
    ''' <summary>
    ''' Save user preferences for the COM port and parameters.
    ''' </summary>

    Private Sub SavePreferences()

        ' To define additional settings, in the Visual Studio IDE go to
        ' Solution Explorer > right click on project name > Properties >
Settings.

        If (MyPortSettingsDialog.cmbPort.SelectedIndex > -1) Then

            ' The system has at least one COM port.

            Settings.Default.ComPort =
MyPortSettingsDialog.cmbPort.SelectedItem.ToString
            Settings.Default.BitRate =
CInt(MyPortSettingsDialog.cmbBitRate.SelectedItem)
            Settings.Default.Handshaking =
DirectCast(MyPortSettingsDialog.cmbHandshaking.SelectedItem, Handshake)
            Settings.Default.OpenComPortOnStartup =
MyPortSettingsDialog.chkOpenComPortOnStartup.Checked
            Settings.Default.Save()
        End If
    End Sub

    ''' <summary>
    ''' Use stored preferences or defaults to set the initial port parameters.
    ''' </summary>

    Private Sub SetInitialPortParameters()

        ' Get preferences or default values.

        GetPreferences()

        If ComPorts.comPortExists Then

            ' Select a COM port and bit rate using stored preferences if
available.

            UsePreferencesToSelectParameters()

            ' Save the selected indexes of the combo boxes.

            MyPortSettingsDialog.SavePortParameters()

        Else

            ' No COM ports have been detected. Watch for one to be attached.

            tmrLookForPortChanges.Start()
            DisplayStatus(ComPorts.noComPortsMessage, Color.Red)

        End If

        UserPort1.ParameterChanged = False
    End Sub

    ''' <summary>
    ''' Saves the passed port parameters.
```

```
    ''' Called when the user clicks OK on PortSettingsDialog.
    ''' </summary>

    Private Sub SetPortParameters(ByVal userPort As String, ByVal userBitRate
As Integer, _
                                  ByVal userHandshake As Handshake)

        Try

            ' Don't do anything if the system has no COM ports.

            If ComPorts.comPortExists Then

                If MyPortSettingsDialog.ParameterChanged Then

                    ' One or more port parameters has changed.

                    If (String.Compare(MyPortSettingsDialog.oldPortName,
CStr(userPort), True) <> 0) Then

                        ' The port has changed.
                        ' Close the previously selected port.

                        UserPort1.PreviousPort = UserPort1.SelectedPort
                        UserPort1.CloseComPort(UserPort1.SelectedPort)

                        ' Set SelectedPort to the current port.

                        UserPort1.SelectedPort.PortName = userPort
                        UserPort1.PortChanged = True

                    End If

                    ' Set other port parameters.

                    UserPort1.SelectedPort.BaudRate = userBitRate
                    UserPort1.SelectedPort.Handshake = userHandshake

                    MyPortSettingsDialog.SavePortParameters()

                    UserPort1.ParameterChanged = True

                Else
                    UserPort1.ParameterChanged = False

                End If
            End If

        Catch ex As InvalidOperationException

            UserPort1.ParameterChanged = True
            DisplayException(ModuleName, ex)

        Catch ex As UnauthorizedAccessException

            UserPort1.ParameterChanged = True
            DisplayException(ModuleName, ex)

            ' This exception can occur if the port was removed.
            ' If the port was open, close it.

            UserPort1.CloseComPort(UserPort1.SelectedPort)
```

```
        Catch ex As System.IO.IOException

            UserPort1.ParameterChanged = True
            DisplayException(ModuleName, ex)

        End Try
    End Sub

    ''' <summary>
    ''' Trim a richtextbox by removing the oldest contents.
    ''' </summary>
    '''
    ''' <remarks >
    ''' To trim the box while retaining any formatting applied to the retained
contents,
    ''' create a temporary richtextbox, copy the contents to be preserved to
the
    ''' temporary richtextbox,and copy the temporary richtextbox back to the
original richtextbox.
    ''' </remarks>

    Private Sub TrimTextBoxContents()

        Dim rtbTemp As New RichTextBox
        Dim textboxTrimSize As Integer

        ' When the contents are too large, remove half.

        textboxTrimSize = maximumTextBoxLength \ 2

        rtbMonitor.Select(rtbMonitor.TextLength - textboxTrimSize + 1,
textboxTrimSize)
        rtbTemp.Rtf = rtbMonitor.SelectedRtf
        rtbMonitor.Clear()
        rtbMonitor.Rtf = rtbTemp.Rtf
        rtbTemp = Nothing
        rtbMonitor.SelectionStart = rtbMonitor.TextLength
    End Sub

    ''' <summary>
    ''' Set the text in the ToolStripStatusLabel.
    ''' </summary>
    '''
    ''' <param name="status"> the text to display </param>

    Private Sub UpdateStatusLabel(ByVal status As String)

        ToolStripStatusLabel1.Text = status
        StatusStrip1.Update()
    End Sub

    ''' <summary>
    ''' Set the user preferences or default values in the combo boxes and
ports array
    ''' using stored preferences or default values.
    ''' </summary>

    Private Sub UsePreferencesToSelectParameters()

        MyPortSettingsDialog.SelectComPort(UserPort1.SavedPortName)
        MyPortSettingsDialog.SelectBitRate(UserPort1.SavedBitRate)
```

```
        UserPort1.SelectedPort.BaudRate = UserPort1.SavedBitRate
        MyPortSettingsDialog.SelectHandshaking(UserPort1.SavedHandshake)
        UserPort1.SelectedPort.Handshake = UserPort1.SavedHandshake
        MyPortSettingsDialog.chkOpenComPortOnStartup.Checked =
savedOpenPortOnStartup
    End Sub

    ''' <summary>
    ''' Depending on the text displayed on the button, open or close the
selected port
    ''' and change the button text to the opposite action.
    ''' </summary>

    Private Sub btnOpenOrClosePort_Click(ByVal sender As Object, ByVal e As
System.EventArgs) _
        Handles btnOpenOrClosePort.Click

        If (btnOpenOrClosePort.Text Is ButtonTextOpenPort) Then
            UserPort1.OpenComPort()
            If UserPort1.SelectedPort.IsOpen Then
                rtbMonitor.Enabled = True
                btnOpenOrClosePort.Text = ButtonTextClosePort
            End If

        ElseIf (btnOpenOrClosePort.Text Is ButtonTextClosePort) Then
            UserPort1.CloseComPort(UserPort1.SelectedPort)
            If Not UserPort1.SelectedPort.IsOpen Then
                rtbMonitor.Enabled = False
                btnOpenOrClosePort.Text = ButtonTextOpenPort
            End If
        End If
    End Sub

    ''' <summary>
    ''' Create an instance of the ComPorts class.
    ''' Initialize port settings and other parameters.
    ''' specify behavior on events.
    ''' </summary>

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        Show()

        ' Create an instance of the ComPorts class for accessing a specific
port.

        UserPort1 = New ComPorts

        MyPortSettingsDialog = New PortSettingsDialog

        tmrLookForPortChanges.Interval = 1000
        tmrLookForPortChanges.Stop()

        InitializeDisplayElements()

        SetInitialPortParameters()

        If ComPorts.comPortExists Then
            UserPort1.SelectedPort.PortName =
ComPorts.myPortNames(MyPortSettingsDialog.cmbPort.SelectedIndex)
```

```
            ' A check box enables requesting to open the selected COM port on
start up.
            ' Otherwise the application opens the port when the user clicks
the Open Port
            ' button or types text to send.

            If MyPortSettingsDialog.chkOpenComPortOnStartup.Checked Then

                UserPort1.PortOpen = UserPort1.OpenComPort()
                rtbMonitor.Enabled = True
                AccessForm("DisplayCurrentSettings", "", Color.Black)
                AccessForm("DisplayStatus", "", Color.Black)

            Else
                DisplayCurrentSettings()
            End If
        End If

        ' Specify the routines that execute on events in other modules.
        ' The routines can receive data from other modules.

        AddHandler ComPorts.UserInterfaceData, AddressOf AccessFormMarshal
        AddHandler PortSettingsDialog.UserInterfaceData, AddressOf
AccessFormMarshal
        AddHandler PortSettingsDialog.UserInterfacePortSettings, AddressOf
SetPortParameters

        'Initialize graph

        CreateGraph(zg1, zg2, zg3)
    End Sub

    ''' <summary>
    ''' Close the port if needed and save preferences.
    ''' </summary>

    Private Sub MainForm_FormClosing(ByVal sender As Object, ByVal e As  _
                                     System.Windows.Forms.FormClosingEventArgs)
Handles Me.FormClosing

        UserPort1.CloseComPort(UserPort1.SelectedPort)
        SavePreferences()
    End Sub

    ''' <summary>
    ''' Do whatever is needed with new characters in the textbox.
    ''' </summary>

    Private Sub rtbMonitor_TextChanged(ByVal sender As System.Object, ByVal e
As System.EventArgs) _
        Handles rtbMonitor.TextChanged

        ProcessTextboxInput()
    End Sub

    ''' <summary>
    ''' Look for ports. If at least one is found, stop the timer and
    ''' select the saved port if possible or the first port.
    ''' This timer is enabled only when no COM ports are present.
    ''' </summary>
```

```
    Private Sub tmrLookForPortChanges_Tick(ByVal sender As Object, ByVal e As
System.EventArgs) _
        Handles tmrLookForPortChanges.Tick

        ComPorts.FindComPorts()

        If ComPorts.comPortExists Then

            tmrLookForPortChanges.Stop()
            DisplayStatus("COM port(s) found.", Color.Black)

            MyPortSettingsDialog.DisplayComPorts()
            MyPortSettingsDialog.SelectComPort(UserPort1.SavedPortName)
            MyPortSettingsDialog.SelectBitRate(UserPort1.SavedBitRate)
            MyPortSettingsDialog.SelectHandshaking(UserPort1.SavedHandshake)

            ' Set selectedPort.

