RESPONDING TO DANGEROUS ACCIDENTS AMONG THE ELDERLY: A FALL DETECTION DEVICE WITH ZIGBEE-BASED POSITIONING

A Thesis presented to
The Faculty of
California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Biomedical Engineering
By
Michael Putnam
March 2012

COMMITTEE MEMBERSHIP

TITLE: Responding to Dangerous Accidents Among the Elderly: A Fall Detection Device with Zigbee-Based Positioning

AUTHOR: Michael Putnam

DATE SUBMITTED: September 2012

COMMITTEE CHAIR: Dr. Lily Laiho, Biomedical Engineering

COMMITTEE MEMBER: Dr. Dean Arakaki, Electrical Engineering

COMMITTEE MEMBER: Dr. Tali Freed, Industrial and Manufacturing Engineering

Abstract

Responding to Dangerous Accidents Among the Elderly: A Fall Detection Device with Zigbee-Based Positioning

By Michael R. Putnam

The following paper describes a fall detection and activity monitoring system with position detection based on Zigbee transceivers.The main objective is to reduce the time taken for emergency personnel to respond to falls among the elderly. Especially when the victim is unconscious or delirious, position tracking reduces location determination time within a busy hospital or nursing home environment and facilitates immediate treatment. Reduced response times correlate to decreased morbidity and mortality rates. Background is provided on the major wireless network advances currently deployed in a healthcare setting for asset and personnel tracking, etiology of falls, and several methods of detecting falls using sensors and image processing techniques. Data analysis proves that a precise coordinate tracking system was infeasible using the XBee RF module (based on the Zigbee protocol) due to environmental noise, a poor antenna construction and lack of precise signal strength measurements. A primitive scheme with lower resolution and higher reliability associating a single location with each Zigbee transceiver was employed. A pedometer function was added to the project to monitor the user's daily activity and to potentially serve as a predictor of falls through the interpretation of mobility and gait patterns related to step counts.

Keywords: Accelerometer, Zigbee, XBee, Fall Detection, Indoor Positioning, Location Tracking

ACKNOWLEDGEMENTS

I know I cannot do justice to the all the generous, eager, and adept individuals that have aided with this project in the span of a brief paragraph, nevertheless, I'll attempt to express my gratitude for their invaluable assistance. I'd like to thank my family for supporting me in my decision to return to graduate school to pursue a master's in the exciting and innovative field of biomedical engineering. Many thanks to the IEEE student chapter for teaching me how to accurately characterize the capabilities of the XBee wireless communications. Thanks to the QL+ lab for providing me with a space to tinker with and test my design. Thanks to Yashar Bahman for his deft insights into development of the software platform and to Zachary Tiulentino for his help in guiding me down the tortuous path of initial conceptualization to a definite project idea. I greatly appreciate Professor Arakaki's assistance in providing testing time within the anechoic chamber and aiding in executing and analyzing experiments, which ultimately allowed me to understand the limitations of the XBee system. I am especially grateful to Professor Freed for opening my eyes to the capabilities of wireless asset tracking through her exploratory class on RFID technology. Most of all, I would like to acknowledge the support and inspiration provided by Professor Laiho, both for suggesting this important project, with its significance to reducing injuries, deaths and health care costs among the elderly, and for continued guidance along the way to its completion.

Content

List of Figures

List of Tables

Introduction

Objective

According to the population reference bureau, the number of Americans over the age of 65 will more than double from 40 million to 89 million by 2050, while the relative number of working age adults capable of supporting this growing segment will shrink [1]. Given high health care costs for seniors coupled with high existing costs driven by spending on technological services, new drugs and long-term care services for chronic illness, there exists a strong demand for novel methodologies which can relieve the burden placed on conventional care [2]. Fall detection and pedometer-based physical activity monitoring are two such areas where medical devices can reduce the pressure on traditional health care infrastructure.

A combined fall detection and pedometer system with approximate position estimation capabilities was designed and tested to determine its ability to consistently and accurately track falls and daily activity within an indoor setting. Numerous studies have examined fall detection devices as a means of reducing the alarmingly high morbidity/mortality rate associated with injurious falls among the elderly. Additionally, many have investigated deploying wireless location tracking within a hospital, nursing home or residential living space in order to monitor patient activity and physiological data. Integrating these two technologies in an indoor location-aware fall detection system is a logical next step, providing the general location at which a fall occurs and further reducing medical personnel response time for fall events. A project named Complete Ambient Assisted Living Experiment (CAALYX) funded by the European Commission combined fall detection and GPS location tracking, however its intended focus is outdoor

positioning [27]. The pedometer is included to monitor activity, encourage exercise as a form of preventative care, and to serve as an investigational tool for determining the cause behind fall instances. The proposed system is designed to detect falls and paces using a differential capacitive accelerometer and relay indoor position via communication between Xbee wireless modules.

Background

Home healthcare monitoring

The integration of wireless networks with diagnostic sensors within a home environment has the potential to greatly improve health care accessibility and efficiency for the growing number of seniors and retirees in developed nations, without straining existing resources. By reducing risks associated with home health-care, more seniors can enjoy the independence and higher quality of life afforded by living at home. Examples of home healthcare wireless monitoring include real-time location tracking, telemedicine, blood pressure monitoring and tracking, alerting the deaf about critical audible information, and fall detection [3]. According to a study by Varshney, such technologies could improve quality of service for patients in both cities and rural areas, enhance the productivity and retention of health-care providers and reduce long-term costs [4]. Wireless devices can provide a constant update of the health of a subject and can immediately alert medical personnel to any physiological irregularities or injuries that may occur, enhancing providers' ability to predict disease and adverse conditions and reducing complications that may require costly trips to the emergency room.

There are numerous hospital wireless networks examples that may be extended to a residential setting. A network of RF-based wireless sensors can be used to track the exact location and condition of patients and link this real-time information to existing electronic medical records minimize diagnosis and treatment time allowing physicians to identify patients in need. In a mass casualty event such a network enhances first responders' ability to triage and utilize limited human resources to enable hospitals to handle larger patient loads [15]. Smart hospitals using wireless networks to track patient IDs have been proposed with the aim of minimizing the hundreds of thousands of deaths annually attributed to preventable in-hospital errors. For instance, a nurse can read a patient's tag using a reader embedded within their smart phone or PDA in order to gain access to electronic medical records and check that the correct patient is transferred to the operating room at the correct time [16]. An investigation of the prominent methodologies used to track and identify the status of assets within a health care environment is provided below.

Methods of Wireless Healthcare Monitoring

RFID

The primary application of Radio Frequency Identification (RFID) in health care is asset tracking. The technology involves RF communication between tags and a reader, with the reader requesting data from tags and the tags responding with their unique electronic product code (EPC) in addition to other relevant data based on the application. The reader transmits received tag ID information either directly to a computer/smart phone or through a network via

a coordinator/router. ID information can be used to acquire more extensive information on the tagged asset from a database. Figure 1 provides a flow diagram for a typical RFID network.

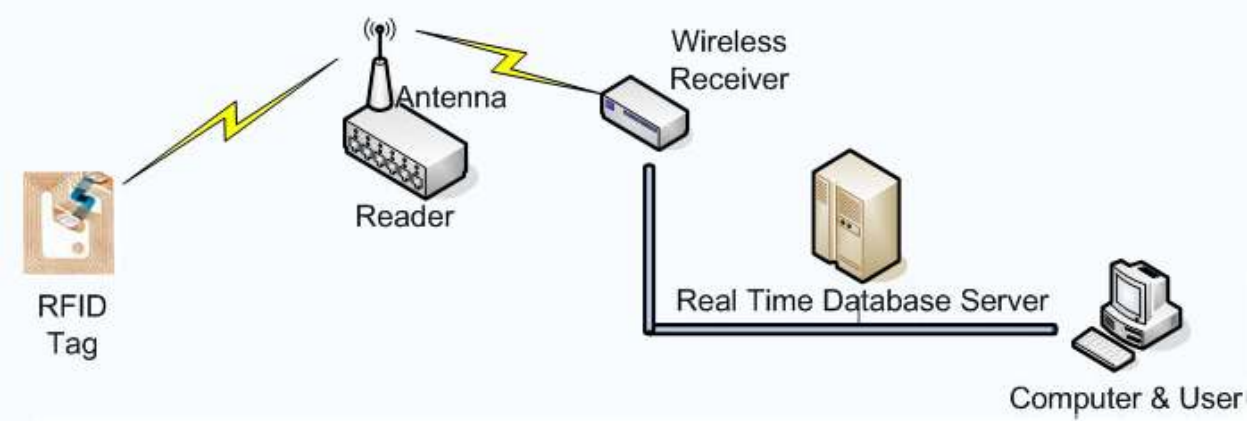

Figure 1: Diagram of an RFID System

Tags can be classified according to their power source and operating range. Active tags contain their own power source allowing data transmission without reader interrogation and operate at high frequencies which allow communication with a reader up to a distance of 500 meters [17]. Passive tags are powered by the read signal sent from the reader, either through the principle of mutual induction or backscattering. They operate at low frequencies that correspond to read ranges from a couple of centimeters to 10 meters, making them ideal for close-contact applications [18]. Semi-passive tags also have their own power source, but like passive tags must be energized by a reader in order to transmit data. Their signal range falls within the range for active and passive tags.

Within a hospital, clinic or nursing home environment, stationary readers can be used to monitor mobile assets. By placing readers at the entrances and exits to operating rooms, physicians can determine if any tagged surgical instruments were left within a patient

following the operation [17]. Readers can also be placed within medicine cabinets and prescription medications can be tagged and cross-referenced with patient IDs to ensure that patients are taking the correct medications at the correct times. The status of expensive equipment such as automated external defibrillators and artificial heart and lung machines which are periodically leased or rented to other facilities, can be monitored [17].

Mobile devices such as smart phones and PDAs can also serve as RFID readers. Patient records can be accessed through the mobile operating system of such devices by scanning passive RFID-enabled wristbands. Heart rate, blood pressure, temperature and O_2 saturation levels from biological sensors can be accessed and updated from the hospital database via mobile scanning of RFID tags. By scanning a patient tag in conjunction with machine read labels placed on blood vials, such "point-and-shoot" capabilities could be used to update the hospital information system whenever blood samples are collected from patients [17].

Some examples of commercially deployed RFID tracking systems include the Aeroscout system, Ekahau's Real Time Location Tracking System (RTLS), and AwarePoint's Aware360° suite™. All employ IR beacons or exciters that engage any active tags within their IR footprint, prompting them to send data either through a wireless network of sensors to a central hub or directly to an RTLS controller where position is approximated. This data can be sent to a smart phone, tablet, PC or kiosk via a Zigbee or WiFi wireless network. The WiFi option makes use of existing voice and data infrastructure to route asset data to the controller. The controller subsequently uses either received signal strength or time of arrival in order to triangulate position and asset location is displayed in software, typically overlaid on existing

floor plans of the facility. Health care applications using these systems include inventory management, patient flow analysis, hand washing compliance, and automated scheduling of operations and check-ups. Figure 2 displays the Aeroscout system, including tags, beacons and connecting bridge module.

Figure 2: Awarepoint RFID Sensor

Bluetooth and Zigbee

Bluetooth and Zigbee are modes of short-range (10 meters) wireless communication based on the IEEE 802 standard which are mostly recommended as a means to send biometric data concerning a patient's vital signs to a Wi-Fi enabled terminal. Though both can be used for location tracking applications, they rely on received signal strength in order to triangulate the position of an asset, a method which can lead to highly disparate results based

on the degree of interference between a source and receiver. In comparison to ultra wide-band (UWB) and other RFID-related modes of communication which can detect asset position within 50cm, the maximum accuracy achievable with Bluetooth and Zigbee is a couple of meters.

Bluetooth is a protocol envisioned as a replacement for cables which can be used to create an ad-hoc personal area network between a master and up to 7 slave devices [18]. Bluetooth has a relatively higher power consumption and higher data rate than Zigbee [18]. Typically Bluetooth provides a machine to machine interface between pairs of devices: mobile phones, computers, faxes, or In the case of home healthcare, between a biosensor and a terminal computer [18]. For instance, Bluetooth can be used to transmit systolic and diastolic pressure readings from a cuffless blood pressure monitor to a PDA [18]. Additionally Bluetooth enabled phones combined with Bluetooth beacons have been used as part of a location tracking system for locating patients and personnel within an operating ward [19].

Zigbee is a method of mesh networking, with each network capable of supporting up to 65,000 nodes. The physical and medium access control layers are defined according the 802.15.4 protocol, while the actual Zigbee functionality is provided by the network and application layers. A mesh network allows end devices to communicate via paths through other nodes in the network, with a coordinator node managing how information is transmitted or received [21]. Though it transmits data at a lower rate than Bluetooth, it has a 1000-fold lower latency and a lower power consumption [18], making it ideal for clinical monitoring where any delay or loss of power can potentially be detrimental to a patient's health outcome. Zigbee has been used to establish a wireless body area network (WBAN) which integrates information from multiple physiological signal devices and transmits it to a mobile

system where vital signs can be monitored [22]. Like Bluetooth, it can also be used for coarse asset tracking. An example of a Zigbee mesh network is provided in Figure 3, while a hypothetical WBAN setup is described in Figure 4.

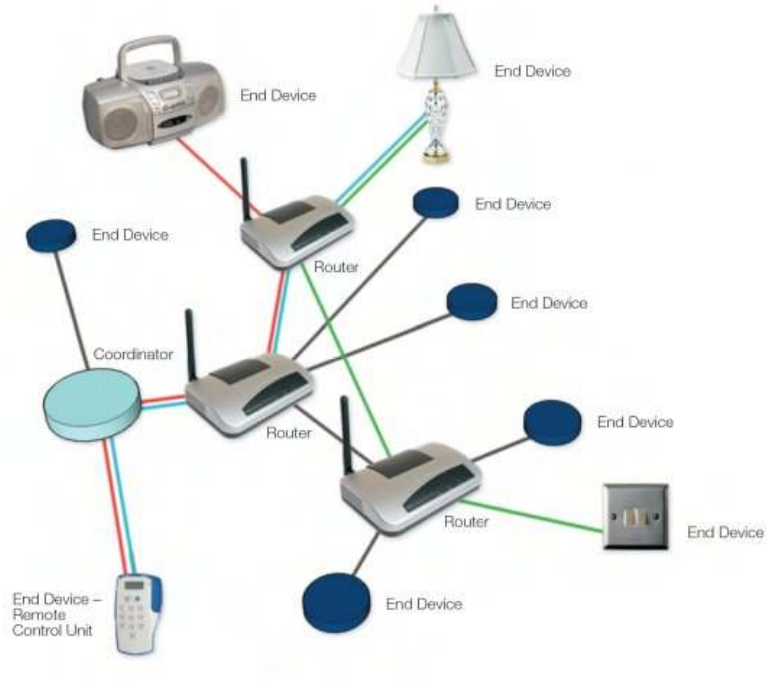

Figure 3: Zigbee Network Deployment

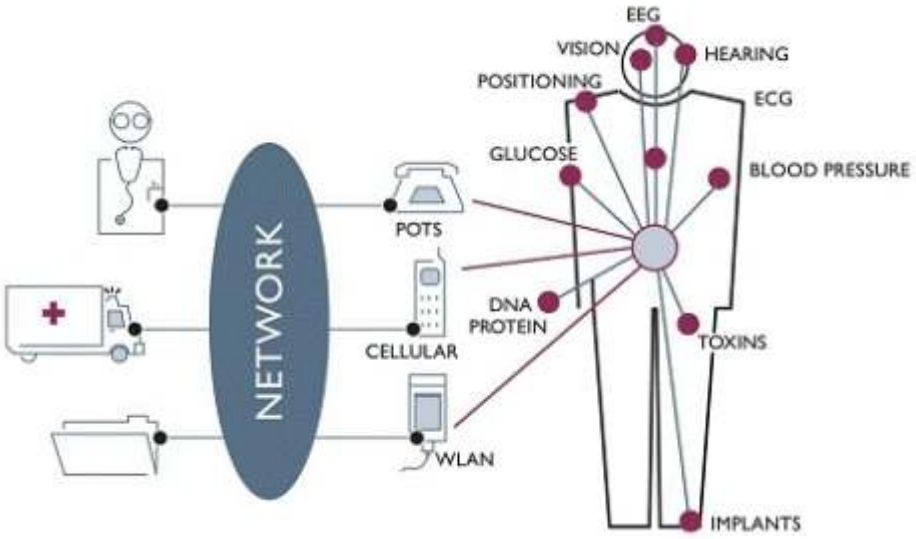

Figure 4: Wireless Body Area Network

Ultrasound

Ultrasound is high pressure audio waves above the frequency threshold for human hearing which can carry information in a manner similar to RF waves, though at much lower data rates. Although it is conventionally used as an imaging modality, ultrasound has also been considered as an alternative to RFID and 802 protocols in health care asset tracking because metallic objects do not adversely affect signal integrity [23]. The cricket compass is a system originally developed for context-aware mobile computing such as aiding the visually impaired in navigation, which detects both the orientation and position of a person. Active beacons transmit ultrasound pulses to a mobile device or tag, a transducer on a tagged asset detects the wave and sends an acknowledgement to a number of passive receivers, and the time-of-flight is used to determine distance of the tag from each receiver [24]. Ultrasound requires line of sight exposure between transmitter and receiver, which makes it unfeasible in busy, unpredictable environments. Due to this limitation, it has been recommended that

ultrasound be combined with RFID in a multimodal system where a less-accurate RF signal can be used in case line-of-sight conditions for ultrasound are not met [25].