            SetPortParameters(UserPort1.SavedPortName,
CInt(UserPort1.SavedBitRate), _
                              DirectCast(UserPort1.SavedHandshake, Handshake))

            DisplayCurrentSettings()
            UserPort1.ParameterChanged = True
        End If
    End Sub

    ''' <summary>
    ''' Clear text in the rtb.
    ''' </summary>

    Private Sub BtnClearAll_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) _
        Handles BtnClearAll.Click

        rtbMonitor.Clear()
        rtbStatus.Clear()
        rtbMonitor.SelectionColor = colorTransmit
        i2 = 1
        i = 0
        acc = 0
        b = 0
        c = 1
        d = 0
        storage = ""
        length2 = 0
    End Sub

    ''' <summary>
    ''' Look for COM ports and display them in the combo box.
    ''' </summary>

    Private Sub PortSettingToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) _
        Handles PortSettingToolStripMenuItem.Click

        ComPorts.FindComPorts()

        MyPortSettingsDialog.DisplayComPorts()
        MyPortSettingsDialog.SelectComPort(UserPort1.SelectedPort.PortName)
        MyPortSettingsDialog.SelectBitRate(UserPort1.SelectedPort.BaudRate)
```

```
MyPortSettingsDialog.SelectHandshaking(UserPort1.SelectedPort.Handshake)

        UserPort1.ParameterChanged = False

        ' Display the combo boxes for setting port parameters.

        MyPortSettingsDialog.ShowDialog()
    End Sub

    Private Sub ExitToolStripMenuItem1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) _
        Handles ExitToolStripMenuItem1.Click

        UserPort1.CloseComPort(UserPort1.SelectedPort)
        SavePreferences()
        End
    End Sub

    ''' <summary>
    ''' Initialize Graph.
    ''' </summary>

    Private Sub CreateGraph(ByVal zgc1 As ZedGraphControl, _
                            ByVal zgc2 As ZedGraphControl, ByVal zgc3 As
ZedGraphControl)
        Dim myPane1 As GraphPane = zgc1.GraphPane
        Dim myPane2 As GraphPane = zgc2.GraphPane
        Dim myPane3 As GraphPane = zgc3.GraphPane

        ' Set the titles and axis labels
        myPane1.Title.Text = "Acceleration x Versus Time"
        myPane2.Title.Text = "Acceleration y Versus Time"
        myPane1.XAxis.Title.Text = "Time"
        myPane1.YAxis.Title.Text = "Acceleration"
        myPane1.XAxis.Type = AxisType.Date
        myPane2.XAxis.Title.Text = "Time"
        myPane2.YAxis.Title.Text = "Acceleration"
        myPane2.XAxis.Type = AxisType.Date
        myPane3.Title.Text = "Acceleration z Versus Time"
        myPane3.XAxis.Title.Text = "Time"
        myPane3.YAxis.Title.Text = "Acceleration"
        myPane3.XAxis.Type = AxisType.Date
    End Sub
End Class
```

PortSettingsDialog.vb

```
Option Explicit On
Option Strict On

Imports System.IO.Ports

''' <summary>
''' Provides a dialog box for viewing and selecting COM ports and parameters.
''' </summary>

Public Class PortSettingsDialog
```

```
    Private bitRates(10) As Integer
    Private oldBitRateIndex As Integer
    Private oldHandshakeIndex As Integer
    Friend oldPortName As String
    Private settingsInitialized As Boolean

    ' These events enable other modules to detect events and receive data.

    Friend Shared Event UserInterfacePortSettings(ByVal selectedPort As String,
ByVal selectedBitRate As Integer, ByVal selectedHandshake As Handshake)
    Friend Shared Event UserInterfaceData(ByVal action As String, ByVal
formText As String, ByVal textColor As Color)

    ''' <summary>
    ''' Display available COM ports in a combo box.
    ''' Assumes ComPorts.FindComPorts has been run to fill the myPorts array.
    ''' </summary>

    Friend Sub DisplayComPorts()

        ' Clear the combo box and repopulate (in case ports have been added or
removed).

        cmbPort.DataSource = ComPorts.myPortNames

    End Sub

    ''' <summary>
    ''' Set initial port parameters.
    ''' </summary>

    Friend Sub InitializePortSettings()

        If Not settingsInitialized Then

            'Bit rates to select from.

            bitRates(0) = 300
            bitRates(1) = 600
            bitRates(2) = 1200
            bitRates(3) = 2400
            bitRates(4) = 9600
            bitRates(5) = 14400
            bitRates(6) = 19200
            bitRates(7) = 38400
            bitRates(8) = 57600
            bitRates(9) = 115200
            bitRates(10) = 128000

            'Place the bit rates and handshaking options in the combo boxes.

            cmbBitRate.DataSource = bitRates
            cmbBitRate.DropDownStyle = ComboBoxStyle.DropDownList

            ' Handshaking options.

            cmbHandshaking.Items.Add(Handshake.None)
            cmbHandshaking.Items.Add(Handshake.XOnXOff)
            cmbHandshaking.Items.Add(Handshake.RequestToSend)
            cmbHandshaking.Items.Add(Handshake.RequestToSendXOnXOff)

            cmbHandshaking.DropDownStyle = ComboBoxStyle.DropDownList
```

```
            'Find and display available COM ports.

            ComPorts.FindComPorts()

            cmbPort.DataSource = ComPorts.myPortNames

            cmbPort.DropDownStyle = ComboBoxStyle.DropDownList

            settingsInitialized = True
        End If
    End Sub

    ''' <summary>
    ''' Compares stored parameters with the current parameters.
    ''' </summary>
    '''
    ''' <returns>
    ''' True if any parameter has changed.
    ''' </returns>

    Friend Function ParameterChanged() As Boolean

        Return (oldBitRateIndex <> cmbBitRate.SelectedIndex) Or _
                  (oldHandshakeIndex <> cmbHandshaking.SelectedIndex) Or _
                  ((String.Compare(oldPortName, CStr(cmbPort.SelectedItem),
True) <> 0))
    End Function

    ''' <summary>
    ''' Save the current port parameters.
    ''' Enables learning if a parameter has changed.
    ''' </summary>

    Friend Sub SavePortParameters()

        oldBitRateIndex = cmbBitRate.SelectedIndex
        oldHandshakeIndex = cmbHandshaking.SelectedIndex
        oldPortName = CStr(cmbPort.SelectedItem)

    End Sub

    ''' <summary>
    ''' Select a bit rate in the combo box.
    ''' Does not set the bit rate for a COM port.
    ''' </summary>
    '''
    ''' <param name="bitRate"> The requested bit rate </param>

    Friend Sub SelectBitRate(ByVal bitRate As Integer)

        cmbBitRate.SelectedItem = bitRate

    End Sub

    ''' <summary>
    ''' Select a COM port in the combo box.
    ''' Does not set the selectedPort variable.
    ''' </summary>
    '''
    ''' <param name="comPortName"> A COM port name </param>
    '''
```

```
    ''' <returns > The index of the selected port in the combo box </returns>

    Friend Function SelectComPort(ByVal comPortName As String) As Integer

        ' If comPortName doesn't exist in the combo box, SelectedItem remains
the same.

        cmbPort.SelectedItem = comPortName

        If (cmbPort.SelectedIndex > -1) Then

            ' At least one COM port exists.

            If (Not (String.Compare(cmbPort.SelectedItem.ToString, comPortName,
True) = 0)) Then

                ' The requested port isn't available. Select the first port.

                cmbPort.SelectedIndex = 0

            End If
        Else

            ' No COM ports exist.

            RaiseEvent UserInterfaceData("DisplayStatus",
ComPorts.noComPortsMessage, Color.Red)
            ComPorts.comPortExists = False
        End If

        Return cmbPort.SelectedIndex
    End Function

    ''' <summary>
    ''' Sets handshaking in the combo box.
    ''' Does not set handshaking for a COM port.
    ''' </summary>
    '''
    ''' <param name="requestedHandshake"> the requested handshaking as a
System.IO.Ports.Handshake value. </param>

    Friend Sub SelectHandshaking(ByVal requestedHandshake As Handshake)

        cmbHandshaking.SelectedItem = requestedHandshake

    End Sub

    ''' <summary>
    ''' Initialize port settings.
    ''' InitializeComponent is required by the Windows Form Designer.
    ''' </summary>
    Sub New()

        InitializeComponent()

        InitializePortSettings()

    End Sub

    ''' <summary>
    ''' The port parameters may have changed.
    ''' Make the parameters available to other modules.
```

```
    ''' </summary>

    Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnOK.Click

        Dim statusMessage As String

        ' Set the port parameters.

        If ComPorts.comPortExists Then
            RaiseEvent  UserInterfacePortSettings(cmbPort.SelectedItem.ToString,
CInt(cmbBitRate.SelectedItem), DirectCast(cmbHandshaking.SelectedItem,
Handshake))
        End If

        RaiseEvent UserInterfaceData("DisplayCurrentSettings", "", Color.Black)

        If cmbPort.SelectedIndex = -1 Then
            statusMessage = ComPorts.noComPortsMessage
        Else
            statusMessage = ""
        End If

        RaiseEvent UserInterfaceData("DisplayStatus", statusMessage,
Color.Black)

    End Sub

    ''' <summary>
    ''' Configure components in the dialog box.
    ''' </summary>

    Friend Sub PortSettingsDialog_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load

        btnOK.DialogResult = Windows.Forms.DialogResult.OK
        Me.AcceptButton = btnOK
        btnOK.Focus()

    End Sub

    ''' <summary>
    ''' Don't save any changes to the form.
    ''' </summary>

    Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCancel.Click
        Me.Hide()
    End Sub
End Class
```

ComPorts.vb

```
option explicit on
Option Strict On

Imports System.IO.Ports
Imports System.Runtime.Remoting.Messaging

''' <summary>
```

```
''' Routines for finding and accessing COM ports.
''' </summary>

Public Class ComPorts

    Const ModuleName As String = "ComPorts"

    ' Shared members - do not belong to a specific instance of the class.

    Friend Shared comPortExists As Boolean
    Friend Shared myPortNames() As String
    Friend Shared noComPortsMessage As String = "No COM ports found. Please
attach a COM-port device."

    Friend Shared Event UserInterfaceData _
           (ByVal action As String, ByVal formText As String, ByVal textColor
As Color)

       ' Non-shared members - belong to a specific instance of the class.

    Friend Delegate Function WriteToComPortDelegate(ByVal textToWrite As
String) As Boolean

    Friend WriteToComPortDelegate1 As New WriteToComPortDelegate(AddressOf
WriteToComPort)

    Private SerialDataReceivedEventHandler1 _
      As New SerialDataReceivedEventHandler(AddressOf DataReceived)
    Private SerialErrorReceivedEventHandler1 _
          As New SerialErrorReceivedEventHandler(AddressOf ErrorReceived)

    ' Local variables available as Properties.

    Private m_ParameterChanged As Boolean
    Private m_PortChanged As Boolean
    Private m_PortOpen As Boolean
    Private m_PreviousPort As New SerialPort
    Private m_ReceivedDataLength As Integer
    Private m_SavedBitRate As Integer = 9600
    Private m_SavedHandshake As Handshake = Handshake.None
    Private m_SavedPortName As String = ""
    Private m_SelectedPort As New SerialPort

    Friend Property ParameterChanged() As Boolean
        Get
            Return m_ParameterChanged
        End Get
        Set(ByVal value As Boolean)
            m_ParameterChanged = value
        End Set
    End Property

    Friend Property PortChanged() As Boolean
        Get
            Return m_PortChanged
        End Get
        Set(ByVal value As Boolean)
            m_PortChanged = value
        End Set
    End Property

    Friend Property PortOpen() As Boolean
```

```
    Get
        Return m_PortOpen
    End Get
    Set(ByVal value As Boolean)
        m_PortOpen = value
    End Set
End Property

Friend Property PreviousPort() As SerialPort
    Get
        Return m_PreviousPort
    End Get
    Set(ByVal value As SerialPort)
        m_PreviousPort = value
    End Set
End Property

Friend Property ReceivedDataLength() As Integer
    Get
        Return m_ReceivedDataLength
    End Get
    Set(ByVal value As Integer)
        m_ReceivedDataLength = value
    End Set
End Property

Friend Property SavedBitRate() As Integer
    Get
        Return m_SavedBitRate
    End Get
    Set(ByVal value As Integer)
        m_SavedBitRate = value
    End Set
End Property

Friend Property SavedHandshake() As Handshake
    Get
        Return m_SavedHandshake
    End Get
    Set(ByVal value As Handshake)
        m_SavedHandshake = value
    End Set
End Property

Friend Property SavedPortName() As String
    Get
        Return m_SavedPortName
    End Get
    Set(ByVal value As String)
        m_SavedPortName = value
    End Set
End Property

Friend Property SelectedPort() As SerialPort
    Get
        Return m_SelectedPort
    End Get
    Set(ByVal value As SerialPort)
        m_SelectedPort = value
    End Set
End Property
```

```
    ''' <summary>
    ''' If the COM port is open, close it.
    ''' </summary>
    '''
    ''' <param name="portToClose"> the SerialPort object to close </param>

    Friend Sub CloseComPort(ByVal portToClose As SerialPort)

        Try
            RaiseEvent UserInterfaceData("DisplayStatus", "", Color.Black)

            If (Not IsNothing(portToClose)) Then

                If portToClose.IsOpen Then

                    portToClose.Close()
                    RaiseEvent UserInterfaceData("DisplayCurrentSettings", "",
Color.Black)

                End If
            End If

        Catch ex As InvalidOperationException

            parameterChanged = True
            portChanged = True
            DisplayException(ModuleName, ex)

        Catch ex As UnauthorizedAccessException

            parameterChanged = True
            portChanged = True
            DisplayException(ModuleName, ex)

        Catch ex As System.IO.IOException

            parameterChanged = True
            portChanged = True
            DisplayException(ModuleName, ex)
        End Try

    End Sub

    ''' <summary>
    ''' Called when data is received on the COM port.
    ''' Reads and displays the data.
    ''' See FindPorts for the AddHandler statement for this routine.
    ''' </summary>

    Friend Sub DataReceived(ByVal sender As Object, ByVal e As
SerialDataReceivedEventArgs)

        Dim newReceivedData As String

        Try
            ' Get data from the COM port.

            newReceivedData = selectedPort.ReadExisting

            ' Save the number of characters received.

            receivedDataLength += newReceivedData.Length
```

```
            RaiseEvent UserInterfaceData("AppendToMonitorTextBox",
newReceivedData, Color.Black)

        Catch ex As Exception
            DisplayException(ModuleName, ex)
        End Try
    End Sub

    ''' <summary>
    ''' Provide a central mechanism for displaying exception information.
    ''' Display a message that describes the exception.
    ''' </summary>
    '''
    ''' <param name="ex"> The exception </param>
    ''' <param name="moduleName" > the module where the exception was raised.
</param>

    Private Sub DisplayException(ByVal moduleName As String, ByVal ex As
Exception)

        Dim errorMessage As String

        errorMessage = "Exception: " & ex.Message & _
        " Module: " & moduleName & _
        ". Method: " & ex.TargetSite.Name

        RaiseEvent UserInterfaceData("DisplayStatus", errorMessage, Color.Red)

        ' To display errors in a message box, uncomment this line:
        ' MessageBox.Show(errorMessage)
    End Sub

    ''' <summary>
    ''' Respond to error events.
    ''' </summary>

    Private Sub ErrorReceived(ByVal sender As Object, ByVal e As
SerialErrorReceivedEventArgs)

        Dim SerialErrorReceived1 As SerialError

        SerialErrorReceived1 = e.EventType

        Select Case SerialErrorReceived1

            Case SerialError.Frame
                Console.WriteLine("Framing error.")

            Case SerialError.Overrun
                Console.WriteLine("Character buffer overrun.")

            Case SerialError.RXOver
                Console.WriteLine("Input buffer overflow.")

            Case SerialError.RXParity
                Console.WriteLine("Parity error.")

            Case SerialError.TXFull
                Console.WriteLine("Output buffer full.")
        End Select
    End Sub
```

```
    ''' <summary>
    ''' Find the PC's COM ports and store parameters for each port.
    ''' Use saved parameters if possible, otherwise use default values.
    ''' </summary>
    '''
    ''' <remarks>
    ''' The ports can change if a USB/COM-port converter is attached or
removed,
    ''' so this routine may need to run multiple times.
    ''' </remarks>

    Friend Shared Sub FindComPorts()

        ' Place the names of all COM ports in an array and sort.

        myPortNames = SerialPort.GetPortNames

        ' Is there at least one COM port?

        If myPortNames.Length > 0 Then

            comPortExists = True
            Array.Sort(myPortNames)

        Else
            ' No COM ports found.

            comPortExists = False
        End If

    End Sub

    ''' <summary>
    ''' Open the SerialPort object selectedPort.
    ''' If open, close the SerialPort object previousPort.
    ''' </summary>

    Friend Function OpenComPort() As Boolean

        Dim success As Boolean = False

        Try
            If comPortExists Then

                ' The system has at least one COM port.
                ' If the previously selected port is still open, close it.

                If PreviousPort.IsOpen Then
                    CloseComPort(PreviousPort)
                End If

                If (Not (SelectedPort.IsOpen) Or PortChanged) Then

                    SelectedPort.Open()

                    If SelectedPort.IsOpen Then

                        ' The port is open. Set additional parameters.
                        ' Timeouts are in milliseconds.

                        SelectedPort.ReadTimeout = 10000
```

```
                        SelectedPort.WriteTimeout = 10000

                        ' Specify the routine that runs when a DataReceived
event occurs.

                        AddHandler SelectedPort.DataReceived,
SerialDataReceivedEventHandler1
                        AddHandler SelectedPort.ErrorReceived,
SerialErrorReceivedEventHandler1

                        ' Send data to other modules.

                        RaiseEvent UserInterfaceData("DisplayCurrentSettings",
"", Color.Black)
                        RaiseEvent UserInterfaceData("DisplayStatus", "",
Color.Black)

                        success = True

                        ' The port is open with the current parameters.

                        PortChanged = False

                    End If
                End If

            End If

        Catch ex As InvalidOperationException

            parameterChanged = True
            portChanged = True
            DisplayException(ModuleName, ex)

        Catch ex As UnauthorizedAccessException

            parameterChanged = True
            portChanged = True
            DisplayException(ModuleName, ex)

        Catch ex As System.IO.IOException

            parameterChanged = True
            portChanged = True
            DisplayException(ModuleName, ex)
        End Try

        Return success

    End Function

    ''' <summary>
    '''  Executes when WriteToComPortDelegate1 completes.
    ''' </summary>
    ''' <param name="ar"> the value returned by the delegate's BeginInvoke
method </param>

    Friend Sub WriteCompleted(ByVal ar As IAsyncResult)

        Dim deleg As WriteToComPortDelegate
        Dim msg As String
        Dim success As Boolean
```

```
        ' Extract the value returned by BeginInvoke (optional).

        msg = DirectCast(ar.AsyncState, String)

        ' Get the value returned by the delegate.

        deleg = DirectCast(DirectCast(ar, AsyncResult).AsyncDelegate,
WriteToComPortDelegate)

        success = deleg.EndInvoke(ar)

        If success Then
            RaiseEvent UserInterfaceData("UpdateStatusLabel", "", Color.Black)
        End If

        ' Add any actions that need to be performed after a write to the COM
port completes.
        ' This example displays the value passed to the BeginInvoke method
        ' and the value returned by EndInvoke.