Infrared

Infrared detectors provide short range line-of-sight radiation detection with a wavelength around 9.4 microns. The active badge system (see Figure 5) employs active infrared tags which periodically transmit IR signals to nearby IR receivers, providing room-level accuracy on where an individual is located [26]. In order to distinguish between users, each badge sends out a unique pulse-width modulated signal [26]. As a means of location detection, IR is inexpensive, simple to implement and has been widely tested and accepted within health care facilities, however it has limited range (<6m), low resolution, and requires line-of-sight exposure and carefully controlled light levels.

Figure 5: Active Badge Infrared Tag

Importance of Fall Detection

Fall detection is one application where the convergence of wireless networks and diagnostic sensors can improve quality of life, prevent complications resulting from serious injury and reduce the pressure placed on the existing health care system. According to Hornbrook et al., the estimated fall incidence among those Americans over the age of 75 each year is at least 30% [5]. These falls do not only result in major injuries such as hip fractures, lacerations requiring sutures, dislocations and sprains [6], but also cause psychological distress which further limits independence [7]. In regards to hip fractures, a 1990 study performed by Schneider and Guralnik reports that approximately 20% of victims do not survive the first year following injury, and another 20% have their mobility severely hindered, which can lead to institutionalization and further loss of personal freedom. As the proportion of the population 65 and older continues to rise, fall-related injuries and associated health care costs will become an increasing concern for patients and providers alike. According to studies performed by Gurley and Wild, the earlier falls are detected, the lower the rate of morbidity-mortality [8], hence the need for devices that can quickly and accurately detect the occurrence of these events.

The optimal solution to reducing the morbidity-mortality rate associated with falls is to prevent their occurrence through appropriate therapies, such as strength and balance training. Any successful attempt at prevention must answer the question: what biomechanical factors contribute to a higher susceptibility for falling among the elderly? Age alone is not a reliable indicator, due to the difference between chronological and physiological aging dependent on exercise, diet, medical history and genetic makeup. Although its effect on mobility impairment

has not been fully investigated, joint range of motion, especially rotation of the hip joint (which has been shown to decline up to 20% from the age of 45 to 75) may play a role in increased fall risk. The widely reported decline in muscle strength among the elderly, particularly with relation to joint torques needed to maintain postural balance, is believed to play a role in falls. Central nervous system reaction times, characterized as the time from onset of stimulus to the initiation of myoelectric activity, have also been shown to decline with age, especially when a choice of response options is involved. Reduced proprioceptive activity, which reduces neurological feedback and affects the ability to perform repetitive mechanical action such as walking, can lead to gait instability, which correlates with a higher risk of falling [36]. Decreased vision, cognition and a wide array of degenerative pathologies ranging from Parkinson's to osteoarthritis also make a significant contribution. The mobility impairments brought on by musculoskeletal changes (and not solely by old age) which increase fall risk are loss of postural balance, which can manifest as increased swaying, and gait variability.

Risk factors that significantly increase elderly persons' susceptibility to falling differ based on their association with either extrinsic (environmental) or intrinsic (patient-related) causes. For extrinsic causes, age, diabetes, a prior record of falls, and treatment with neuroleptics and bronchodilators have proven to be the greatest determinants. For intrinsic causes, those factors associated with extrinsic causes in addition to alteration of gait and balance, and dementia were most influential. As determined by Rubenstein et al., institutionalized patients are more vulnerable to intrinsic causes, while community-dwelling individuals are more vulnerable to extrinsic causes. Examining these risk factors, a definitive method of limiting falls is to closely monitor patient's drug intake and increase care and

surveillance of physical activity for those with diabetes or a history of falls. However, given the myriad of unique factors associated with each fall, and the disturbing lack of quantitative studies that provide a correlation between specific biomechanical and physiological factors and risk of falling (many falls go unreported), it can be difficult to predict and prevent injuries beforehand. This increases the focus on systems that allow personnel to respond quickly once the event occurs.

Fall Detection Methods

There are four different methods currently employed for fall detection among the elderly, including user-activated alarms, acceleration-based detectors, imaging-based detectors and passive floor vibration detectors. Each method is described below.

User-Activated Alarms

User-activated alarm devices consist of a wireless transceiver integrated into a wrist or pendant attachment. They require the wearer to manually activate an alarm in the case of a fall, at which time an alert is transmitted over a telephone or broadband network to the appropriate medical personnel. User-activated alarms are the most prevalent fall detection technology, due to their low complexity and affordability. However, they are ineffective in cases where a patient is incapable of activating the alarm due to loss of consciousness, pain, or trauma or because of a condition which impedes clear judgment such as dementia [12]. They also rely on user compliance; if the user takes the device off in the shower then an alarm cannot be triggered if a fall occurs.

Acceleration-Based Detection

Accelerometers measure acceleration by converting physical displacement of a proof mass with respect to its reference frame into an electrical signal. Piezoresistive accelerometers use piezoresistors arranged in a Wheatstone bridge to produce a voltage proportional to acceleration [9]. They are simple and low cost, however they are susceptible to temperature drift and output signals are relatively low [9]. In differential capacitive models the proof mass is encapsulated between two electrodes and deflection of the mass produces a proportional change in capacitance across the junction. These models are lower power, have a higher output range and are more sensitive due to lower noise levels [9]. Figure 6 shows a representative model.

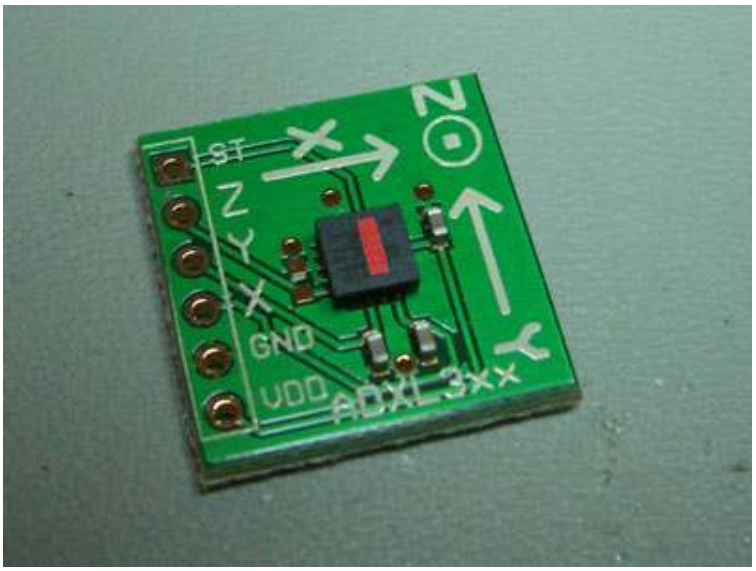

Figure 6: Typical Capacitive Accelerometer

Falls result in an “impact shock” where the downward motion produces a strong acceleration in the opposite direction. Acceleration-based fall detection involves using a

threshold to distinguish this large vertical acceleration vector from the smaller accelerations which are characteristic of activities of daily living (ADL). This is accomplished by tri-axial accelerometers (which measuring linear acceleration) or gyroscopes (which measure angular acceleration) placed on the thigh, waist, wrist, or head [8]. The vertical speed of movements such as rising, bending and sitting are used to set a threshold for fall detection, any speed that exceeds that associated with these movements is considered to characterize a fall. Due to subject-to-subject variability, accelerometers are typically calibrated using "supervised" or "unsupervised" learning. In the former case, subjects are asked to perform ADLs in order to determine execution speeds, in the latter subject movement is recorded over a couple of days and statistical analysis is performed to determine average speeds [8]. Accuracy in determining a fall event varies based on which phases of the event are taken into account (ie: the beginning of the fall, falling velocity, fall impact, and posture after the fall) [10].

The primary advantage of acceleration-based detection is its automatic fall detection capability, which addresses many of the issues associated with manually activated alarms. However, like manually activated alarms, effectiveness relies on user compliance. Additionally, users must remember to charge such devices routinely.

The Lifeline System with AutoAlert™ from Philips (see Figure 7) constantly monitors changes in height and orientation with respect to a horizontal position in addition to acceleration to distinguish between ADLs and a fall. A call to on-call emergency personnel is initiated when either the user presses a button or the user remains inactive for more than 30 seconds following detection of an impact shock. The MCT-241MD PERS system employs similar

accelerometer based detection in addition to wireless transmission of a unique alarm code using anti-collision algorithms to the base station in order to alert emergency personnel.

Figure 7: Philips LifeLine System

Image processing-based classification

Visual sensors can detect falls without requiring the subject to wear sensors or motion detectors. Image processing for fall detection involves a network of cameras which detects the transient presence and posture of a subject and determines abnormal periods of inactivity. One example investigated by Spehr, et al. uses a background subtraction algorithm to filter a person's image from his or her surroundings within a dense environment and then analyzes the temporal orientation to detect a fall [13]. Sixsmith developed an IR system called IRISYS which provides low resolution IR data representing position, size and velocity of a target to a host computer within a nursing-home environment [11]. IRISYS detects both dynamics characteristic of a typical fall as well as periods of inactivity in order to minimize the false alarm rate [11]. The software interface for the IRISYS system is shown in Figure 8.

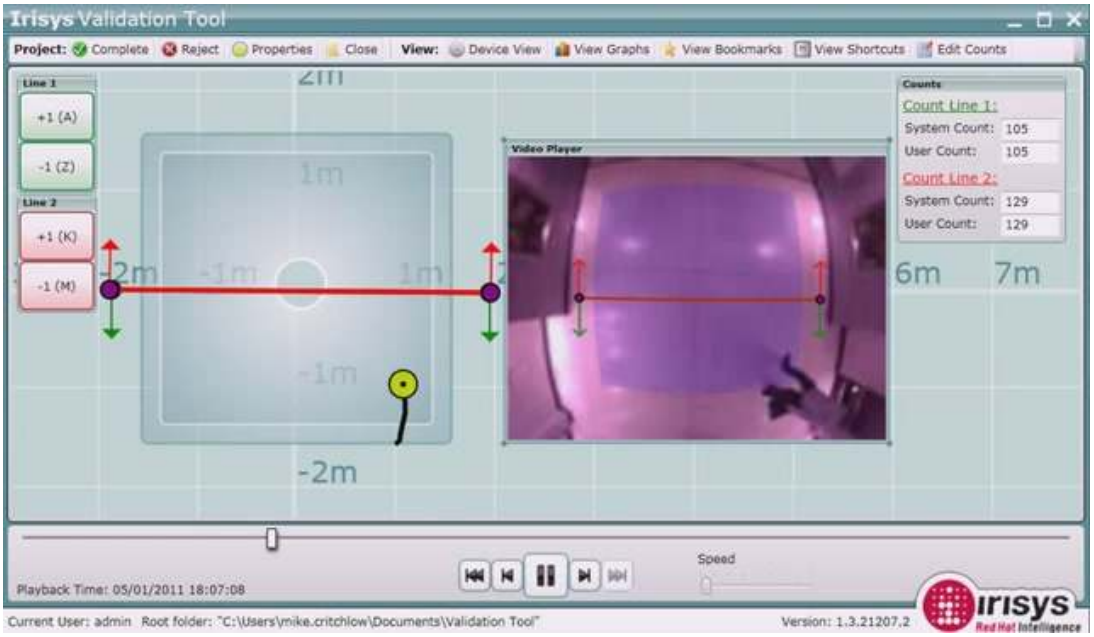

Figure 8: IRISYS Camera Detection Software

One drawback of image processing-based classification is that it can produce many false-positives (ie: a non-fall is recorded as a fall) because it cannot easily distinguish between cases when a subject is lying down or sitting on a sofa as opposed to lying on the floor following a fall [13]. Such systems are expensive and more complicated to install than acceleration-based devices, however they are less intrusive than wearable sensors and can be operated using readily-available image processing software.

Floor Vibration Based Fall Detectors

Vibration based detectors operate on the principle that different excitation activities produce unique vibration patterns. A fall will be detected when the appropriate pattern is measured. Sensors are embedded in the floor of each room within a nursing home

room and detect movement within a 15 ft range. This method can distinguish between human falls and the dropping of objects typically found in a nursing home [13]. The system's detection capabilities outperform the IRISYS system previously mentioned, which could only detect ~36% of falls within a test facility [13]. The main disadvantage of the vibration-based system is the costly installation process.

Pedometer-Based Physical Activity Monitoring

Pedometers, used ubiquitously to monitor exercise, have only recently been linked to improved health outcomes among users. They are useful motivational tools, enabling one to set practical goals towards maintaining a healthy lifestyle according to the mandate of the US Department of Health and Human Services, which recommends at least 30 minutes of physical activity per day for adults. This has the concomitant benefit of a reduced risk of heart disease, diabetes and stroke, which has the potential to reduce health care costs by billions of dollars per annum. In a study performed by Bravata, D, et al. the use of pedometers was found to positively correlate with increased physical activity, lower body mass index and lower blood pressure [14].

In addition to improving independent health outcomes, pedometers can provide physicians in an out-patient clinic setting with a gauge of a person's physical wellness, allowing them to predict the onset of disease and advise the patient to take the necessary precautions to maintain health. For instance, if a patient's step count and associated calorie count, which is monitored by an out-patient clinic on a daily basis, begins to decline significantly, a physician can use this data to quickly determine if the patient's sudden lethargy is a result of disease

onset, at which point they can request a check-up. Pedometers can also be used to monitor progress during rehabilitation following surgical procedures or treatment for diseases such as stroke or Chronic Obstructive Pulmonary Disease.

The pedometer step count can also be used to provide information leading up to a fall. A high amount of physical activity prior to the fall would indicate muscle fatigue played a role in the accident, while a lower amount might indicate that the patient was in a debilitated state. Pedometers can also be used to detect variability in gait, which is a contributing factor to a higher fall risk.

Device Design

Establishing Design Requirements

In developing the proposed fall-alert system, the most important requirements are accurate position estimation, high fall-detection and pace sensitivity, compact size and low power consumption. A target of ±3m was chosen for position estimation accuracy to correlate with RFID tracking accuracy values mentioned in the literature. Providing the approximate location was deemed sufficient because medical personnel can quickly locate the injured party once the general area is known. Fall detection accuracy was determined as the measure of how many actual falls occurred in relationship to the number of total falls, including false positives and false negatives, detected by the device. As any undetected fall could mean a potentially life-threatening injury, it is crucial that the device have close to 100% accuracy. It was determined that the size of RFID tags should be minimized to avoid encumbering the user's movement. Battery life should allow the user to leave the tag unattended for most of the day,

and only require recharging/swapping of batteries at convenient intervals. Ideally the beacons should be wall-pluggable to avoid constant recharging. For the prototype, system coverage should enable detection of fall location within a conventional residential dwelling, coverage being limited by the number of nodes available.

Engineering Requirements				
Spec	**Parameter**	**Target**	**Tolerance**	**Priority**
1	Location estimation accuracy	±3 m	N/a	H
2	Fall detection accuracy	90%	±5%	H
3	Size of user tag and beacons	3" X 6" X 2"	±0.25"	H
4	Weight	< 1 lb	0.5 lb	M
5	Battery Life	12 hours	±1 hr	M
6	System coverage	100 m2	10%	L

Table 1: Engineering Requirements

Design Development

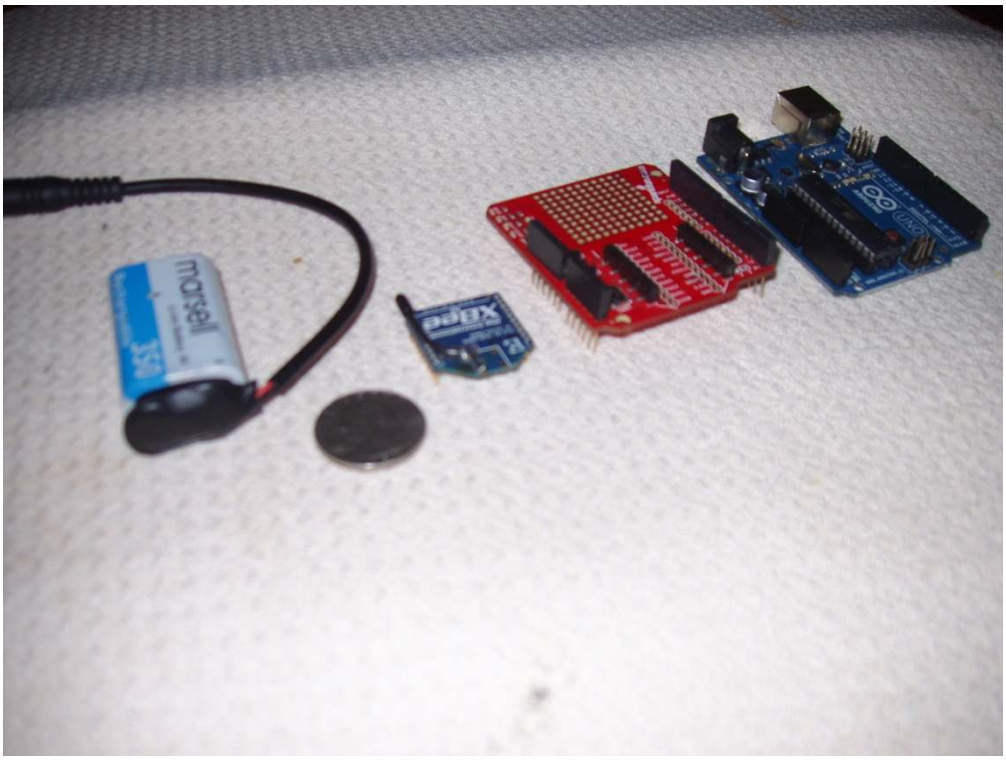

Figure 9: Hardware Components of Access Point

Location Tracking System

All location tracking options are capable of meeting the position estimation accuracy requirements, including RFID, Zigbee, infrared and ultrasound. An early design uses RFID reader software built into the user's cell phone along with passive tags located on the periphery of the region of interest to trilaterate user position. An advantage of this design is its low power consumption; passive tags receive power from the reader signal. High cost and complexity eliminated this idea from further consideration. Trilateration using ultrasound was also proposed. Time of flight measurements between a tag and a number of passive receivers are used to determine distance, which is translated into user position with high accuracy. Similar to the RFID-based system, this idea was abandoned due to high costs and design

complexity, in addition to sender to receiver line of sight requirements for accurate measurements.

A system based on Zigbee wireless transmission addressed the need for a low-cost, low-complexity system, while sacrificing some of the accuracy achievable with ultrasound. Tag location is determined based on signal strength measurements, and devices communicate via a well-established serial protocol. Power requirements were low compared to other wireless standards.

Fall Detection System

To meet the accuracy requirements of 90%, an accelerometer was chosen to detect forces associated with a fall. It requires significantly less infrastructure than an imaging or floor-vibration based system and has a faster response time than manually activated alarms. A differential capacitive model was chosen due to its wider range of detection, higher resolution and lower power requirements compared to piezo-resistive models.

Final System Design

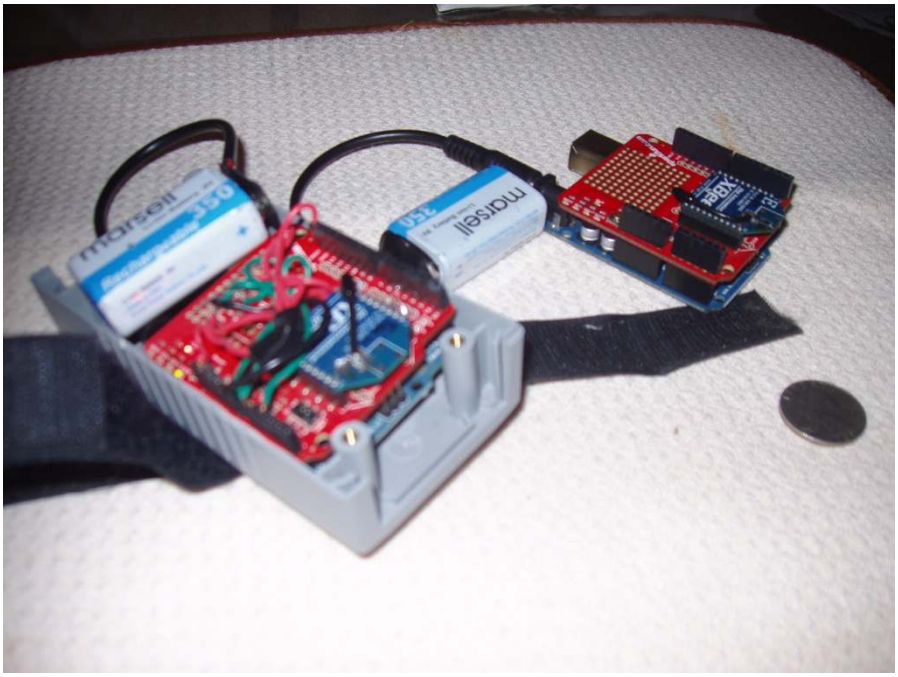

Figure 10: User Tag in Enclosure and Access Point

The proposed system integrates low granularity location tracking based on the Zigbee protocol with acceleration-based fall detection to transmit fall time and location within a room. A battery powered user tag worn around the waist communicates with Zigbee nodes located throughout the room in order to establish the subject's position. Through extensive testing (see following experimental data), it was determined that signal strength measurements were not a reliable means of pinpointing position. In lieu of such a trilateration system a simpler model linking each room to a unique tag was implemented. In the case of a fall, an alarm is triggered, and position data at the time of the fall is sent to a local PC with wireless connectivity, where software provides a visual indication of the fall event and logs this

information for future reference. The current system can be expanded to relays fall alerts to a home healthcare monitoring system via the internet, notifying medical personnel of fall events.

Hardware Components

Location-aware fall detection system hardware components and sub-components and justification for selecting each one is provided below:

- User Tag
    - MMA7361 Accelerometer
    - Xbee RF Module
    - MicroSD card reader
    - Arduino UNO Microcontroller USB Board
    - Li-Ion 9V battery
- Locator Beacons
    - Xbee RF Module
    - Arduino UNO Microcontroller USB Board
    - Li-Ion 9V Battery
- Local PC
    - ETRX2 USB Stick
    - Fall Alert Software

MMA7361 Accelerometer

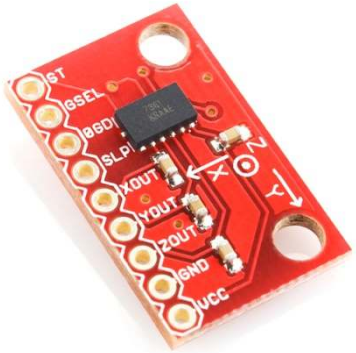

Figure 11: MMA7316 Accelerometer Breakout Board

The MMA7361 tri-axial capacitive accelerometer sends an analog voltage between 0-3.3V representing current accelerations to the Arduino UNO for processing. The device was selected because it has a range of +/- 6g with a resolution of 210mV/g, sufficient for reading 3-4g impact shocks associated with a fall event and discriminating these from accelerations experienced during ADLs such as sitting, lying down and walking. The device has a 1-pole low pass filter with a f_{3dB} = 400Hz to filter out EMI. When the device is not in use, it can be placed in sleep mode to limit power consumption to 3uA.

Xbee RF Module

Figure 12: XBee RF Module

The Xbee RF Module wirelessly transmits and receives data in order to estimate user tag postion via the locator beacons and send fall alerts to the local PC. The module consists of an ASIC which implements a serial command set on top of the ZigBee 802.15.4 physical layer and an antenna which sends data within the 2.4 GHz ISM operating frequency band. Operating at a transmit power of 2mW provides a theoretical indoor signal range of 40 meters, which allows coverage of a room which is 800m^2. The RF Module is mounted on a PCB which provides the 3.3V operating voltage and connects it to the microcontroller board.

Zigbee was selected for providing wireless connectivity because the project requires a communication standard which is designed to send relatively low-volume data packets and consume minimal power. In addition, Zigbee devices are inexpensive when compared to RFID and bluetooth, and associated coding for sending and receiving of data is based on an intuitive serial protocol. Furthermore, using Zigbee's mesh networking capabilities and the appropriate number of nodes the project can be expanded to send alerts across multiple nodes within a network, providing fall detection and localization throughout all rooms within a home or nursing home environment.

MicroSD Card Reader

Figure 13: MicroSD Card Shield

In the first design iteration, the card reader periodically writes the user tag 2-D position to a 1GB MicroSD Card. This data storage device is chosen because it provides generous space for accommodating extensive periods of data-logging and runs on the SDFAT file storage protocol, which allows data to be easily read as text files on a PC. The reader was to be mounted on a PCB which provides connection to the Arduino UNO board. However, when the final room by room location tracking model took precedence, it was decided that data saved to the card could be more conveniently saved to the software program, obviating the need for the SD card reader.

Arduino UNO Microcontroller USB Board

Figure 14: Arduino UNO Board

The ATMega328 microcontroller housed on the Arduino UNO board calculates acceleration magnitude based on accelerometer analog voltage readings, controls data transmission via the Xbee RF module, calculates 2-D position via a trilateration algorithm and sends fall alerts to the local PC whenever an acceleration threshold is met. The Arduino UNO was selected primarily due to its low power consumption, USB programming interface and its intuitive, user friendly development environment. The ATMega328 chip at the core of the device consists of an 8-bit RISC architecture running at 16MHz, which is sufficient for handling the low-complexity fixed point calculations involved in determining acceleration and location. It provides 32kB of flash memory and 2kB of SRAM for storing and running firmware code. In addition to its favorable hardware characteristics, many hardware peripherals have been developed specifically for the Arduino UNO and there is extensive code documentation and a large community of knowledgeable, dedicated users to draw upon in developing projects.

While adequate for a prototype design, the board size could be reduced significantly in the future by designing a custom PCB using the ATMega328 in a surface mounted quad flat package footprint.

Li-Ion 9V Battery

Figure 15: 9V Rechargeable Battery

9V 600 mAh Li-Ion rechargeable batteries power the user tag and the locator beacons. Assuming the Arduino UNO has a maximum current requirement of 50mA, these batteries can theoretically power the tag/beacons up to 12 hours. In reality the lifetime is significantly lower. In future iterations of the project, batteries with smaller size and longer life should be considered to reduce reliance on the user for recharging.

USB Z-Stick

Figure 16: USB Z-Stick

The ZStick operating on the 802.15.4 protocol receives serial data indicating fall alerts from the user tag and forwards it to the Fall Alert software. Operating in transparent mode, it serves as another node within the network and can transmit and receive data from any other node assigned to the same PAN ID.

Fall Alert Software

The fall alert software's main function is to provide a visual indication of when and where a fall occurs within a room. In addition, data concerning location, steps taken within each room and falls are automatically logged to a file for future analysis. A user can input biological data such as age and weight into the program to determine calories consumed during daily activities. The software is written using the visual C++ programming language, which allows for design of an event-driven graphical user interface. A fall event is indicated by an ASCII message which is intended in a future iteration to trigger transmission of an alarm signal to a remote client program.

A map of the facility to be monitored is hard-coded into the program, with specific tags assigned to specific rooms within the area. An icon indicating a tagged user's position monitors user position, as well as when a fall occurs. It is desirable to be able to load in schematic layouts and assign tags to rooms as part of a calibration routine, instead of having to customize the software code for each layout.

User tag communication is accomplished via a handshake routine to ensure data transmission is synchronized. First the tag transmits a unique ASCII character, the software sends back a signal acknowledging the request, and communication is established. The software receives data (steps, location and fall instances) from the user tag. Following data exchange, the software sends another character to the user tag indicating end of transmission.

Budget

Item	Name	Description	Vendor	P/N	Cost/Unit	Units	Cost
1	Xbee 1mW Wire Antenna	2.4 GHz	SparkFun	WRL-08665	$22.95	5	$114.75
2	9V to Barrel Jack Adapter		SparkFun	PRT-09518	$5.90	5	$29.50
3	USB Cable A to B	Programming Cable	SparkFun	CAB-00512	$3.95	1	$3.95
4	Arduino Uno	Microcontroller	SparkFun	DEV-09950	$29.95	5	$149.75
5	Xbee Shield	Xbee Adapter for Arduino Uno	SparkFun	WRL-09976	$24.95	5	$124.75
6	Arduino Stackable Headers	Set of 4	SparkFun	PRT-10007	$1.50	5	$7.50
7	9V Lithium Ion Battery Charger		SparkFun	PRT-10265	$9.95	1	$9.95
8	9V Li-Ion Rechargeable Battery	350 mAh	SparkFun	PRT-10053	$4.95	3	$14.85

9	MMA 7361	Triple Axis Accelerometer	SparkFun	SEN-09652	$19.95	1	$19.95
10	Xbee Zstick		Digi	XU-Z11	$49.00	1	$49.00
						Total	$523.95

Table 2: Bill Of Materials

Microcontroller Software Design

Software loaded into the ATMega328's onboard flash memory performs the following key functions

- User Tag
    - Fall-based detection
        - Acceleration vector calculations from accelerometer readings
        - Transmission of fall alerts to local PC
    - Location Tracking
        - Routinely sends out signals to determine which nodes are in the area
    - Communication with User Software
        - Routinely sends information concerning location, steps taken, and occurrence of fall to the GUI Fall Alert Software
- Locator Beacons
    - Send data to user tag upon request

Fall Detection Algorithm

Acceleration data from the MMA7361 is continuously received by the Arduino in the form of an analog voltage from 0-3.3V. This value is translated into a digital 10-bit value by the Arduino's ADC using the analogRead function. The digital equivalent is converted into a

value between 0 mV and 3300 mV and used to determine the acceleration in increments of g = 9.8m/s^2 according to the following equation:

$$\boldsymbol{A_x = \frac{V_x - V_{xo}}{\sigma}}$$

where V_x is the x-acceleration value in mV received from the accelerometer, V_{xo} is the x voltage offset in mV and σ = 206mV/g indicates the sensitivity of the device (ie: acceleration detection resolution). The voltage offset is set such that when the accelerometer is placed on a flat surface, $A_x = A_y = 0$ and $A_z = 1g$.