        Console.WriteLine("Write operation began: " & msg)
        Console.WriteLine("Write operation succeeded: " & success)

    End Sub

    ''' <summary>
    ''' Write a string to the SerialPort object selectedPort.
    ''' </summary>
    '''
    ''' <param name="textToWrite"> A string to write </param>

    Friend Function WriteToComPort(ByVal textToWrite As String) As Boolean

        Dim success As Boolean

        Try
            ' Open the COM port if necessary.

            If (Not (selectedPort Is Nothing)) Then
                If ((Not selectedPort.IsOpen) Or portChanged) Then

                    ' Close the port if needed and open the selected port.

                    portOpen = OpenComPort()

                End If
            End If

            If selectedPort.IsOpen Then
                selectedPort.Write(textToWrite)
                success = True
            End If

        Catch ex As TimeoutException
            DisplayException(ModuleName, ex)

        Catch ex As InvalidOperationException
            DisplayException(ModuleName, ex)
            parameterChanged = True
            RaiseEvent UserInterfaceData("DisplayCurrentSettings", "",
Color.Black)
```

```
        Catch ex As UnauthorizedAccessException
            DisplayException(ModuleName, ex)

            ' This exception can occur if the port was removed.
            ' If the port was open, close it.

            CloseComPort(selectedPort)
            parameterChanged = True
            RaiseEvent UserInterfaceData("DisplayCurrentSettings", "",
Color.Black)

        End Try

        Return success

    End Function

End Class
```

APPENDIX B: MPLAB Source Code

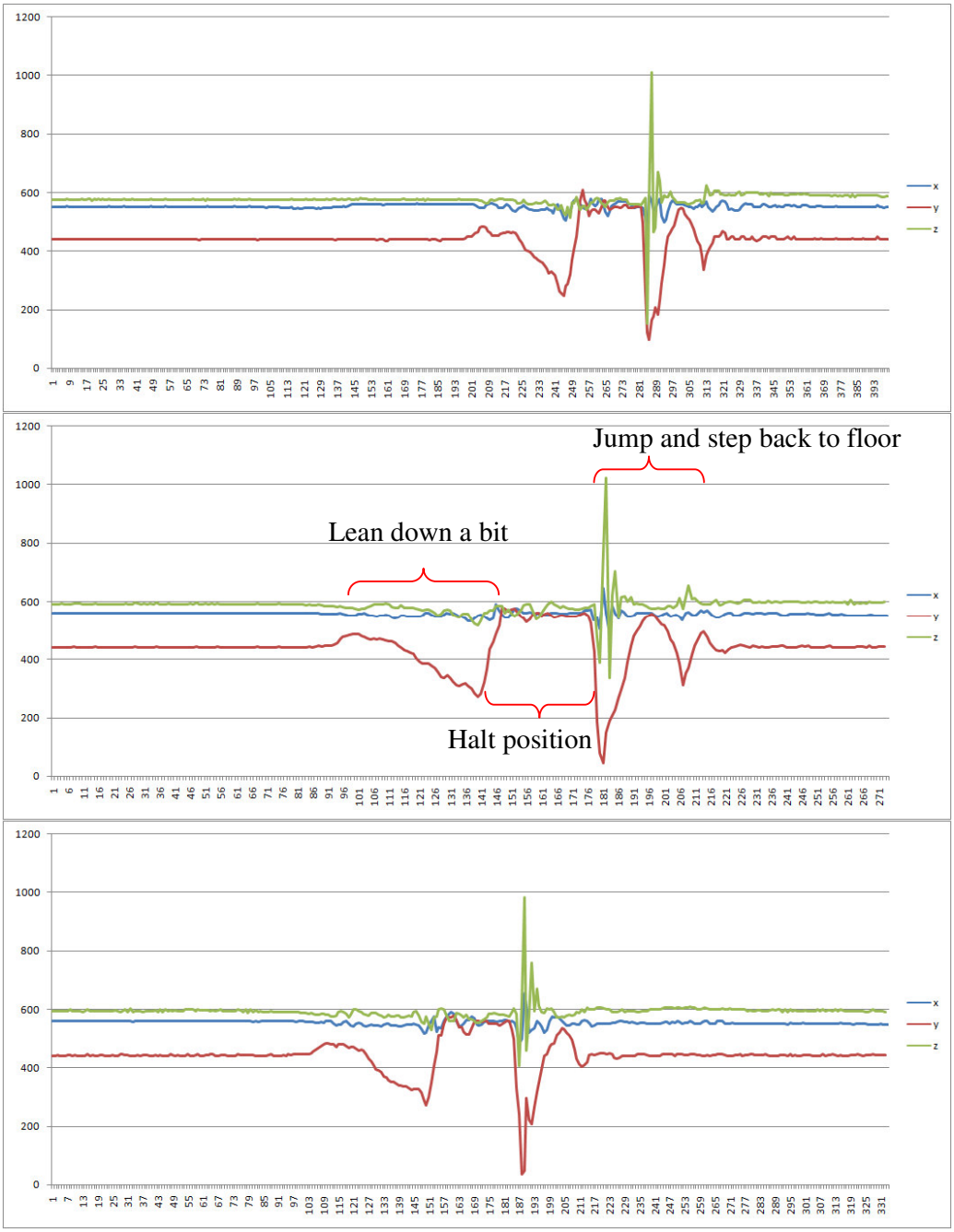
```
#include <pic.h>
#include <math.h>
#include <stdio.h>
#include "delay.h"
#include "delay.c"

__CONFIG(0x3F32);

void initPIC();
void convert(int num);
void GetSensor(int *SData);
void display(unsigned int c);
unsigned char receive(void);

int SData[3];
unsigned char a;

void main(void)
{
      unsigned int one;
      unsigned int ten;
      unsigned int hundreds;
      unsigned int thousands;
      unsigned int Data;
      unsigned int Threshold=1112;
      unsigned long x;
      unsigned long x2;
      unsigned long y;
      unsigned long y2;
      unsigned long z;
      unsigned long z2;
      unsigned int calculated_acceleration;

//    int i;

      initPIC();

      while(1)                      // Wait for 'ok' to be entered
by user
      {
            a = receive();

            if (a == 'o')
            {
                  a = receive();
                  if (a == 'k') break;
            }
      }
```

```
	display(0x0a);		//Enter new line

	// Text will display on Hyperterminal after 'ok' is entered

	while(1)
	{
		GetSensor(SData);

		Data=SData[0]/10;
		one=SData[0]%10;
		ten=Data%10;
		Data=Data/10;
		hundreds=Data%10;
		Data=Data/10;
		thousands=Data%10;
		display(0x30+thousands);
		display(0x30+hundreds);
		display(0x30+ten);
		display(0x30+one);
		display(0x20);			// Space
		Data=SData[1]/10;
		one=SData[1]%10;
		ten=Data%10;
		Data=Data/10;
		hundreds=Data%10;
		Data=Data/10;
		thousands=Data%10;
		display(0x30+thousands);
		display(0x30+hundreds);
		display(0x30+ten);
		display(0x30+one);
		display(0x20);			// Space
		Data=SData[2]/10;
		one=SData[2]%10;
		ten=Data%10;
		Data=Data/10;
		hundreds=Data%10;
		Data=Data/10;
		thousands=Data%10;
		display(0x30+thousands);
		display(0x30+hundreds);
		display(0x30+ten);
		display(0x30+one);
		display(0x0a);			//Go to new line
		//display(0x0d);