From A_x, A_y, and A_z (the acceleration in the x, y, and z direction respectively) the magnitude of acceleration is calculated as:

$$\boldsymbol{A_{total} = \sqrt{A_x{}^2 + A_y{}^2 + A_z{}^2}}$$

This value is compared to a predetermined acceleration threshold, when acceleration exceeds the threshold, the firmware sends a signal along with estimated position to the software indicating a fall via the sendSPICommand function. The acceleration threshold is determined by observing accelerations associated with ADLs such as sitting, lying down, and fast-paced walking and setting the value so as not to trigger a false positives.

The sendSPICommand sends an alert signal to the USB ZStick along with a checksum which verifies the data integrity. This signal is read through the serial port by the FallAlert software which provides a near-real time indication of the event.

Location Tracking

Locating the 2-D position of a tagged object within an indoor environment involves the translation of signal strength readings between several reference nodes and the tagged object into distances. Friis' Transmission Formula indicates that if the power of the transmitter and antenna gain of the transmitter and receiver are known the power of the received signal in dBm can be determined from the inverse square of the distance between source and receiver according to the following formula:

$$\frac{P_r}{P_t} = G_t G_r \left(\frac{\lambda}{4\pi R}\right)^2$$

where λ is wavelength, R is the distance between antennas, G_r is the gain of the receiving antenna, G_t is the gain of the transmitting antenna, P_r is the input power of the receiving antenna and P_t is the output power of the transmitting antenna.

Given distance, a trilateration algorithm which determines the point of intersection between the reference node signal footprints at the tagged object location can be used to estimate position. However, there are many factors within the environment which cause expected distances to deviate from those predicted using the Friis formula. Such factors include inter-signal interference, multi-path interference, reflection off metallic surfaces, diffraction,

and attenuation caused by transmission through objects. More sophisticated algorithms that use noise margins and probability maps are employed to address non-ideal environments.

Position Estimation Algorithms

Algorithms investigated in estimating position based on signal strength include simple point matching, K-nearest neighbors and Bayesian sampling. Each is discussed in some detail below:

Simple Point Matching

In simple point matching, signal strength is compared to a set of reference values +/- one standard deviation [27]. If the signal strength cannot be assigned to any zone, the error margin for each reference is doubled until the value matches one of the zones. Signal strength is measured from multiple access points to enhance accuracy.

K-Nearest Neighbors

Using the K-nearest neighbors algorithm, reference readings for each zone are acquired and nearest adjoining zone values are compared to calculated values and used to determine weighted averages representing the estimated location of an object. For a particular zone i and k individual Zigbee nodes, the weight for the ith zone is the product of the Gaussian distributions for each node signal strength:

$$W_i = \prod_{j=1}^{k} \frac{1}{\sqrt{2\pi\sigma_{ij}{}^2}} * e^{-(R_{ij} - \mu_{ij})^2 / 2\sigma_{ij}{}^2}$$

where R_{ij} is the signal strength for zone i received from node j, σ_{ij} is the standard deviation of that signal strength, and μ_{ij} is the mean of the reference signal strength

Using these weights, a position in x and y can be determined based on the relative positions of each node within the room.

$$[x,y] = \left[\frac{\sum_{j=1}^{k} W_j X_j}{\sum_{j=1}^{k} W_j} , \frac{\sum_{j=1}^{k} W_j Y_j}{\sum_{j=1}^{k} W_j} \right]$$

Bayesian Sampling

Bayesian Filtering determines position based upon a probability distribution of the target's location. A space is discretized and used to develop a Bayesian reference network derived from conditional probabilities of a target being measured at a particular location and an a-priori probability of the target being at a different location [28]. The probability distribution is then obtained by inverting this network [28].

Firmware Implementation of Location Tracking

Before a location tracking algorithm was programmed, several of the above algorithms were tested by processing data through Excel formulas to determine accuracy by acquiring average signal strength readings and standard deviations. The Xbee RF module

continuously outputs received signal strength values from -92dBm to -23dBm as a pulse width modulated signal with a maximum period of 64 us, the length of each pulse in microseconds is read by the Arduino microcontroller using the pulseIn command. 100 readings are used to determine the average and standard deviation in signal strength for each node within each of 9 zones. Experimental setup and data collection are fully described in the following section.

Quantitative Testing

Distance v. Signal Strength Determination

To develop a tri-lateration algorithm for determining position, it was necessary to find a mathematical relationship between signal strength and distance. One receiver tag connected to the computer was used to record average signal strength readings from a source tag which was placed at discrete 3 foot intervals away from the receiver, with measurements taken from 3 – 21 ft. The test was performed 5 times at 5 different source power levels in the QL+ Lab at Cal Poly.

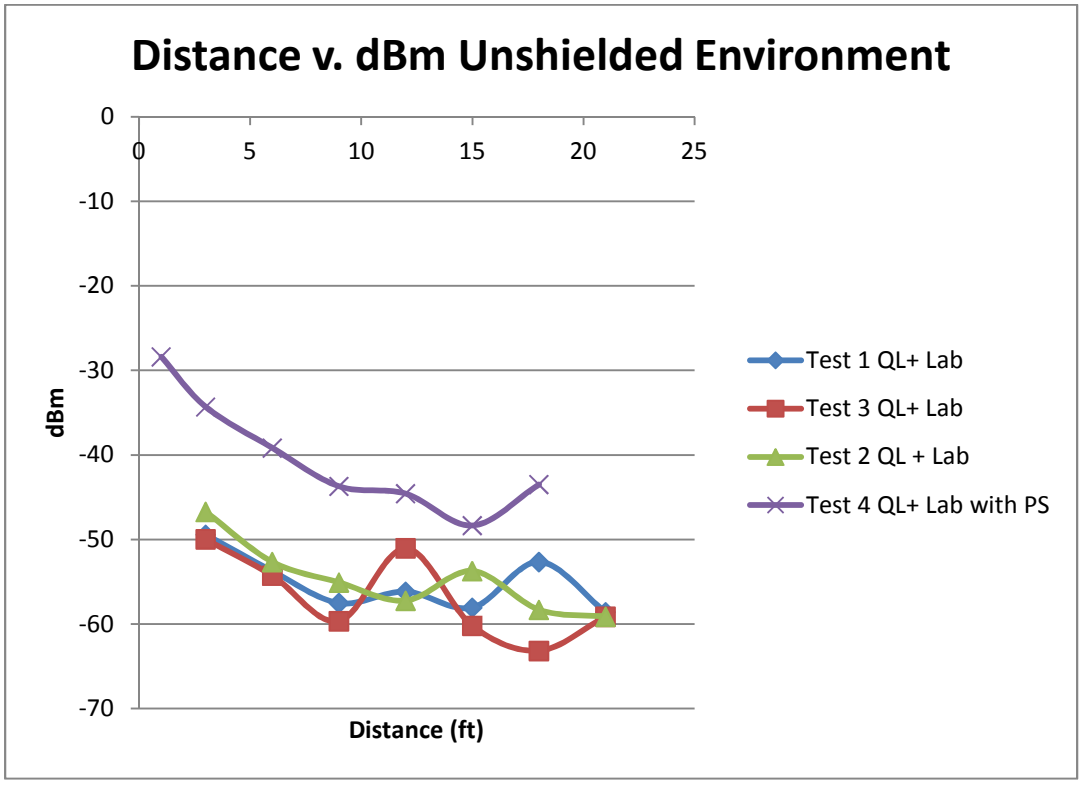

Figure 17: Distance v RSSI Unshielded Environment

As shown in figure 17 within 3-9 ft of the receiving antenna, signal strength drops off as expected, but as the source antenna is moved farther away from the receiver, values rise vary in a pattern not explicable by the inverse square relationship between signal strength and distance. In addition readings at a single position varied slightly with time. The likely cause of this undesirable behavior is inter-signal interference caused by other devices transmitting in or near the 2.4GHz channel such as Bluetooth, cell phones and other Wi-Fi enabled devices, as well as multipath, through which multiple paths of signal propagation interfere constructively and destructively with each other due to reflection. Given the difficulty of determining reliable distance measurements from signal strength, an alternate scheme based on raw signal strength values was proposed. This scheme does not rely on a system which follows Friis formula, but does require that signal strength measurements at individual points are repeatable and distinguishable.

Location Tracking Test

Experimental Setup

In order to test the raw signal strength scheme, a testing space was established in Cal Poly's Engineering Building 192, room 330. A 15' X 15' grid of 9 zones with centers spaced 7.5' apart was marked. RF nodes (access points) were placed at each corner of this grid for sending signals to the user tag. Due to slight variations in transmit power caused by low battery life, initial battery voltage was recorded such that batteries could be recharged to their initial voltage values between data collection sets.

Experimental Procedure

To establish a map of reference values, the signal strength received by the blind node (user tag) from each of the 4 access points was measured at the center of each zone 4 orientations (N, S, E, W). Location data was gathered for the blind node for two sets of measurements, one with the blind node placed at the center of each zone in differing orientations, and one with the tag placed in different orientations at varying off-center points throughout the grid. Results were tabulated in Excel using both the Simple Point Matching and KNN algorithms to determine how closely experimental values followed expected values.

Results

Simple Point Matching Accuracy

To determine relative error, the average error distance was calculated as follows:

$$\overline{\boldsymbol{Error}} = \sqrt{\left(\boldsymbol{X_0} - \boldsymbol{X_{ref}}\right)^2 + \left(\boldsymbol{Y_o} - \boldsymbol{Y_{ref}}\right)^2}$$

In simple point matching, a comparison determined which set of reference values most closely approximated the actual values. If actual values were within the noise margin for more than one zone, the noise margin was halved successively until one zone was isolated. Using this algorithm, zone accuracy was significantly less than desirable, and any similarities were most

likely due to random variations. Mean error distance for all measurements on center was 13.63 ft. The results for simple point matching are detailed in Appendix A-i.

KNN Accuracy

The KNN method yielded higher accuracy than the SPM method, but did not achieve the original design requirement of 3 feet. The error distance was 8.81 ft for on-center measurements and 8.17 ft for off-center measurements. The mean error for on center measurements was 27.30% in x and 38.65% in y and 36.44% in x and 27.84% in y for off center measurements. Results for KNN testing are detailed in Appendix A-ii

Analysis / Discussion

Neither algorithm for position estimation delivered the accuracy desired (±4.24 mean error distance maximum). Signal strength variation over time causes inverse square relationship deviations between signal strength and distance within the test environment. Possible causes include limitations of the XBee RF module, and/or changes in interference patterns over time, factors which were subsequently investigated by testing the system in a low-noise environment.

Anechoic Chamber Testing

In order to investigate the root cause of high errors in detection, a follow up test was performed in an anechoic chamber with high-RF shielding. If it could be proven that the inverse square law between signal strength and distance held in a shielded environment, it would indicate that the high errors in position estimation found outside the chamber are due to RF interference. Conversely, if chamber data continued to deviate from expected values, it would indicate an issue with the hardware, most likely due to the transmission capabilities of the X-Bee RF module or the antenna orientation.

Experimental Setup and Procedure

Two sets of tests were performed, one using an XBee transceiver to measure signal strength and another using a dipole antenna connected via coax cable to an HP8593A spectrum analyzer. The procedure was similar to the distance v. signal determination performed in a non-shielded environment with receivers operating at the highest transmit power. Care was taken to ensure that antennas were oriented in the same vertical position. For the first test, signals were acquired at 1 ft intervals from 3 ft to 15 ft from an RF source. Additionally, 30 signal strength measurements were taken at a fixed location of 10 ft from the source to determine repeatability of an individual reading. For the second test, signals were acquired at 1 ft intervals from 1 ft to 15 ft from the source transceiver using the dipole antenna.

Results

Using the conversion from pulse width to dBm (decibels milliwatts), differences between subsequent dBm values were compared to expected values which followed the inverse square law relationship. Analyzing the data from the first set of anechoic chamber measurements, no clear relationship between signal strength and distance was discernible, as shown in the figure below.

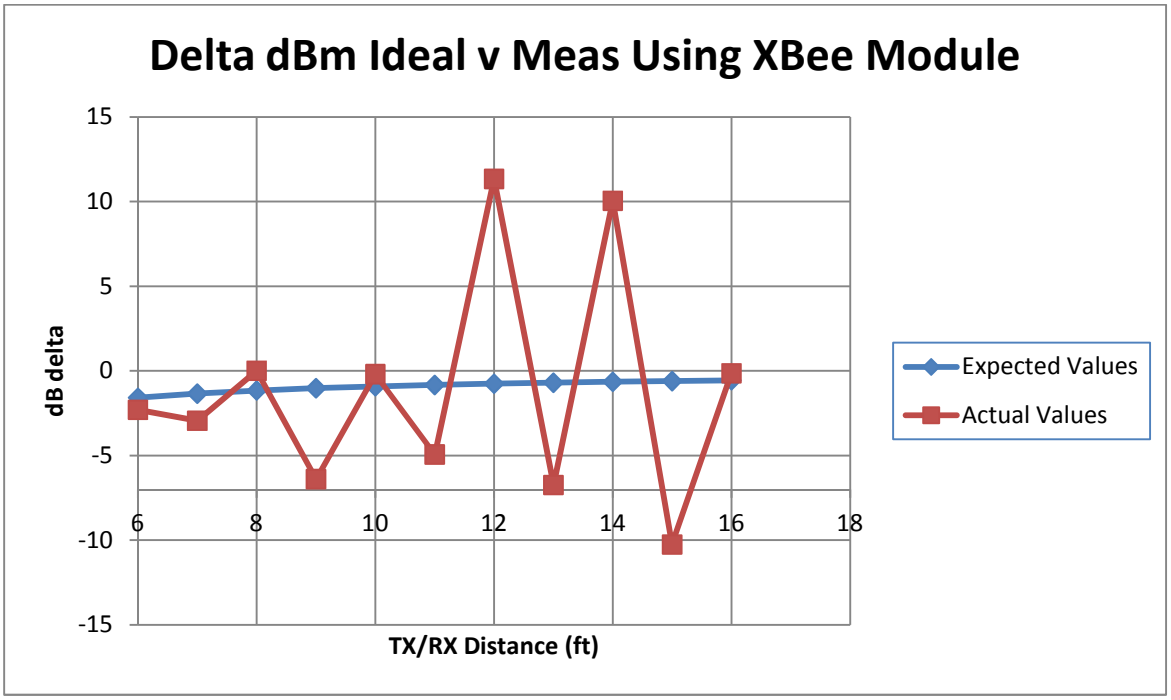

Figure 18: Friis Formula Test Shielded Environment

Surprisingly, the signal variability was also higher at a fixed point than in the test within a non-shielded environment. This indicates that either the RF shielding is not adequate or that high frequency RF interference is not the cause of unexpected variations in signal strength over time and distance. The results of the spectrum analyzer test provided below confirmed that signal strength did not vary with the inverse square of distance, as shown in figure 19.

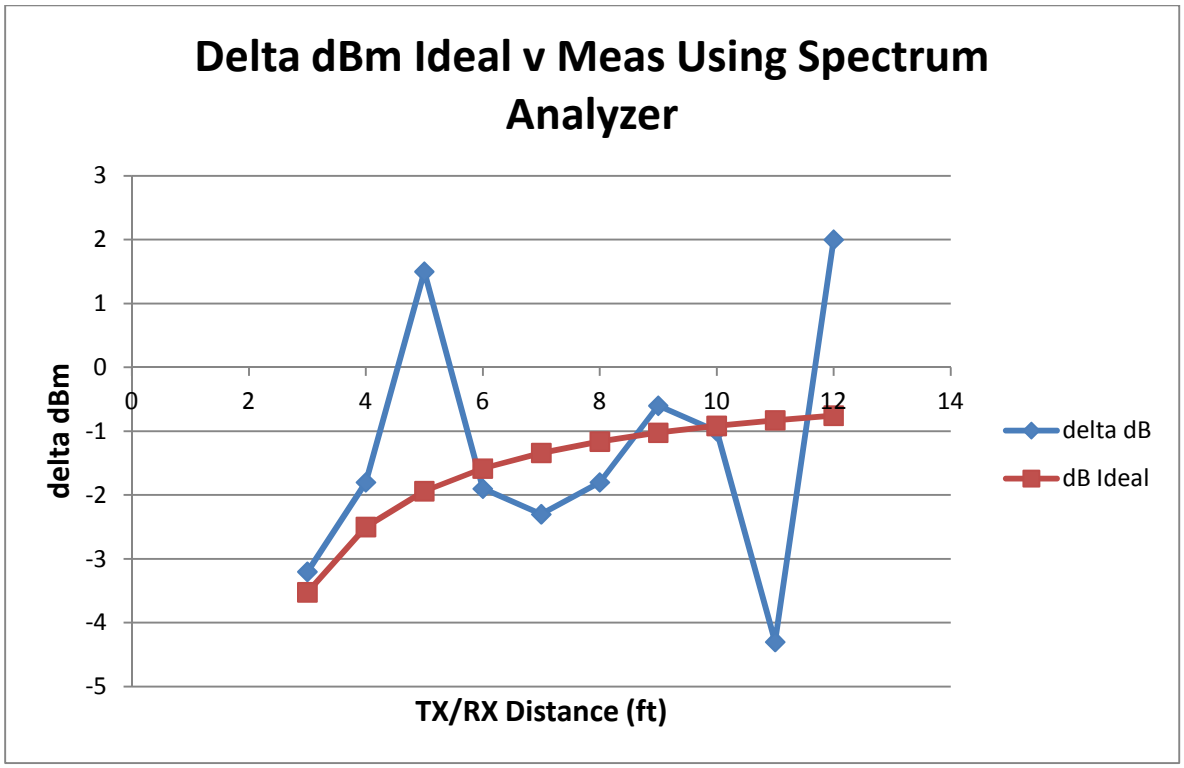

Figure 19: Friis Formula Test using Spectrum Analyzer

This ruled out the possibility that the XBee receiver was incorrectly interpreting received power readings. Rather, the root cause was estimated to be related to an anisotropic antenna footprint inherent to the XBee transmitter architecture. Due to its non-uniform footprint, the Xbee is not a reliable means of accurate position estimation. This conclusion was corroborated in a discussion with support personnel at the XBee manufacturer Digi, who explained that the RF module signal strength readings are not intended for location determination, but are rather a rough indicator of link quality between nodes within a mesh network.

Several modifications were implemented to determine if the XBee antenna transmission characteristics could be improved. It was noted that the antenna does not possess a ground plane, contrary to recommended antenna design. Therefore, a set of ground wires were added to the antenna to see if clearer readings could be obtained. Due to the sensitive nature of the

antenna and less than ideal soldering equipment, the received signal strength was severely attenuated with addition of ground wires. The rudimentary wire antenna was replaced with a more robust UFL antenna; unfortunately data transmission between source and receiver was intermittent, with RSSI readings attenuated with respect to readings obtained using the wire antenna.

Zone Tracking Test

Despite its inferior antenna design and deviations from the Friis formula, results from the previous tests performed in the QL+ lab and the anechoic chamber indicate that there is a discernible difference in signal strength between readings within a few feet of a fixed node (near zone) and beyond this limit (far zone), as shown in figure 20.

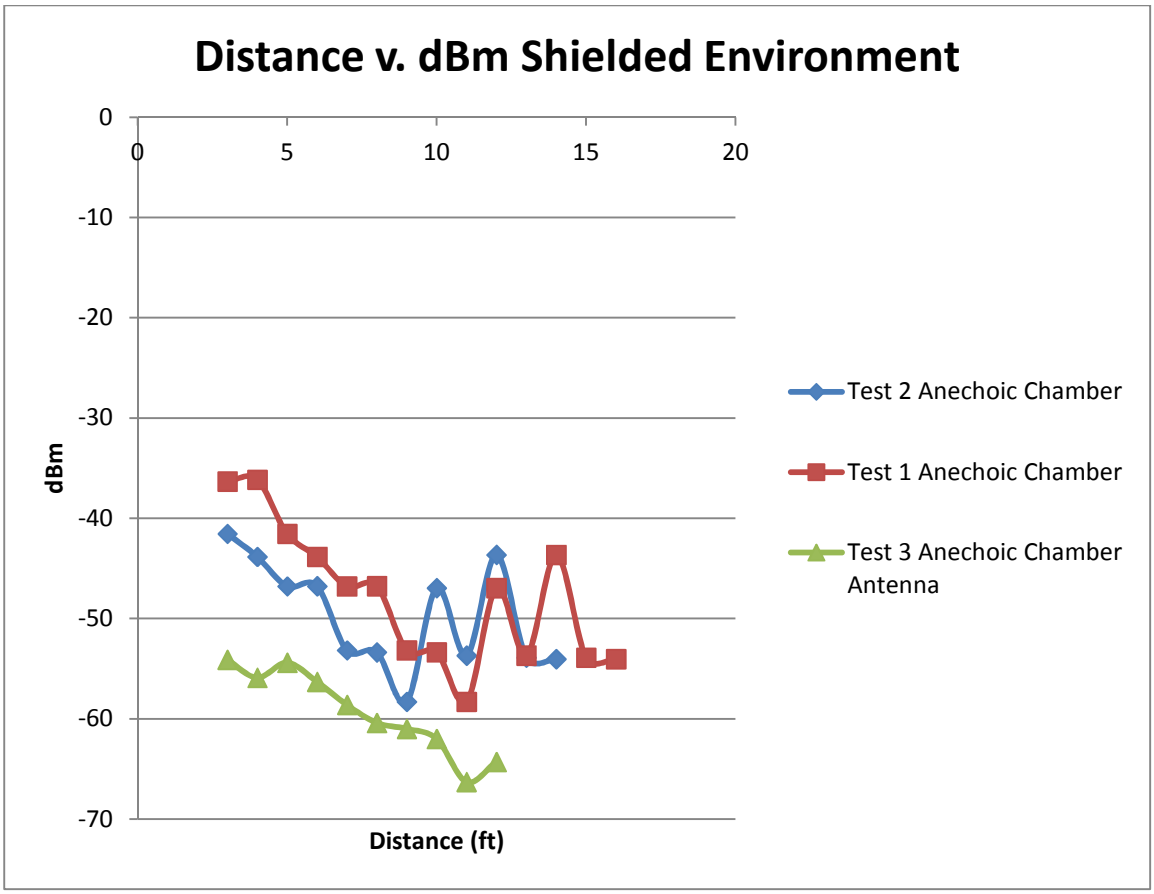

Figure 20: Distance v. RSSI Unshielded Environment

Given this cutoff it was posited that the XBee system could still serve to distinguish the location of a subject within a few meters. This meets the system requirements because it is still adequate for determining within which room a fall occurs.

Experimental Setup and Procedure

Two sets of data were acquired, one for a 15'X15' space and the second for a 30'X15' space. The experimental setup for each test was identical to the location tracking tests

for KNN and SPM algorithms, with 4 fixed nodes and 9 zones. The software design involved the use of a universal threshold RSSI value to distinguish whether the blind node fell within the near or far zone of each fixed node. For instance, if only the RSSI readings from node 1 exceeded the threshold, the blind node was assigned to zone 1. 9 sets of measurements were acquired for each of 9 possible zones, with 3 readings taken at each zone.

Results

For the smaller grid, the XBee blind node was tracked correctly for only 2 of the 9 zones (see Appendix A-iii). For the larger grid this accuracy increased to 5 out of 9 zones (see Appendix A-iv). Despite a high error rate, it is promising that an increase in grid area leads to an increased accuracy. If fixed nodes are placed in four different rooms, the results may provide an adequate means of rough position tracking. Other improvements could be to set a unique threshold for each node, and differentiate further into a near, mid and far zone of signal strength values, since many of the zones with overlapping signals would fall within this mid zone classification.

A test performed with only 4 coverage zones (one for each access point) yielded more accurate results. When the blind node was placed within 1 m of an access point it was assigned to the correct zone by the software almost 100% of the time, provided that the node and access point modules are facing each other. This simplified test did not involve intermediate zones where multiple signals overlap, given the small footprint of each antenna. It did prove the efficacy of using XBee for low resolution tracking on a large room to room scale.

Testing of Room to Room Monitoring

The final test using XBee modules addressed the limitations of using signal strength to predict location, instead serving a check in/check out function similar to RFID and infrared portals placed at strategic entry points throughout a house or hospital. XBee modules were placed near high traffic areas at the entrance to a room or within portals to determine patient room location. When a patient enters a room, the tag sends a signal to the computer, storing the location of the patient, and when they passed by another node, that location data would be modified and/or overwritten.

Experimental Setup and Results

To test the XBee portal configuration, four XBee access points were used within a small house environment, each placed within a room at waist height at a total of four discrete locations. The user passes between the living room, kitchen, bathroom and bedroom and back again at several speeds, and the responsiveness of the access points to user movement is observed. Whenever a user passes through the center of the kitchen where the access point is situated, a signal should be sent from the blind node to the software program indicating the blind node's position. False positives and false negatives were recorded and used to determine the locating accuracy of the algorithm.

$$S = \frac{N_{falls} - (FP + FN)}{N_{falls}} \, x \, 100\%$$

where FP represents false positives, FN represents false negatives, and N_{falls} represents total number of registered falls and S is the sensitivity of the device to false positive and false negatives.

Using this setup, the system was capable of tracking the movement of a user through a house within a few meters. The user tag was moved among each room several times, and changes in location were determined without conflicts between individual nodes which could have resulted in position ambiguity. However, the tag must be moved past each node at a reasonably slow speed (ie: typical walking speed) to allow a position change to be acknowledged. Location changes could be missed if the user is hurrying past the sensor. Given the impeded mobility of the target user group for this product, and the fact that activities are usually conducted at a leisurely pace within the target environments, errors in tracking should be kept to a minimum. At this time and given the current hardware and environmental considerations, coordinate position estimation is not achievable using XBee modules.

Fall Detection and Pedometer Testing

Experimental Setup

Two experiments were undertaken using the accelerometer. In one, the goal of the experiment is to determine if falls could be distinguished from ADLs at 90% accuracy. In the second, the goal was to determine if steps could be monitored accurately using the

accelerometer. Acceleration data from the user tag was acquired via the XStick and logged to a PC using the on-board SPI. The enclosed tag was secured to the subject's hip by means of a velcro strap.

Experimental Procedure

To set an appropriate acceleration threshold, the subject was asked to perform various ADLs including sitting, lying down supine, prone, and to one side, and walking at a slow and rapid pace. Each ADL was repeated twice and maximum and average acceleration values for each ADL were determined. Next the subject engaged in a series of unimpeded frontal falls, side falls and backwards falls onto a mattress from a stationary position. Accelerations due to impact shock and immediately following each fall were used to set a minimum acceleration threshold for fall detection. Results for calibration are provided below.

B. Accelerometer Readings for ADLs and Falls

Event	Average Acceleration	Max Acceleration
Falling Backwards	1.09	8.07
Falling Forwards	0.95	9.44
Side Fall	1.08	5.28
Lying Down Prone	0.97	3.07
Lying Down Side	1.02	2.75
Lying Down Supine	1.06	2.66
Sitting	0.98	2.03
Walking	0.99	2.19

Table 3: Accelerometer Readings

The same procedure was repeated for the calibrated tag and false-positive and false-negative values were recorded. Fall detection sensitivity was calculated according to the following formula:

$$S = \frac{N_{falls} - (FP + FN)}{N_{falls}} \; x \; 100\%$$

Results

Following calibration, there was no difficulty distinguishing falls from ADLs. Falls occur at accelerations greater than 5 g's as measured by the accelerometer, while the highest acceleration associated with ADLs was 3 g's during the act of lying down in a prone position.

Analysis / Discussion

A clear distinction can be drawn between falls where all bodily restraint is lost and fully controlled acts such as sitting and standing. The acceleration threshold between these events is great enough that fall detection sensitivity is virtually 100%. The result of the pedometer test was equally promising, with data identifying a near constant low acceleration value associated with each step.

System Validation

Once location tracking, step counting and fall detection schemes met specifications, they were integrated with software to detect and display the location of fall events and to monitor steps and calories burned. Tests were performed in the Cal Poly Engineering Building 192, Room 330 with four location nodes, a user tag, and a laptop running the Fall Alert software.

Experimental Setup

Nodes were situated within the room at 25 ft spacing. Although all tags respond to the user tag request for data, only one experiences a strong near field signal, allowing the user tag to associate with a unique zone when in close proximity of a node. The user tag was placed within a plastic enclosure and attached to the subject's hip using a Velcro strap.

Experimental Procedure

The first test verified that the user tag was associated with the correct location as the user walked casually through the room, with step counts properly logged by the software. A grid was laid out within the room, with 5 vertices. The user followed a predefined path, stopping at each vertex to determine the location detected by the Fall Alert software at each positions. An estimate of accuracy was calculated by taking the difference of the total number of readings and erroneous readings divided by the total number of readings.

The second test verified that falls could be distinguished from ADLs. The user performed the following ADLs: sitting down, walking, running, and lying down. Following this, the user tag was removed from the subject's waist and falls were simulated by dropping the tag from head height onto a soft surface. Several simulated falls were recorded from various heights to determine if the software accurately detected exceptionally large downward acceleration as distinguished from smaller magnitudes characteristic of ADLs.

Results and Analysis

The user tag location was accurately defined for all vertices; the software displayed the tag icon properly on the map, as shown in the figures below. Steps were tracked

correctly, and related distances and calorie consumption were logged to a data file. Given battery life effects the signal strength of each node and the Fresnel zone of each antenna differs slightly, not all zones were shaped identically; however, an algorithm differentiates the strongest node from the group, and consistently identifies the correct zone.

The second test proved that falls could be distinguished from ADLs given appropriate threshold acceleration for fall detection (5g). A fall alarm was displayed within the software program immediately after a fall occurred, verifying alarms could be detected quickly by the software. Falls were logged within a file, and falls were not mistaken for extra steps by the Arduino software.