		x=SData[0];
		x2=x*x;
		y=SData[1];
		y2=y*y;
		z=SData[2];
		z2=z*z;
		calculated_acceleration=(unsigned
int)round(sqrt((x2+y2+z2)));
		if(calculated_acceleration >= Threshold)
		{
			display(0x65);		//"e"
			while(1)
			{
				RD1=1;
```

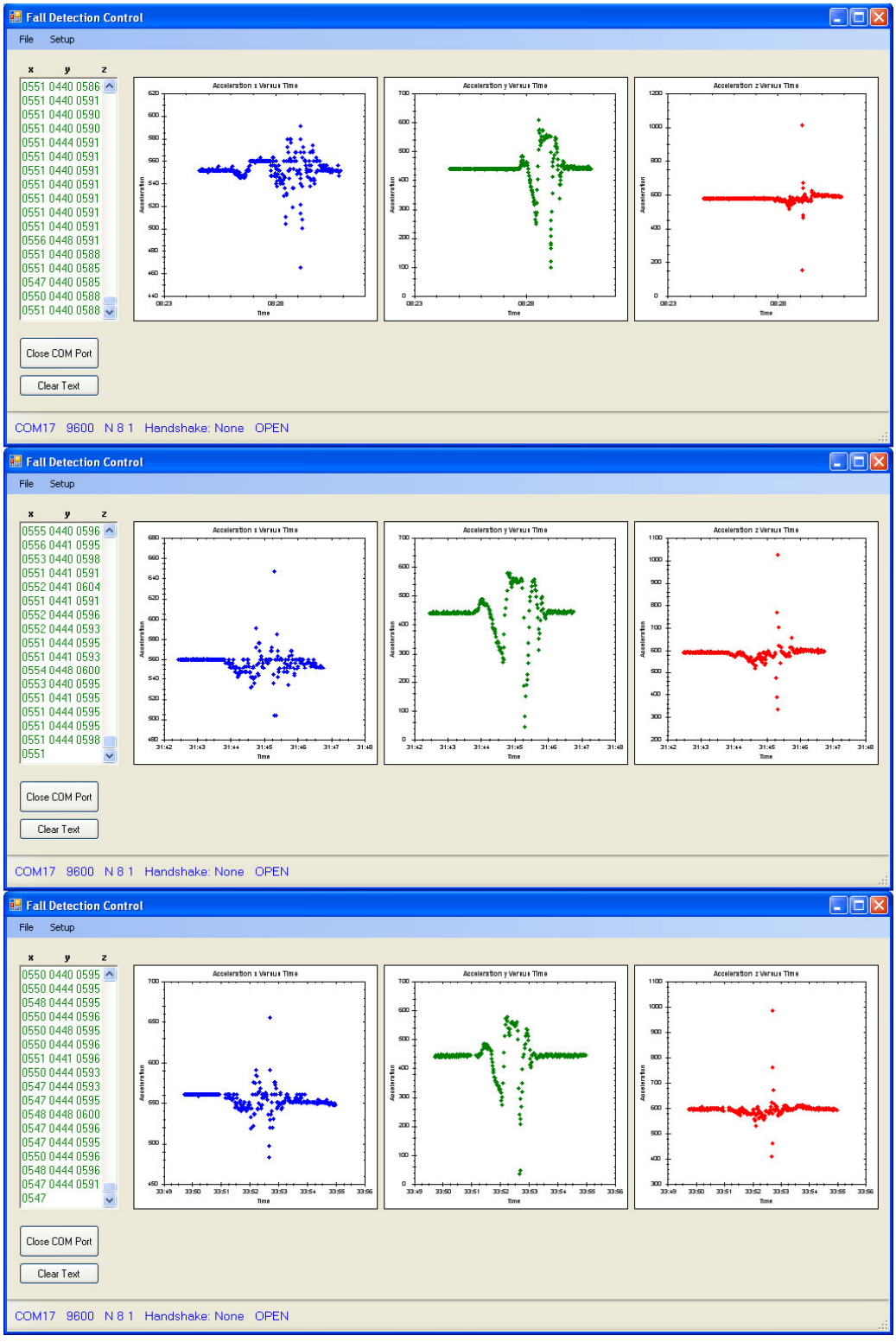
```
                  }
            }
      }
}

void initPIC()
{

      SPBRG=129;                          // set baud rate as 9600
baud
      BRGH=1;                                   // High speed data
transmission
      TXEN=1;                                   // Enable_Tx
      CREN=1;                                   // Enable_Rx
      SPEN=1;                                   // Set_Serial_Pins

      TRISE=0b00000111;
      TRISA=0b00101111;
      TRISD=0;
      ADIE=1;                                   //Enables the A/D
converter interrupt
      ADCON1=0x83;                        //Right justified. RA3-
Vref, RA0,1,2,5-analogue, RE0,1,2-digital
      ADCON0=0x41;                        //fosc/8, on A/D converter
      ADRESL=0;
      ADRESH=0;
      PORTD=0;
}

void display(unsigned int c)        // subrountine to display the
text on the screen
{
      while (TXIF == 0);
      TXREG = c;
}

unsigned char receive(void)               // subrountine to receive
text from PC
{
      while (RCIF == 0);
      a = RCREG;
      return a;
}

void GetSensor(int *SData)
{
      unsigned int temp;
      CHS2=0;CHS1=0;CHS0=0;
      DelayUs(20);
      ADGO=1;
      DelayUs(20);
      if(ADIF)
      {
            SData[0]=ADRESL;
            temp=ADRESH*0b100000000;
            SData[0]+=temp;
      }

      CHS2=0;CHS1=0;CHS0=1;
      DelayUs(20);
      ADGO=1;
```

```
	DelayUs(20);
	if(ADIF)
	{
		SData[1]=ADRESL;
		temp=ADRESH*0b100000000;
		SData[1]+=temp;
	}