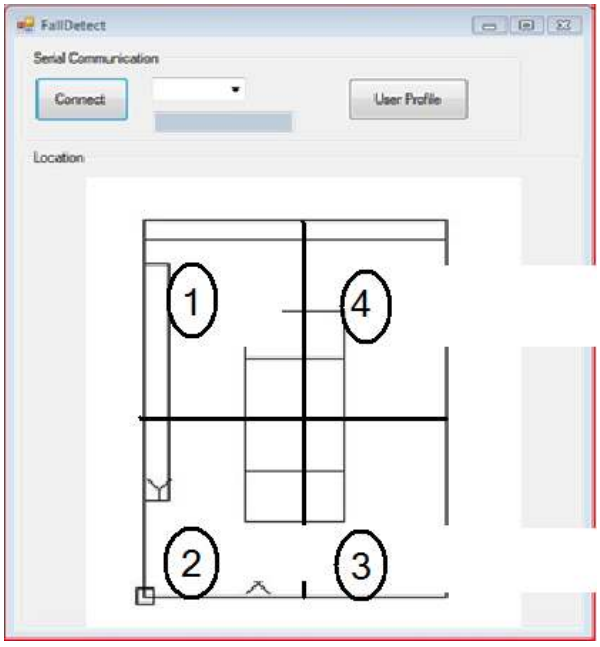

Figure 21: Software Zone Layout

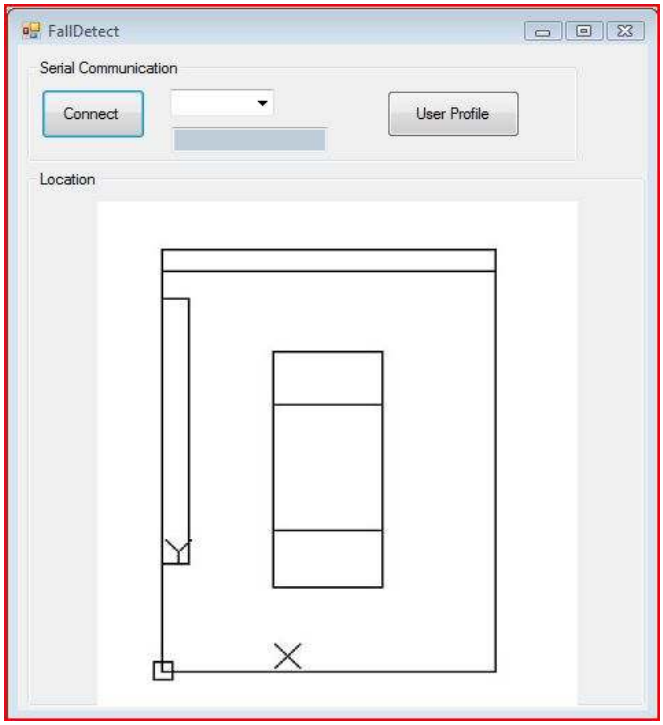

Figure 22: FallDetect Initial Screen

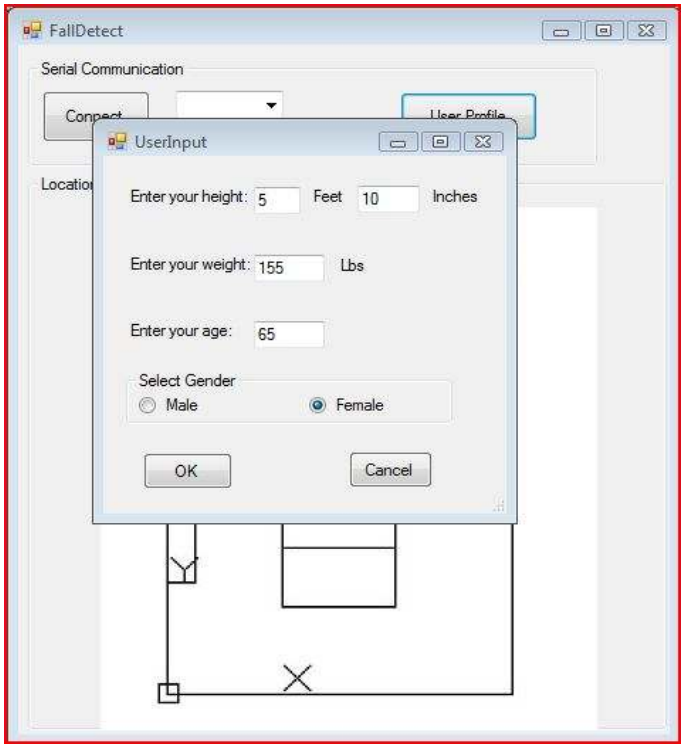

Figure 23: User Input Menu

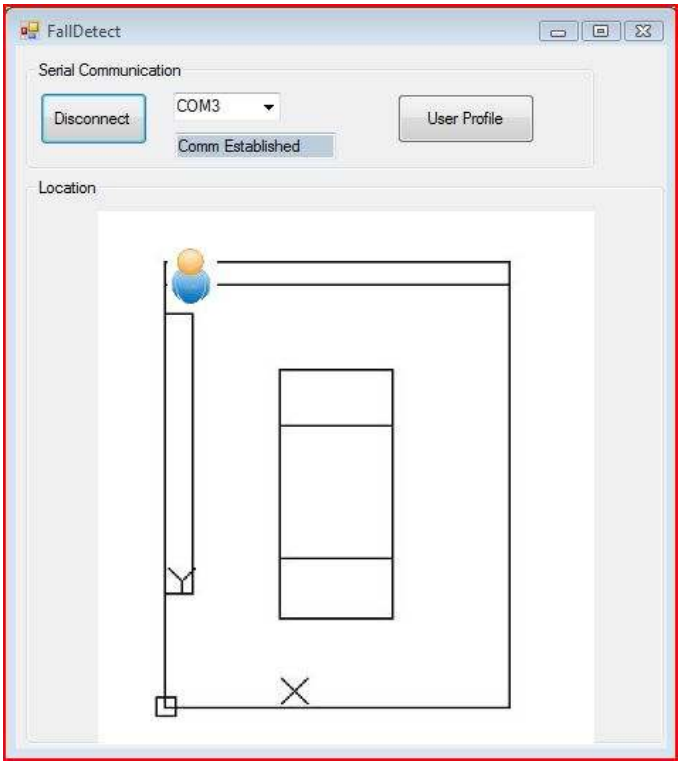

Figure 24: User Tag in Zone 1

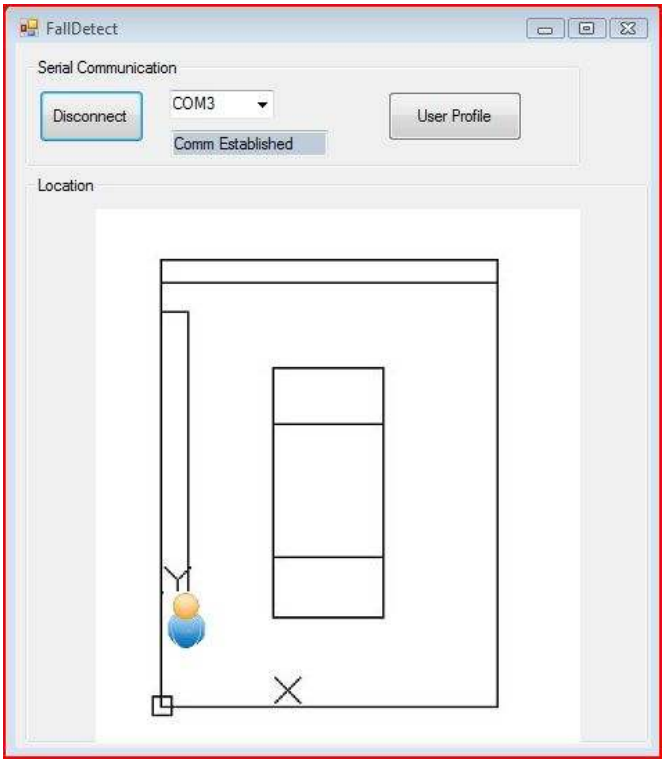

Figure 25: User Tag in Zone 2

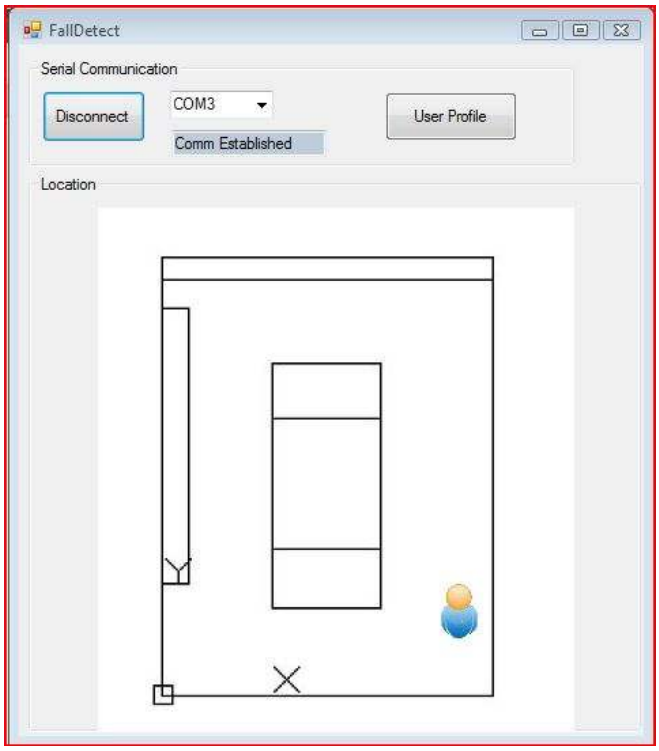

Figure 26: User Tag in Zone 3

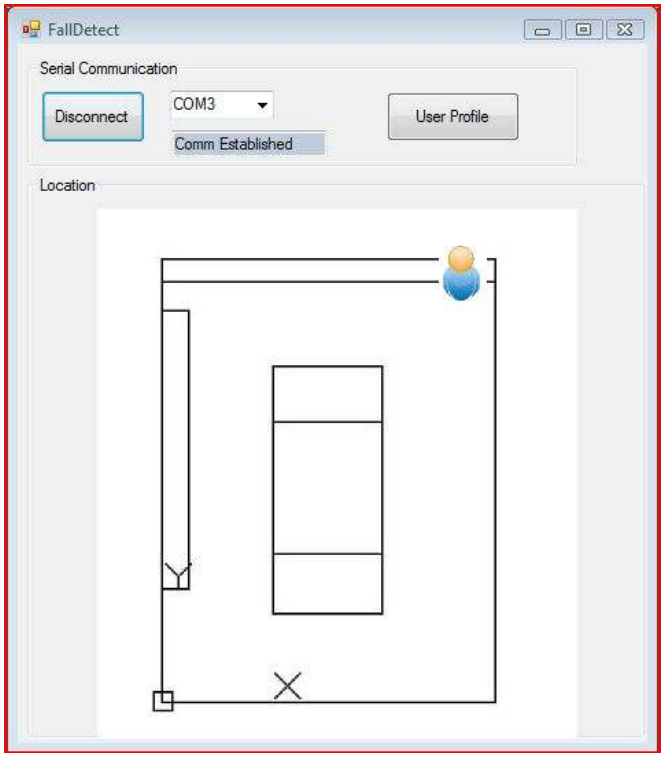

Figure 27: User Tag in Zone 4

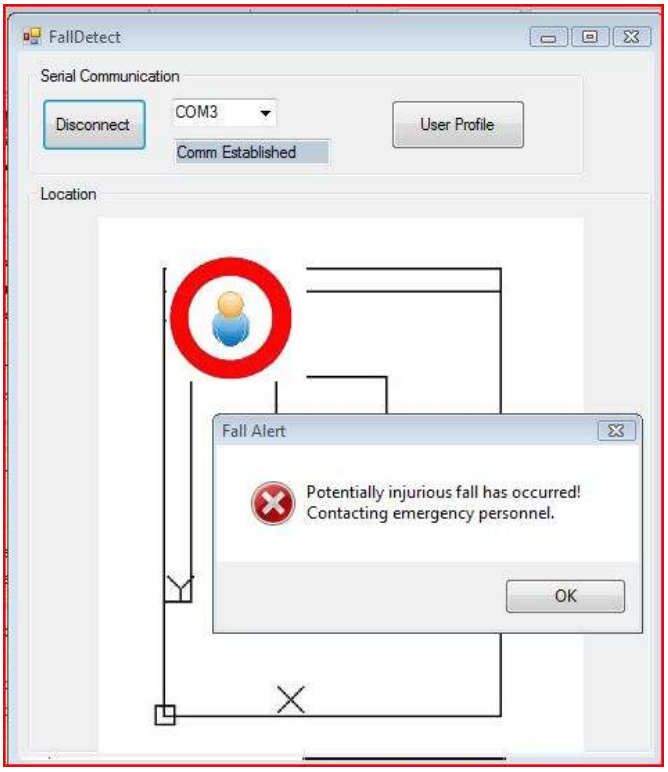

Figure 28: Fall Detected

Future Recommendations

The current system works effectively for approximate position estimation, however, an access point is needed for each location, and the software must be modified to account for new access points. As per the original aim of the project, it would be ideal if 3 locator beacons could be used to triangulate/trilaterate a 2D position within a large square area (several 1000 ft^2). Many studies have shown this to be possible given adequate hardware. In order to implement this scheme, XBee modules with high-quality antennas should be employed, since the cause of inaccurate signal strength readings was claimed to be partly due to anisotropic antenna patterns. Additionally, received signal strength has proven to be a parameter ill-suited to approximating distance. A custom application specific integrated circuit should be built to detect signal strength and convert this into a PWM signal that could be read by the Arduino.

The size of the tag is another concern that can be quickly addressed by a smaller microcontroller, such as the MSP430 from Texas Instruments. Such a change would make the user tag more inconspicuous and comfortable for the user, and avoid obstructing movement.

High power requirements hamper the current implementation, limiting access point battery life to a few hours. If the user tag possesses an infrared motion sensor to indicate when a user is in transit, the access points could be configured in low-power mode to prevent high levels of static power dissipation and awakened only when necessary.

PC software must periodically send log data and fall alerts to a remote client computer or cell phone, where the data can be interpreted and emergency personnel dispatched to the site of the accident. Software could be configured in Java to run on a mobile phone with mobile

computing capabilities and send a signal when prompted by the accelerometer. The accelerometer could even be integrated into the “smart” phone.

Although this module is primarily a fall detection device, fall prevention is the ultimate aim. With this goal in mind, it would be useful to expand the system’s breadth (assuming substantial funds) to include other types of monitors (heart rate, oxygen saturation, blood pressure) as part of a wireless body-area network similar to the European CAALYX system in order to determine the onset of pathological conditions which may lead to increased fall vulnerability. Angular sensors (ie: accelerometers) could be employed to monitor range of motion, and force sensors placed within the shoe sole could monitor muscle strength. The microcontroller would serve as the central processing hub. The challenge with integrating multiple sensors is to develop a system which does not obstruct movement, has low power requirements, and is simple to assemble and disassemble.

Though not related to a wireless body area network implementation, a smart medicine cabinet using RFID technology could be designed to correlate falls with drug intake. Such a system would record each time that a drug was taken via a reader in the cabinet and advise a physician if a patient misses a dose or overdoses on a particular medication. Each bottle of medication would be affixed with an RFID label, when the bottle is removed for medications administration, the reader would acknowledge the removal and update a registry in the PC software indicating uptake. A vibrational buzzer or audible signal on the user tag could be used to remind the user when medication should be taken. The goal of such a system would be to limit occurrences of drug-induced falls, especially where neuroleptics or bronchodilators are prescribed.

References

[1] L Jacobson, et al. “America’s Aging Population.” Population Bulletin 66, no. 1 (2011).

[2] “US Healthcare Costs: Background Briefs.” http://www.kaiseredu.org/Issue-Modules/US-Health-Care-Costs/Background-Brief.aspx. n.d. Web. 20 October, 2011.

[3] CR Baker, et al. “Wireless Sensor Networks for Home Health Care.” AINAW, Ontario, Canada, 2007.

[4] U Varshney, “Pervasive Healthcare and Wireless Health Monitoring.” Mobile Networks and Applications, Vol. 12, No. 2-3, pp. 113-127, 2007.

[5] J Dai, X Bai, Z Yang, Z Shen, D Xuan. “Mobile Phone-based Pervasive Fall Detection.” Journal of Personal and Ubiquitous Computing, 14(7):633-643, Oct. 2010.

[6] PA Stalenhoef, HFJ Crebolder, JA Knottnerus, FGE van der Horst. “Incidence, Risk Factors and Consequences of Falls Among Elderly Subjects Living in the Community: A Criteria-Based Analysis.” Eur J Public Health. 7:328-334. 1997.

[7] OP Ryynanen, SL Kivela, R Honkanen, P Laippala. “Falls and Lying Helpless in the Elderly”. Z Gerontol.; 25(4):278-82, 1992.

[8] N Noury, A Fleury, P Rumeau, AK Bourke, G ‘O Laighin, V Rialle, JE Lundy. “Fall Detection - Principles and Methods.” pg. 1663–1666. Proceedings of the 29th Annual International Conference of the IEEE EMBS, August 2007.

[9] CC Yang; YL Hsu. “A Review of Accelerometry-Based Wearable Motion Detector for Physical Activity Monitoring.” Sensors 2010, 10, 7772-7788.