	CHS2=0;CHS1=1;CHS0=0;
	DelayUs(20);
	ADGO=1;
	DelayUs(20);
	if(ADIF)
	{
		SData[2]=ADRESL;
		temp=ADRESH*0b100000000;
		SData[2]+=temp;
	}
}
```

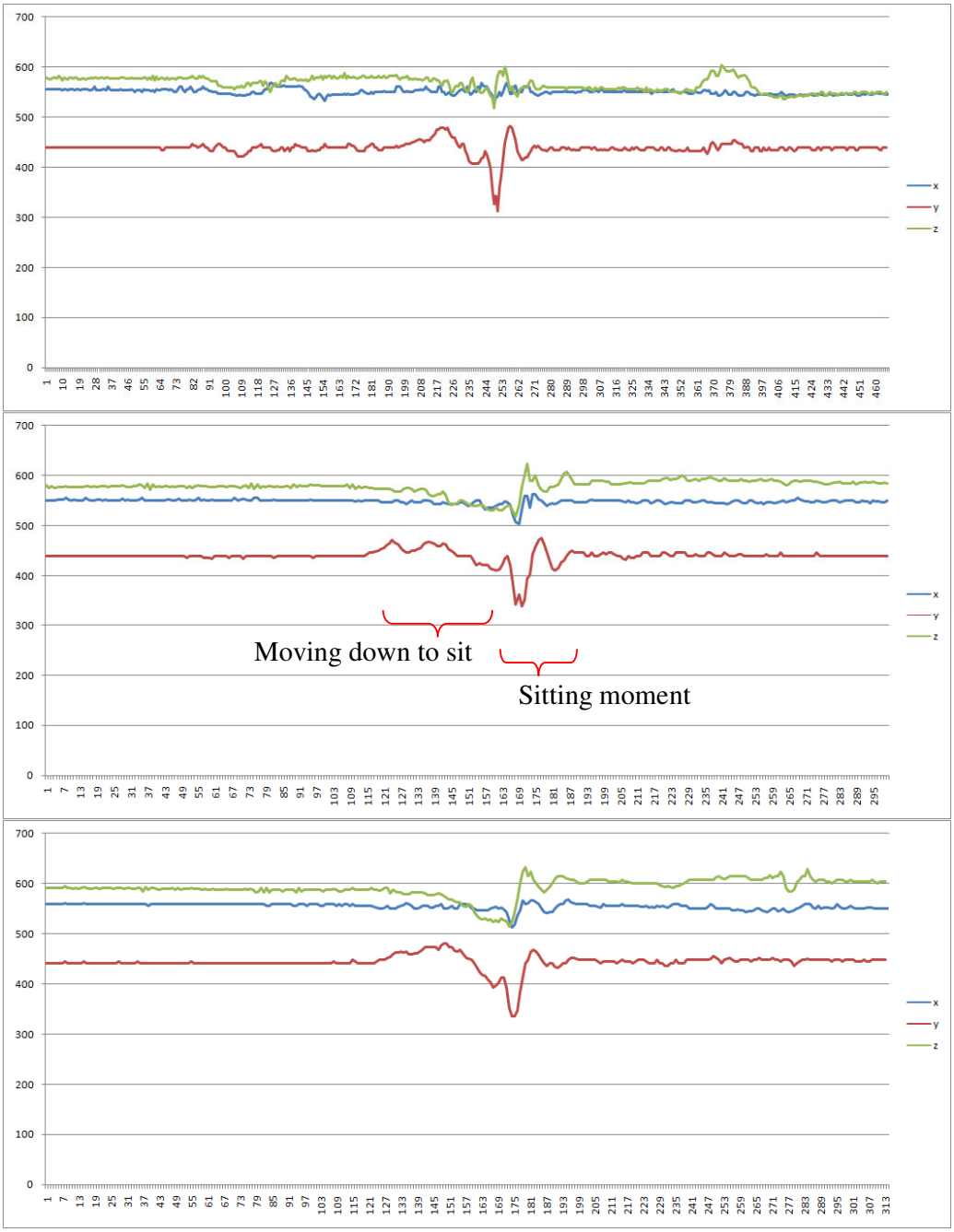
APPENDIX C: Graphs and Figures



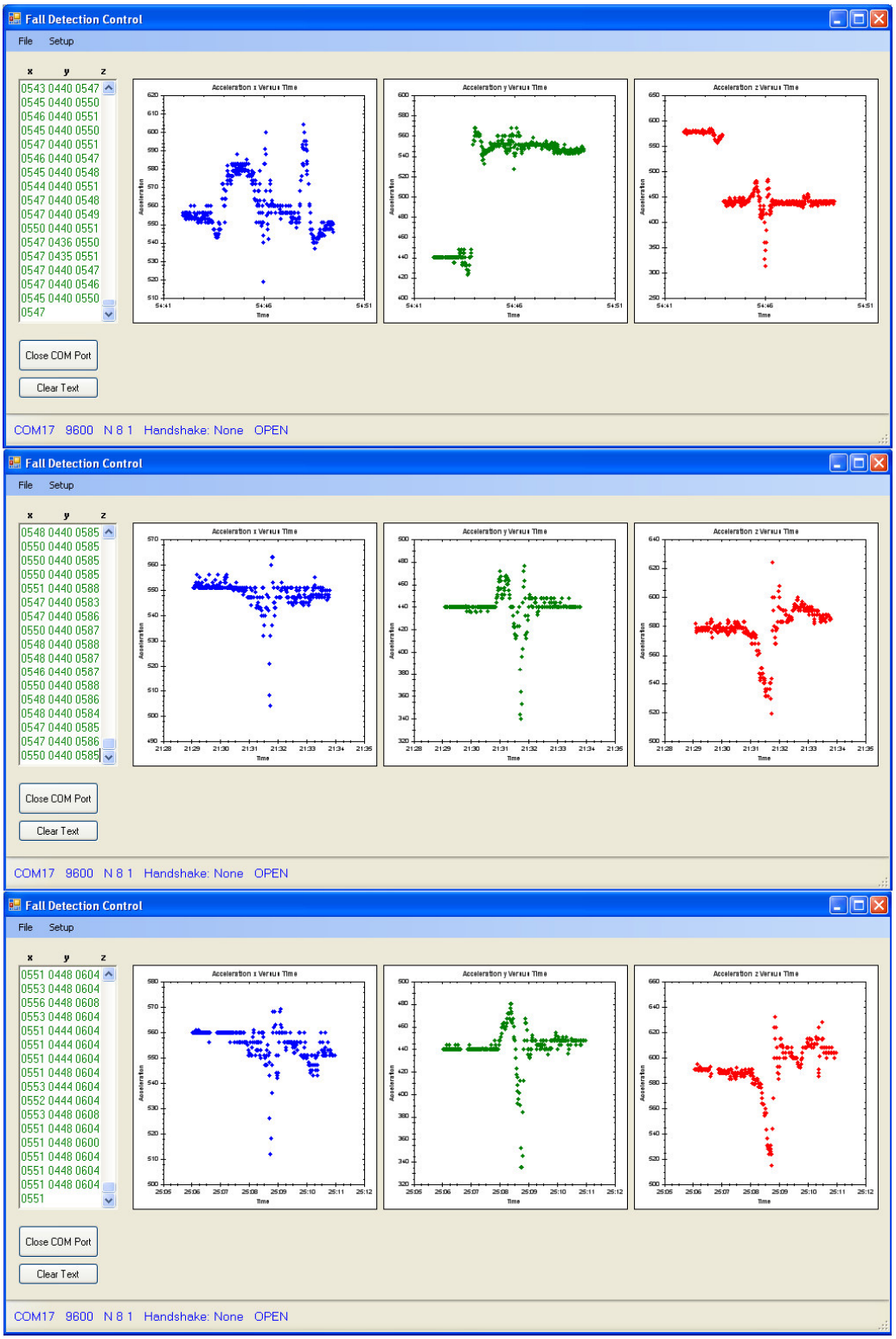
Jump motion



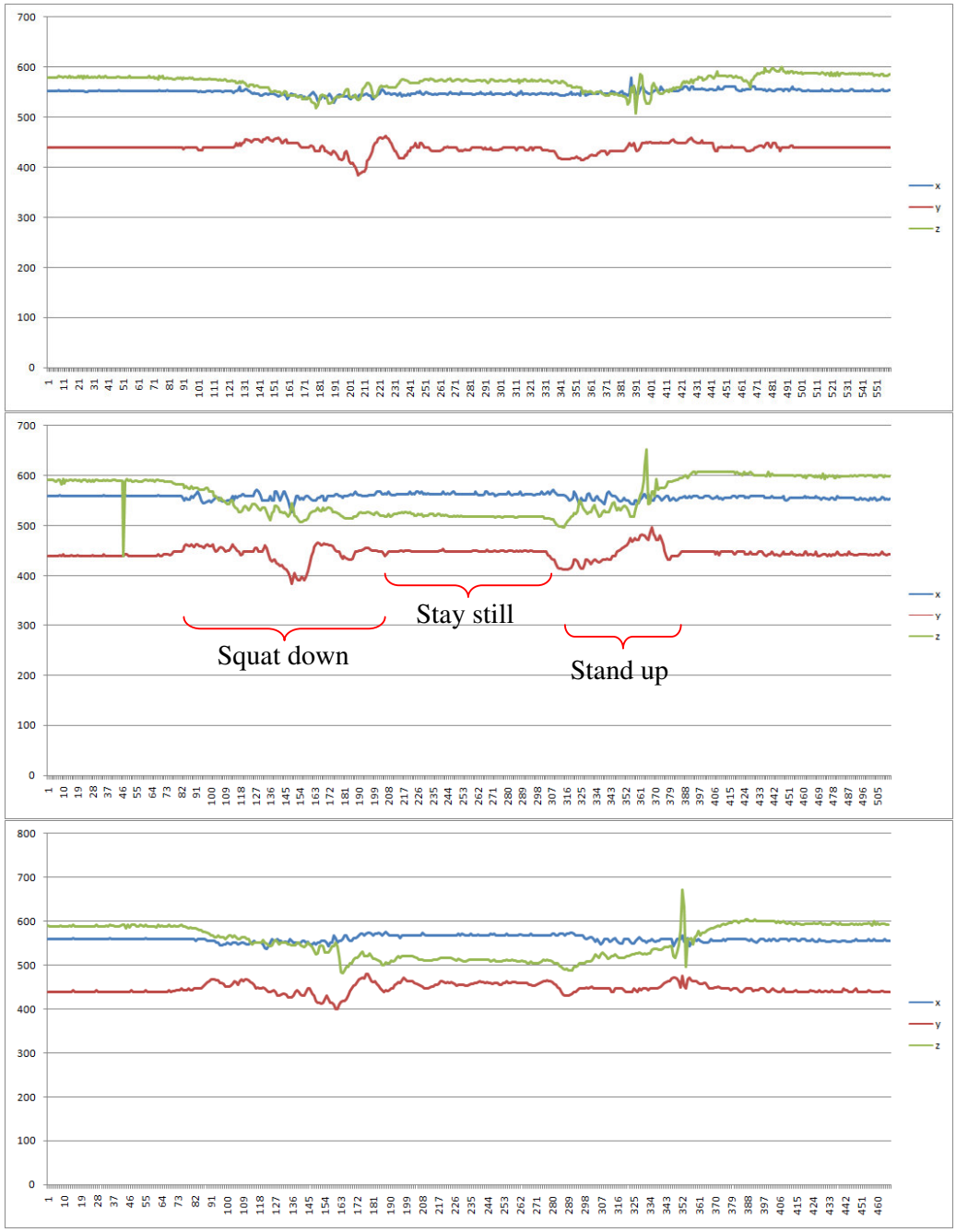
GUI when obtaining jump motion data



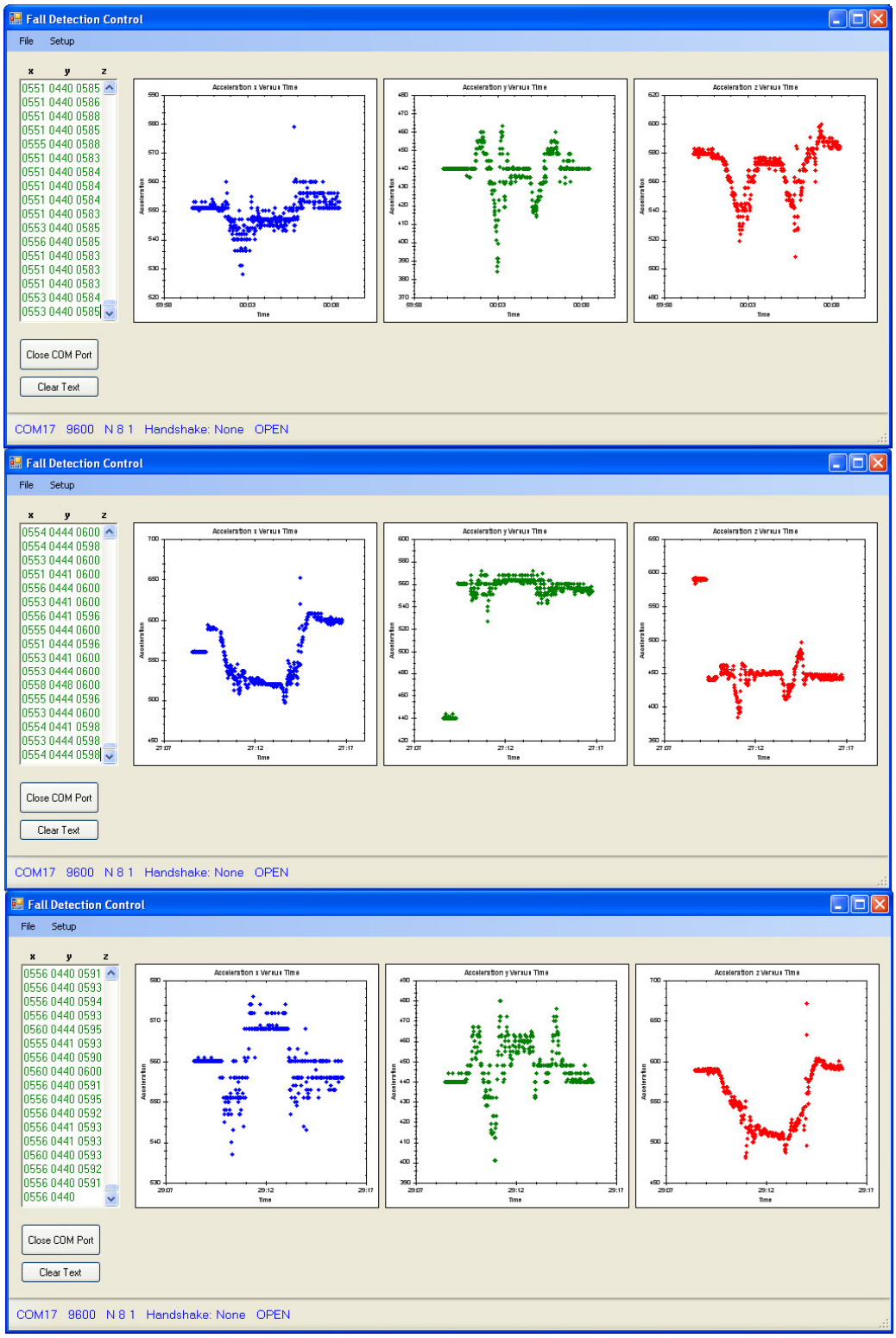
Sit motion



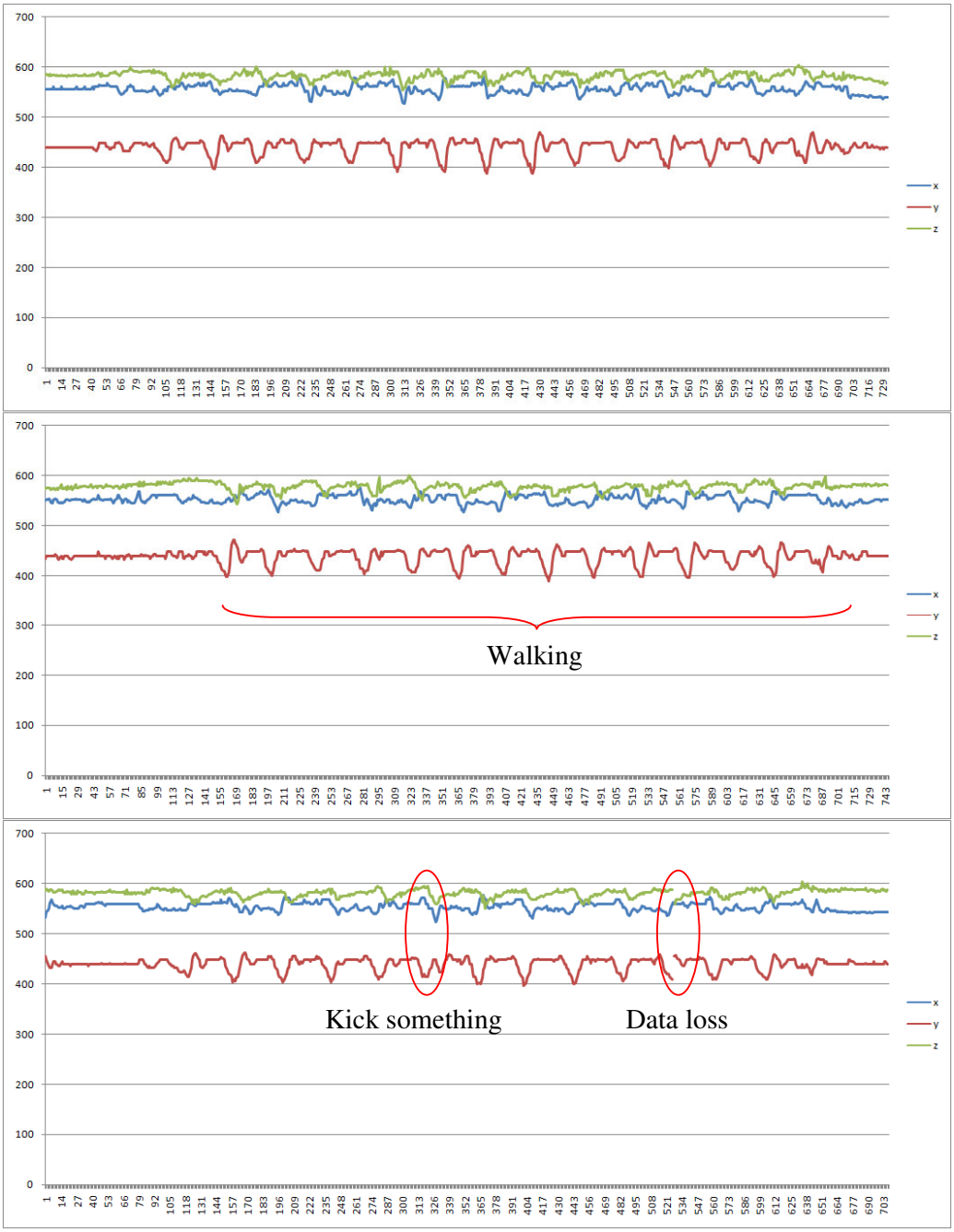
GUI when obtaining sit motion data



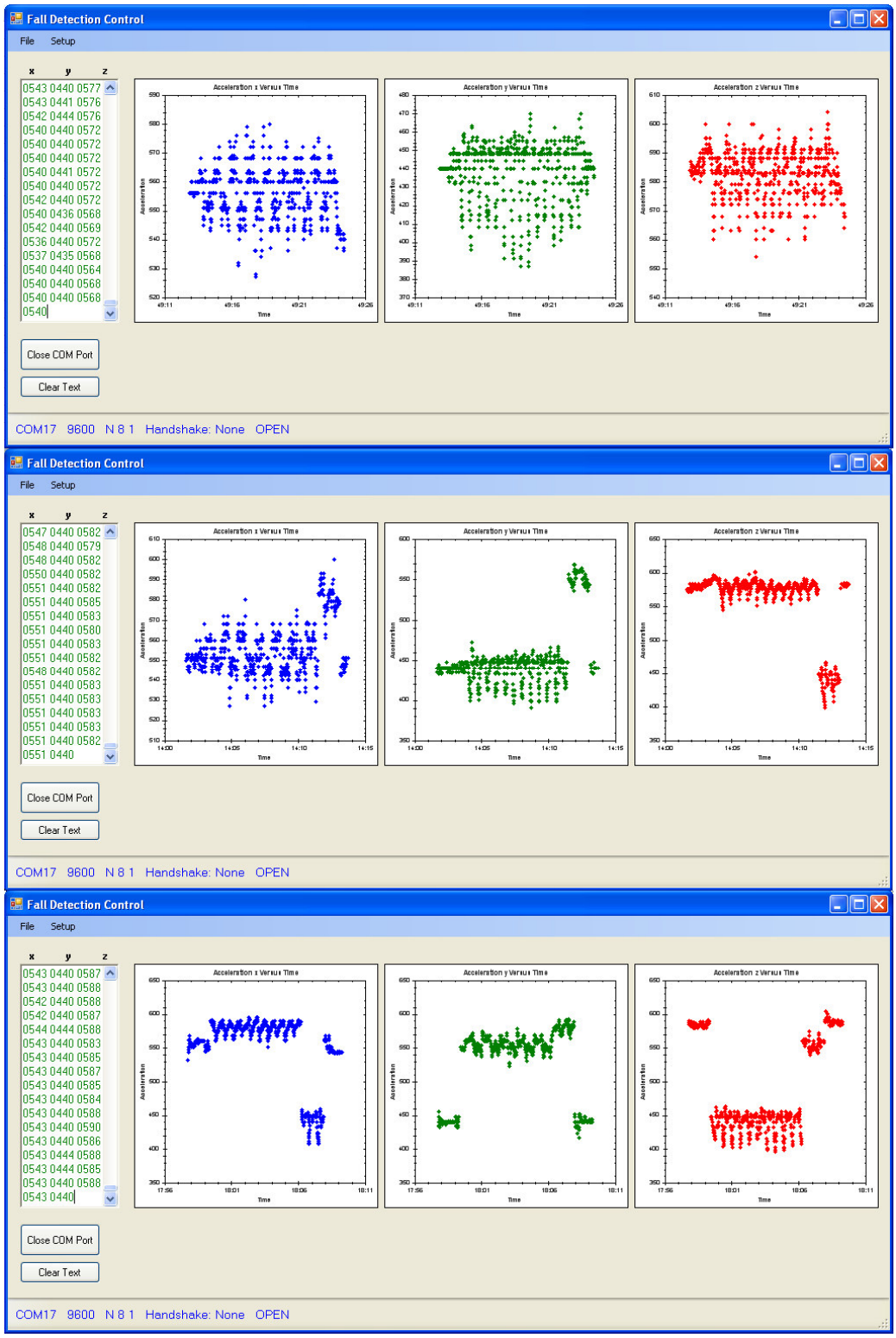
Squat and stand back motion



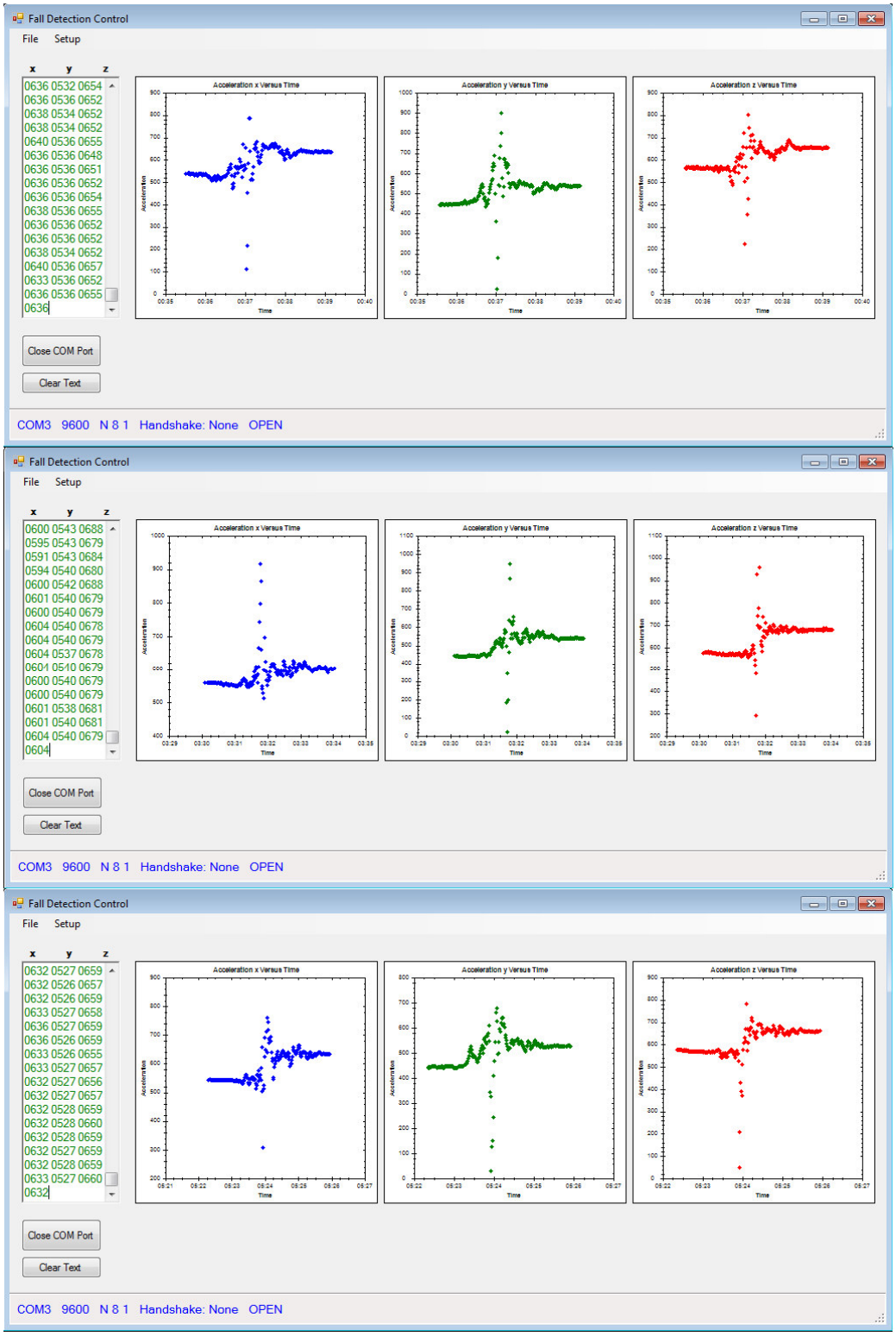