[10] M Kangas, A Konttila, P Lindgren, P Winblad, T Jamsa. “Comparison of Low-Complexity Fall Detection Algorithms for Body Attached Accelerometers.” Gait & Posture, vol. 28, issue 2, pp. 285–291, 2008.

[11] A Sixsmith, N Johnson. “A Smart Sensor to Detect the Falls of the Elderly.” IEEE Pervasive Computing Magazine, pp 42–47, April–June 2004.

[12] M Alwan, PJ Rajendran, S Kell, et al. “A Smart and Passive Floor-Vibration Based Fall Detector for Elderly.” Proceedings of the 2nd IEEE International Conference on Information and Communication Technologies, held in Damascus, Syria; IEEE Press Online. pp. 1003–1007. April 24–28, 2006.

[13] J Spehr, M Gvercin, S Winkelbach, E Steinhagen-Thiessen, F Wahl. "Visual Fall Detection in Home Environments." International Conference of the International Society for Gerontechnology, Pisa, Italy, 2008.

[14] DM Bravata, C Smith-Spangler, V Sundaram, AL Gienger, N Lin, R Lewis, CD Stave, I Olkin, JR Sirard. "Using Pedometers to Increase Physical Activity and Improve Health: a Systematic Review." JAMA 298:2296–2304, 2007.

[15] D Malan, TRF Fulford-Jones, M Welsh, S Moulton. "CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care." Proc of the MobiSys 2004 Workshop on Applications of Mobile Embedded Systems (WAMES 2004) 12-14, 2004.

[16] P Fuhrer, D Guinard. "Building a Smart Hospital using RFID Technologies: Use Cases and Implementation." Proc of the 1st European Conference on eHealth (ECEH06), 2006.

[17] A Cangialosi, JE Monaly, SC Yang Jr. "Leveraging RFID In Hospitals: Patient Life Cycle and Mobility Perspectives." IEEE Communications Magazine, Volume 45, Issue 9, Sep. 2007.

[18] "Zigbee and Bluetooth: Strengths and Weaknesses for Industrial Applications." than.kaist.ac.kr/courses/2005/cs492/mailing_archive/pptO9wPB7V7CA.ppt. n.d. 12 December 2011.

[19] N Chevrollier, N Golmie. "On the Use of Wireless Technologies in Healthcare Environments." Proc. ASWN '05, Paris, France, June 2005.

[20] TR Hansen, JE Bardram, M Soegaard. "Moving Out of the Lab: Deploying Pervasive Technologies in a Hospital." IEEE Pervasive Computing, 5(3):24-31. 2006.

[21] "How Zigbee Works" http://homepage.uab.edu/cdiamond/How%20Zigbee%20Works.htm. n.d. 15 July 2011.

[22] J Jung, K Ha, J Lee, Y Kim, D Kim."Wireless Body Area Network in a Ubiquitous Healthcare System for Physiological Signal Monitoring and Health Consulting." International Journal of Signal Processing and Pattern Recognition, Vol.1, pp. 47-54, 2008.

[23] MC O'Connor. "Testing Ultrasound to Track, Monitor Patients." RFID J 1(31):2. 2006.

[24] N Priyantha, A Chakraborty, H Balakrishnan. "The Cricket location support system." Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom), Boston, MA, Aug. 2000.

[25] K Lorincz, M Welsh. "MoteTrack: A Robust, Decentralized Approach to RF-Based Location Tracking." Proceedings of the International Workshop on Location- and Context-Awareness (LoCA) at Pervasive, Oberpfaffenhofen, Germany, May 2005.

[26] R Want, A Hopper, V Falcao, J Gibbons. “The Active Badge Location System.” ACM Transactions on Information Systems, Vol. 40, No. 1, pp. 91-102, January 1992.

[27] MN Kamel Boulos, A Rocha, A Martins, ME Vicente, A Bolz, R Feld, I Tchoudovski, M Braecklein, JO Nelson, G Laighin, C Sdogati, F Cesaroni, M Antomarini, A Jobes, M Kinirons. “CAALYX: a new generation of location-based services in healthcare.” International Journal of Health Geographics 2007, 6:9.

[28] E Elnahrawy, X Li, RP Martin. “The Limits of Localization Using Signal Strength: A Comparative Study.” IEEE Sensor and Ad hoc Communications and Networks Conference (SECON), Santa Clara, CA, October 2004.

[29] V Seshadri, GV Zaruba, M Huber. “A Bayesian Sampling Approach to In-Door Localization of Wireless Devices Using Received Signal Strength Indication.” PERCOM Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications, 2005.

[30] “Awarepoint: How it Works.” http://www.awarepoint.com/our-solutions/how-it-works/. n.d Web. 16 September 2011.

[31] “Aeroscout: Technology Overview.” http://www.aeroscout.com/content/technology n.d. Web. 17 September 2011.

[32] “Ekahau Product Categories.” http://www.ekahau.com/products/products-overview.html n.d. Web. 17 September 2011.

[33] “Lifeline with Autoalert Option.” http://philips.lifelinesystems.com/content/lifeline-products/auto-alert n.d. Web. 11 October 2011.

[34] “Visonic Fall Detector Product Description.” http://www.visonic.com/Products/Wireless-Emergency-Response-Systems/Fall-detector-mct-241md-pers-wer n.d. Web. 11 October 2011.

[35] A Bueno-Cavanillas, F Padilla-Ruiz, JJ Jimenez-Moleon, CA Peinado-Alonso, R Galvez-Vargas. “Risk Factors in Falls Among the Elderly According to Extrinsic and Intrinsic Precipitating Causes.” Eur J Epidemiol;16: 849–859. 2000/

[36] AB Schultz “Mobility Impairment in the Elderly: Challenges for Biomechanics Research.” J. Biomech. 25:519–528. 1992.

Image References

[1] "UK RFID." http://ukrfid.com/what-is-rfid/ Web. 13 September 2012.

[2] "Healthcare IT News and Opinion 2/2/11." http://histalk2.com/wp-content/uploads/2011/02/awarepoint1.jpg. 2 February 2011. Web. 13 September 2012.

[3] "Jennic Product – MITS World." http://www.mitscomponent.com/product_50_Jennic%20Product.html 2008. Web. 14 September 2012.

[4] "Mobile Body Area Networks." http://thenerdinsurance.us/?tag=mobile-body-area-network
January 2011. Web. 14 September 2012.

[5] "Definition of Terms: Smart Lab." http://www.pocketvnc.com/blog/?page_id=526 Web. 14 September 2012.

[6] "Protolab ADXL3xx Accelerometers." http://protolab.pbworks.com/w/page/19403600/ADXL3xx%20Accelerometers 2007. Web. 14 September 2012.

[7] "Monitoring Your Parents' Falls – NYTimes." http://gadgetwise.blogs.nytimes.com/2010/03/18/monitoring-your-parents-falls/ March 2010. Web. 14 September 2012.

[8] "People Counting and Flow Rate Information." http://www.airport-int.com/article/flow-rate-information.html May 2010. Web. 14 September 2012.

Appendices

A. Location Tracking Tests

The first two of the following spreadsheets were used in order to determine the accuracy of the sensor network's ability to accurately track position based off of reference signal strength measurements. Two mathematical approaches were employed, one involving simple point matching and the second using a more complicated probabilistic method known as K-Nearest-Neighbors. The last two tables explore using zones associated with each node to determine relative position. While less accurate than the other approaches, they do not rely on the same correlation between distance and signal strength to deduce location.

i. SPM Calcs

SPM On-Center									
Actual	P1	P2	P3	P4	P5	P6	P7	P8	P9
X	0	7.5	15	0	7.5	15	0	7.5	15
Y	0	0	0	7.5	7.5	7.5	15	15	15
Calculated	6	6	9	5	8	7	6	2	4
X	7	15	15	7.5	7.5	0	15	7.5	0
Y	15	7.5	15	7.5	15	15	7.5	0	7.5
Error Distance	16.55	10.61	15.00	7.50	7.50	16.77	16.77	15.00	16.77
Avg Error Distance	13.61								

Table 4: Single Point Matching Calculations

ii. KNN Calcs

KNN Errors Off Center									
Actual Values	P1	P2	P3	P4	P5	P6	P7	P8	P9
X	2.8	7.8	15	11.67	1	0.42	12.67	14.83	15
Y	2.8	0	15.3	7.5	6.92	15	6.75	14.83	10.83
Calculated Values									
X	7.5	15	15	0	7.5	0	9.13402649	15	0
Y	0	0	0	7.5	15	15	13.3659735	15	7.5
Error in Distance	5.47	7.20	15.30	11.67	10.37	0.42	7.50	0.24	15.37
Error Values in X (%)	31.33	48.00	0.00	77.80	43.33	2.80	23.57	1.13	100.00
Error Values in Y (%)	18.67	0.00	102.00	0.00	53.87	0.00	44.11	1.13	22.20
Mean Error	8.17								
Mean Error in X (%)	36.44								
Mean Error in Y (%)	26.89								

Table 5: K-Nearest Neighbors Calculations (Off Center)

KNN Errors On Center									
Actual Values	P1	P2	P3	P4	P5	P6	P7	P8	P9
X	0	7.5	15	0	7.5	15	0	7.5	15
Y	0	0	0	7.5	7.5	7.5	15	15	15
Calculated Values									
X	0	0.645218	0	7.5	7.5	7.5	0	7.5	15
Y	0	14.67739	0	7.5	7.5	0	0	7.5	7.5
Error in Distance	0	16.19919	15	7.5	0	10.6066	15	7.5	7.5
Error Values in X (%)	0.00	45.70	100.00	50.00	0.00	50.00	0.00	0.00	0.00
Error Values in Y (%)	0.00	97.85	0.00	0.00	0.00	50.00	100.0	50.00	50.00
Mean Error	8.81								
Mean Error in X (%)	27.30								
Mean Error in Y (%)	38.65								

Table 6: K-Nearest Neighbors Calculations (On Center)

iii. Zone tracking 15’X15’

Orientation: West	Threshold: 38	Grid Dimensions: 15X15'								
Averages										
	Node	Zone 1	Zone 2	Zone 3	Zone 4	Zone 5	Zone 6	Zone 7	Zone 8	Zone 9
	0	50.09	28.97	27.30	35.29	35.29	38.02	28.80	28.13	29.46
	1	31.28	34.88	32.97	40.00	40.00	33.29	38.31	29.03	36.69
	2	20.94	30.81	29.30	25.95	25.95	36.88	35.92	28.35	32.21
	3	35.25	38.73	34.92	38.51	38.51	38.39	41.70	39.82	44.55
Std Dev	0	1.74	2.57	3.15	3.68	3.68	2.95	5.54	1.30	1.21
	1	1.77	2.69	0.00	1.19	1.19	6.57	0.56	0.00	0.29
	2	0.00	3.89	0.00	6.65	6.65	0.00	0.00	0.00	0.68
	3	3.99	5.33	3.05	2.16	2.16	1.05	6.24	1.02	1.17
	Zone Read	1	6	0	3,6	3,6	1,9	6,9	9	9

Table 7: Zone Tracking Accuracy 15'X15'

iv. Zone Tracking 30’X15’

Orientation : West	Threshold: 38	Grid Dimensions: 30’X15'								
Averages										
	Node	Zone 1	Zone 2	Zone 3	Zone 4	Zone 5	Zone 6	Zone 7	Zone 8	Zone 9
	0	42.21	36.35	32.61	29.78	30.88	30.03	27.14	30.93	36.11
	1	29.99	38.14	50.63	0.00	0.00	36.98	32.29	32.35	35.64
	2	22.12	25.19	25.57	26.88	22.26	22.26	34.31	35.53	31.00
	3	23.88	23.88	23.88	35.78	41.59	41.80	35.70	29.71	36.53
Std Dev	0	0.71	2.43	0.33	3.12	9.15	1.91	2.78	6.91	1.47
	1	2.26	1.32	2.43	0.00	0.00	0.07	2.97	0.00	3.41
	2	1.95	0.72	2.28	3.34	0.00	0.00	5.25	1.61	0.00
	3	0.00	0.00	0.00	1.97	0.39	0.41	2.94	2.33	0.54
	Zone Read	1	2,3	3	0,9	1,9	6	0,8	0,4,7	5,9

Table 8: Zone Tracking Accuracy 30'X15'

B. Sample Log File

The software generates a log file each time a new connection is established between the user tag and the PC via the zigbee network. In addition to detecting position and falls, it also keeps a running tally of steps taken in each zone and associated distance and calorie consumption.

3/16/2012 12:25				
Time	**Location**	**Steps**	**Distance (miles)**	**KCal**
3/16/2012 12:25	Zone1	3	0.0006877	0.068774
3/16/2012 12:25	Zone2	6	0.0013755	0.137547
3/16/2012 12:25	Zone3	7	0.0016047	0.160472
3/16/2012 12:25	Zone4	6	0.0013755	0.137547
3/16/2012 12:26	Fall Occurred	7	0.0016047	0.160472
3/16/2012 12:26	Zone1	1	0.0002292	0.022925
3/16/2012 12:26	Fall Occurred	8	0.001834	0.183396
3/16/2012 12:26	Zone3	5	0.0011462	0.114623
3/16/2012 12:26	Zone4	6	0.0013755	0.137547