GUI when obtaining squat and stand back motion data



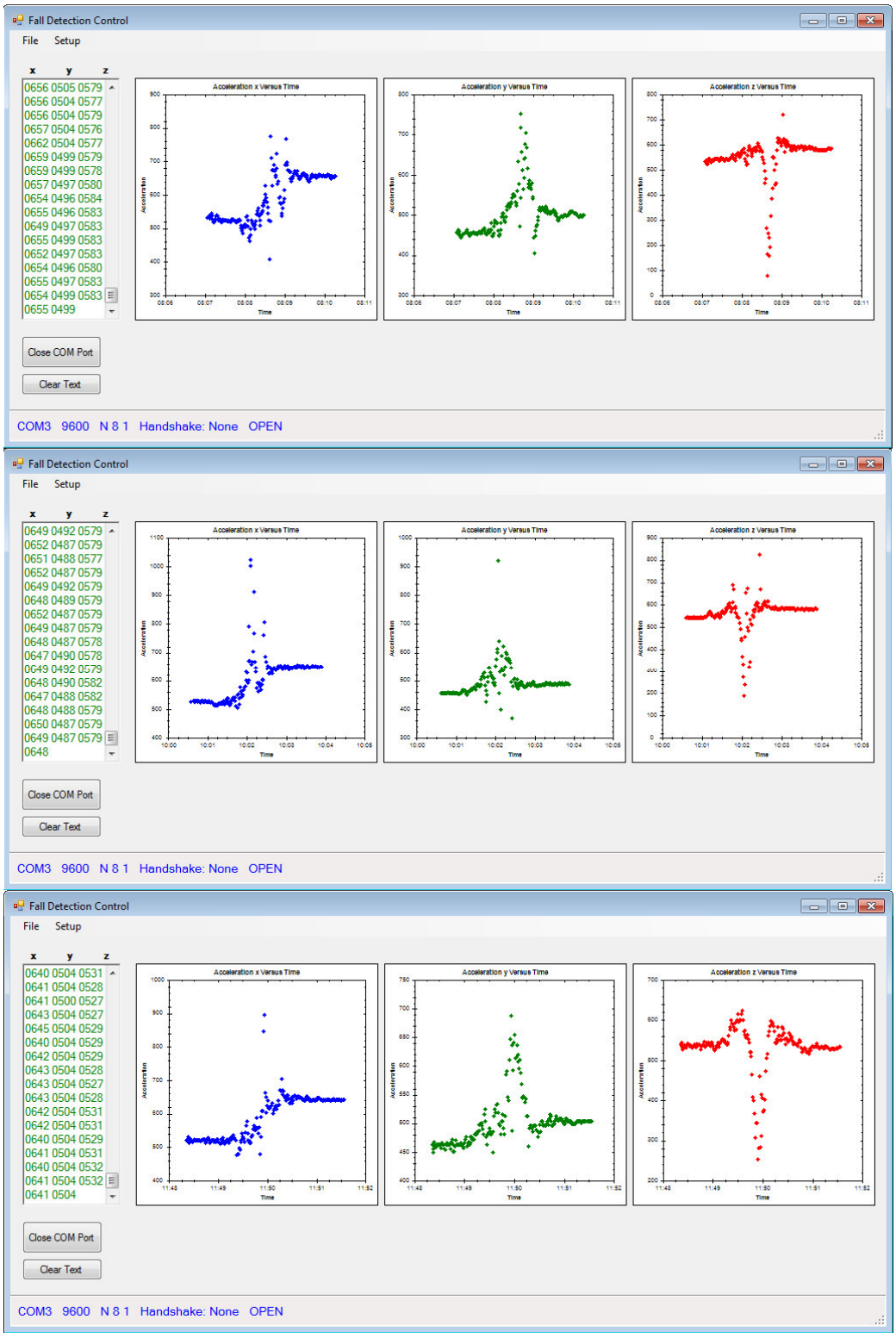
Walk motion



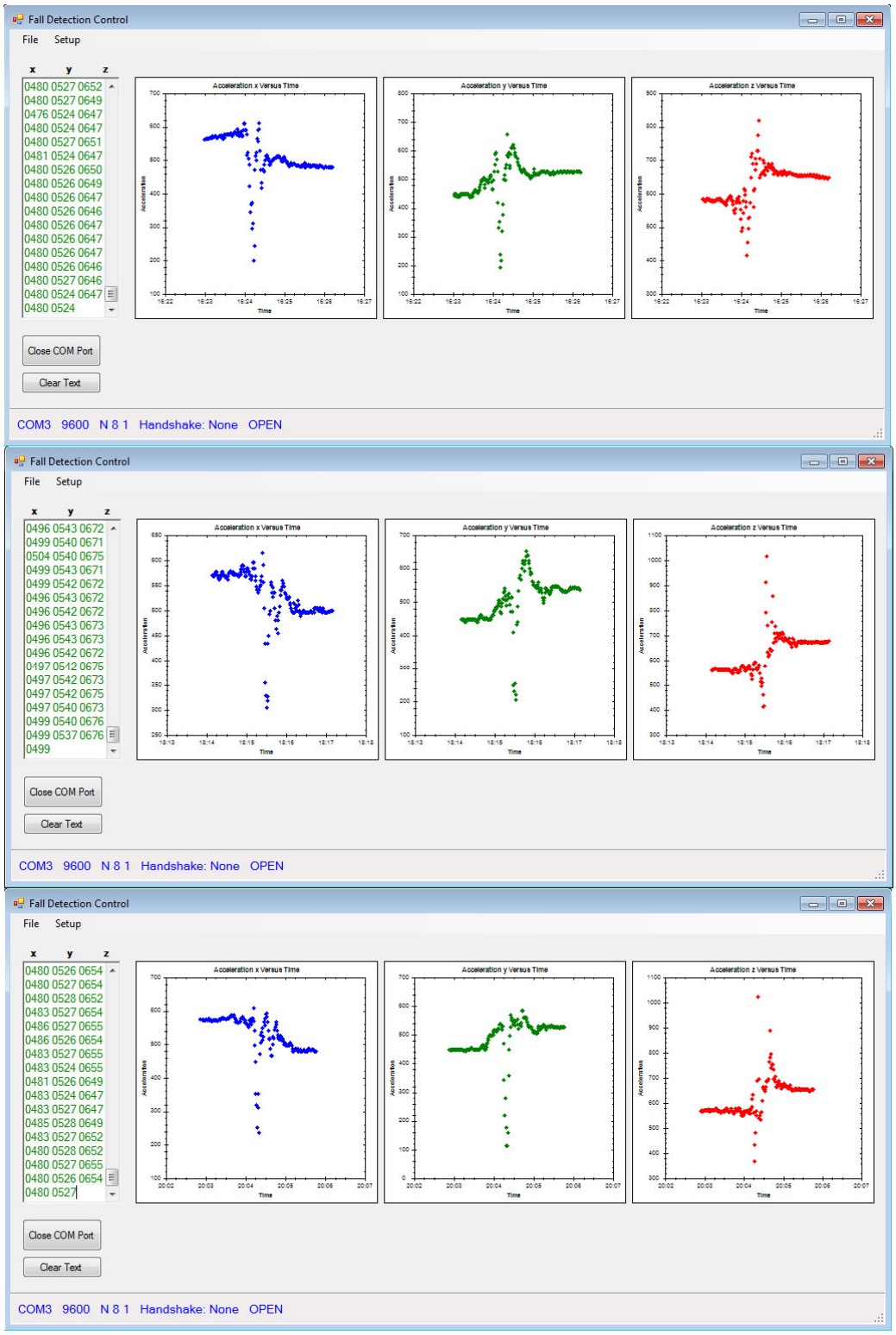
GUI when obtaining walk motion data



GUI when obtaining back falling motion data



GUI when obtaining left falling motion data



GUI when obtaining right falling motion data