Table 9: Sample Log File 3/6/2012 12:25p

C. Arduino Software Code

```
//
//        FILE: UserTag.pde
//        AUTHOR: Mike Putnam
//        PURPOSE: The code performs several functions:
//                        1) Localization of user tag with reference to access points
//                        2) Communicating localization data to PC software
//                        3) Counting number of steps taken
//                        4) Sending an indicator to PC software whenever a fall occurs


// Libraries to include

#include <ctype.h>
#include <math.h>
#include "stdlib.h"

//Add the MMA7361 Library
#include <MMA7361.h>

// Define macroprocessor directives

#define NUM_ELEM 10  //number of elements to include in accelerometer moving average

//Create the variables to be used by MMA7361 library

MMA7361 acc;

// Variable definitions for UserTag.pde

int xAccPin = A0;               //Define the pin to acquire x-acceleration data from
int yAccPin = A1;               //Define the pin to acquire y-acceleration data from
int zAccPin = A2;               //Define the pin to acquire z-acceleration data from
int zeroGDetectPin = 6;         //Define the zero g detect pin
float xOffset = 1680;           //Voltage in mV output by accelerometer along x axis at 0-g
float yOffset = 1700;           //Voltage in mV output by accelerometer along y axis at 0-g
float zOffset = 1620;           //Voltage in mV output by accelerometer along z axis at 0-g
float accVect = 0;              //Sum acceleration (combination of x, y, and z values)
float sum[NUM_ELEM];        //Acceleration array used to compute moving averages
int threshold[4] = { 30, 30, 30, 25};   //Threshold for minimum acceptable received signal
//strength
float fallAcc = 4.0;            //Minimum acceleration necessary to indicate a fall
volatile unsigned int Ticks;    //Holds the pulse count as .5 us ticks
int icrPin = 8;                 //This interrupt handler must use pin 8
int steps = 0;                  //Number of steps taken as measured by accelerometer
```

```
char flag = 0;                  //Flag used to indicate when a step has occurred
unsigned char step_bytes[2];  //PC Software requires step count in bytes
char curr_state = 0;            //Stores current location
char prev_state = 0;            //Stores previous location

float rssi_value;               //Stores received signal strenght from locator nodes
int maxRSSIVal = 0;             //Used to determine the closest locator node
char nodeMax = 0;               //Defines locator node with largest RSSI (closest to user tag)


/*****************************************************************
Function Name: Interrupt Service Routine
Arguments: Timer1 Capture Vector
Return: None
Description: Interrupt service routine used to measure the pulse width associated
with received signal strength as sampled by the XBee S1 ASIC. The interrupt flag is
triggered when a voltage change is sensed on pin 8
*****************************************************************/

ISR(TIMER1_CAPT_vect){
  if( bit_is_set(TCCR1B ,ICES1)){    // was rising edge detected ?
        TCNT1 = 0;                   // reset the counter
  }
  else {                             // falling edge was detected
        Ticks = ICR1;
  }
  TCCR1B ^= _BV(ICES1);              // toggle bit value to trigger on the other edge
}

/*****************************************************************
Function Name: Interrupt Service Routine
Arguments: Timer2 Compare Vector
Return: None
Description: Interrupt service routine used to acquire data from the
accelerometer. Triggered whenever the value in timer 2 matches the
value in compare register A
*****************************************************************/
ISR(TIMER2_COMPA_vect)
{

  accVect = movingAvg(acc.getAcc());        //get the moving average of the magnitude of
acceleration
  //Serial.println(accVect);
  if(accVect > 1.25 && accVect < 2)          //if the moving average is within range raise flag
    flag = 1;
```

```
  if(accVect < 1.05 && flag)                    //when moving average moves out of range again,
indicate that a step has occurred
  {
    steps++;
    flag = 0;
  }
  //Serial.println(steps);
  if (accVect > fallAcc)                        //has a fall occurred?
  {
    TIMSK2 &= (0<<OCIE2A);                      //if so, stop compare interrupts on timer 2
momentarily
  }
}

/*****************************************************************
Function Name: intToBytes
Arguments: integer to be converted to byte format
Return: None
Description: Takes a 16 bit integer and converts it into two 8 bit bytes
*****************************************************************/
void intToBytes(int steps)
{
  step_bytes[0] = steps & 0x00FF;
  step_bytes[1] = (steps & 0xFF00) >> 8;
}

/*****************************************************************
Function Name: movingAvg
Arguments: Acceleration vector
Return: Averaged acceleration vector
Description: Averages the acceleration value over 10 acquisitions
*****************************************************************/
float movingAvg(float accVect)
{
  float avg = 0;

  sum[0] = accVect;
  for(int i = NUM_ELEM-2; i >= 0; i--)
  {
    sum[i+1] = sum[i];
    avg += sum[i+1];
  }
  avg += sum[0];
  return avg /= NUM_ELEM;
}
```

```
/*****************************************************************
Function Name: sendToPC
Arguments: Character indicating location or fall occurence, byte array indicating number of steps
Return: None
Description: Follows established protocol for sending data to the PC software
*****************************************************************/

void sendToPC(char data, unsigned char* steps)
{
  int timeout = 0;
  Serial.flush();                          //get rid of outgoing data stream

  do
  {
    Serial.print('A');                     //initiate comm
    delay(100);
    timeout++;
    if(timeout > 30)
    {
      curr_state = 0;                      //assume comm has been reset
      return;
    }
  }
  while(Serial.read() != 'D');             //repeat sending of A until PC responds with a D
  timeout = 0;                             //reset the timeout
  delay(50);
  Serial.flush();                          //get rid of outgoing data stream
  Serial.print(data);                      //send location or fall instance

  Serial.print(steps[0]);                  //send LSB of no. of steps
  Serial.print(steps[1]);                  //send MSB of no. of steps

}

void setup()
{
  Serial.begin(9600);                      //initialize serial communication
  pinMode(zeroGDetectPin, INPUT);
  pinMode(icrPin,INPUT);                   //set Input capture pin 8 to an input
  TCCR1A = 0x00;                           // COM1A1=0, COM1A0=0 => Disconnect Pin
//OC1 from Timer/Counter 1 -- PWM11=0,PWM10=0 => PWM Operation disabled
  TCCR1B = 0x02;                           // 16MHz clock with prescaler means TCNT1
increments every .5 uS (cs11 bit set
  TCCR2A |= (1<<WGM1);                     // Clear timer 2 on compare match
```

```
  TCCR2B |= (1<<CS12);                    // 16MHz clock with 64 means TCNT1 increments
//every 4 uS
  OCR1B = 0xFFFF;                         // Trigger a compare interrupt every 0.25 seconds
  Ticks = 0;                              // Default value indicating no pulse detected

  TIMSK1 = _BV(ICIE1);                    // Enable input capture interrupt for timer 1
  TIMSK2 |= (1<<OCIE2A);                  // Enable compare interrupt for timer 2
  sei();                                  // Enable global interrupts

  //initialize accelerometer
  acc.init(xAccPin, yAccPin, zAccPin, zeroGDetectPin, xOffset, yOffset, zOffset);

  //initialize array elements to zero
  memset(sum, 0, sizeof(sum));
  memset(step_bytes, 0, sizeof(step_bytes));
}

void loop()
{
     maxRSSIVal = 0;

     //do
     {

       Serial.flush();                    //Get rid of outgoing data stream
       for(char i = 'E'; i <= 'H'; i++)   //Cycle through the four location nodes
       {
         Serial.print(i);                 //Send character to a location node
         delay(50);
         rssi_value = Ticks/2;            //Get the response's signal strength in uS
         Serial.print("RSSI value is:");
         Serial.println(rssi_value);
         rssi_value = 0;

//Determine closest node
//If the value is greater than previous values, store as the maximum
        if (rssi_value > maxRSSIVal && rssi_value > threshold[i - 'E'])
        {
          maxRSSIVal = rssi_value;
          nodeMax = i;                    //Get the letter associated with maximum node
        }
       }

       Serial.print(nodeMax);             //Send character for max node
       delay(50);
       //if(rssi_value >= THRESHOLD && Serial.read() == 'C')
```

```
      if(Serial.read() == 'C')                  //If you receive a C from the locator node, read data
      {
        curr_state = Serial.read();             //Store location data for closest locator node
      }
      else
      {
        curr_state = prev_state;                //If no node is nearby, maintain previous location
//data
      }
    }
    //while((curr_state < 49 || curr_state > 52) && curr_state != 0);
    Serial.flush();                             //Get rid of outgoing data stream
    intToBytes(steps);                          //Convert steps to byte format

    if (accVect > fallAcc)                      //Has a fall has occurred?
    {
     curr_state = '5';                          //If so, set location data to 5
     accVect = 0;                               //Reset acceleration vector
     TIMSK2 |= (1<<OCIE2A);                     //Reenable compare interrupts for timer 2
    }

    if(curr_state != prev_state)                //Is this a new location or a fall event?
    {
       steps = 0;                               //If so reset steps
       sendToPC(curr_state, step_bytes);        //Send location data plus steps in byte format to PC
software
    }

          //If current state is not a fall, set previous state. Otherwise current location does not
change
    if(curr_state != '5') prev_state = curr_state;
    else curr_state = prev_state;
}
//
//    FILE: MMA7361.cpp
//  AUTHOR: Mike Putnam
// PURPOSE: initiatilize accelerometer and convert analog voltages into number of g's

#include "MMA7361.h"
#include "WConstants.h"
#include "WProgram.h"
#include "math.h"
#include "HardwareSerial.h"
#include "inttypes.h"
#include "math.h"
#include "ctype.h"
```

```
MMA7361::MMA7361()
{}

/*****************************************************************
Function Name: getAcc
Arguments: None
Return: Magnitude of x, y and z accelerations in g's (m/s^2)
Description: Takes raw analog voltages and computes acceleration sum vector in g's
*****************************************************************/

float MMA7361::getAcc()
{
    int rxAcc, ryAcc, rzAcc;
    float xAcc = 0;
    float yAcc = 0;
    float zAcc = 0;
    float sumAcc = 0;


    rxAcc = getRawXAcc();          //get digital value for x acceleration
    ryAcc = getRawYAcc();          //get digital value for y acceleration
    rzAcc = getRawZAcc();          //get digital value for z acceleration

    //Convert digital values to g's using equation given in MMA7361 datasheet
    xAcc = (float(mapMMA7361G(getRawXAcc()) - _xOffset)) / 206;
    yAcc = (float(mapMMA7361G(getRawYAcc()) - _yOffset)) / 206;
    zAcc = (float(mapMMA7361G(getRawZAcc()) - _zOffset)) / 206;

    //return the magnitude of individual accelerations in x, y and z
    return sumAcc = sqrt(square(xAcc) + square(yAcc) + square(zAcc));
}
/*****************************************************************
Function Name: getRawXAcc
Arguments: None
Return: 10-bit digital representation of acceleration along x axis
Description: converts analog voltage to digital value using on-board ADC
*****************************************************************/
int MMA7361::getRawXAcc()
{
    return analogRead(_xAccPin);
}
/*****************************************************************
Function Name: getRawYAcc
Arguments: None
Return: 10-bit digital representation of acceleration along y axis
```

```
Description: converts analog voltage to digital value using on-board ADC
*********************************************************************/
	int MMA7361::getRawYAcc()
	{
		return analogRead(_yAccPin);
	}
/*********************************************************************
Function Name: getRawZAcc
Arguments: None
Return: 10-bit digital representation of acceleration along z axis
Description: converts analog voltage to digital value using on-board ADC
*********************************************************************/
	int MMA7361::getRawZAcc()
	{
		return analogRead(_zAccPin);
	}

/*********************************************************************
Function Name: init
Arguments: xAccPin, yAccPin, zAccPin, zeroGDetectPin, xOffset, yOffset, zOffset
Return: None
Description: specifies analog input pins where voltages are converted into 10-bit digital values
*********************************************************************/
	void MMA7361::init(int xAccPin, int yAccPin, int zAccPin, int zeroGDetectPin, int
xOffset, int yOffset, int zOffset)
	{
		pinMode(xAccPin, INPUT);
		pinMode(yAccPin, INPUT);
		pinMode(zAccPin, INPUT);
		pinMode(zeroGDetectPin, INPUT);

		_xAccPin = xAccPin;
		_yAccPin = yAccPin;
		_zAccPin = zAccPin;
		_xOffset = xOffset;
		_yOffset = yOffset;
		_zOffset = zOffset;
	}

/*********************************************************************
Function Name: mapMMA7361G
Arguments: 10 bit value to map
Return: integer representing mapped value
Description: Maps 10 bit value to a voltage in mV
*********************************************************************/
	int MMA7361::mapMMA7361G(int value)
```

```
	{
		return map(value, 0, 1024, 0, 3300);  //map 10-bit value to a voltage in mV
	}

/*****************************************************************
Function Name: getAccOffset
Arguments: none
Return: none
Description: Determines voltage offset values for (0,0,1) g values at rest
*****************************************************************/
	void MMA7361::getAccOffset()
	{
		Serial.println(float(mapMMA7361G(getRawXAcc())));
		Serial.println(float(mapMMA7361G(getRawYAcc())));
		Serial.println(float(mapMMA7361G(getRawZAcc()) - 206));
	}

//
//    FILE: MMA7361.h
//  AUTHOR: Mike Putnam
// PURPOSE: Define variables and functions for calculating acceleration using MMA7361



#ifndef MMA7361_h
#define MMA7361_h



class MMA7361
{
	public:
		MMA7361();
		void init(int xAccPin, int yAccPin, int zAccPin, int zeroGDetectPin, int xOffset,
		int yOffset, int zOffset);
		void getAccOffset();
		int getRawXAcc();
		int getRawYAcc();
		int getRawZAcc();
		float getAcc();

	private:
		int mapMMA7361G(int value);
		int _xAccPin;
		int _yAccPin;
		int _zAccPin;
```

```
        int _xOffset;
        int _yOffset;
        int _zOffset;

};

#endif
```

D. GUI Code

```
// FallDetect.cpp : main project file.

#include "stdafx.h"
#include "Mainform.h"
#include "UserInput.h"

using namespace FallDetect;

[STAThreadAttribute]
int main(array<System::String ^> ^args)
{
	// Enabling Windows XP visual effects before any controls are created
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);

	// Create the main window and run it
	Application::Run(gcnew Mainform());
	return 0;
}

//
//    FILE: Mainform.h
//	AUTHOR: Mike Putnam
//	PURPOSE: Defines main form for Fall Alert and periodic actions to be performed,
// including serial communication with user tag, display of location on a map, and
//logging of data to a csv file

#pragma once
#include <Windows.h>
#include "UserInput.h"
#include "msgBox.h"

//define macroprocessor directives which designate specific location on map
#define KITCHEN '1'
#define LIVING_ROOM '2'
#define BATHROOM '3'
#define BEDROOM '4'
#define FALL '5'

namespace FallDetect {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
```

```
	using namespace System::Data;
	using namespace System::Drawing;
	using namespace System::IO;
	using namespace System::IO::Ports;
	/// <summary>
	/// Summary for Mainform
	/// </summary>
	public ref class Mainform : public System::Windows::Forms::Form
	{
		public:
			UserInput^ UI;                                  //Declare instance
of form for inputting biometric data
			SerialPort^ _serialPort;          //Declare instance of serialport,
for communication with xStick
			SaveFileDialog^ sfd;              //Declare instance of
saveFileDialog, for logging data to PC
			DateTime^ dt;                           //Declare instance of
DateTime, for logging time to log files
			String^ file_name;
			StreamWriter^ file_stream;        //Streamwriter writes data to the
log file
			int state;                                      //stores current
location
			char go;                                        //determines whether
comm has been established with user tag
			int prev_state;                                 //stores previous
location
			int steps;                                      //stores number of
steps determined by accelerometer
			double mileage;                                 //stores mileage
calculated from number of steps

		//Mainform constructor
		Mainform(void)
		{

			InitializeComponent();
			state = 0;
			go = 0;
			prev_state = 0;
			UI = gcnew UserInput();
			mBox = gcnew msgBox();
			_serialPort = gcnew SerialPort();
			//
			//TODO: Add the constructor code here
			//
		}

	protected:
		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		~Mainform()
		{
			if (components)
			{
				delete components;
			}
```

```
		}
	private: System::Windows::Forms::Button^  buttonUserProfile;
	private: System::Windows::Forms::Panel^  panelMain;
	private: System::Windows::Forms::GroupBox^  groupBoxLocation;

	private: System::Windows::Forms::GroupBox^  groupBoxControl;
	private: System::Windows::Forms::ComboBox^  comboBoxConnect;


	private: System::Windows::Forms::Button^  buttonConnect;
	private: System::Windows::Forms::TextBox^  textBoxCommStatus;

	private: System::ComponentModel::IContainer^  components;
	private: System::Windows::Forms::PictureBox^  pictureBoxUserIcon;

	private: System::Windows::Forms::PictureBox^  pictureBoxMap;
	private: System::Windows::Forms::Timer^           timerFallDetect;

	private:
		/// <summary>
		/// Required designer variable.
		/// </summary>


#pragma region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		void InitializeComponent(void)
		{
			this->components = (gcnew System::ComponentModel::Container());
			System::ComponentModel::ComponentResourceManager^  resources =
(gcnew System::ComponentModel::ComponentResourceManager(Mainform::typeid));
			this->panelMain = (gcnew System::Windows::Forms::Panel());
			this->groupBoxLocation = (gcnew System::Windows::Forms::GroupBox());
			this->pictureBoxUserIcon = (gcnew
System::Windows::Forms::PictureBox());
			this->pictureBoxMap = (gcnew System::Windows::Forms::PictureBox());
			this->groupBoxControl = (gcnew System::Windows::Forms::GroupBox());
			this->buttonUserProfile = (gcnew System::Windows::Forms::Button());
			this->textBoxCommStatus = (gcnew System::Windows::Forms::TextBox());
			this->comboBoxConnect = (gcnew System::Windows::Forms::ComboBox());
			this->buttonConnect = (gcnew System::Windows::Forms::Button());
			this->timerFallDetect = (gcnew System::Windows::Forms::Timer(this-
>components));
			this->panelMain->SuspendLayout();
			this->groupBoxLocation->SuspendLayout();
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this-
>pictureBoxUserIcon))->BeginInit();
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this-
>pictureBoxMap))->BeginInit();
			this->groupBoxControl->SuspendLayout();
			this->SuspendLayout();
			//
			// panelMain
			//
			this->panelMain->Controls->Add(this->groupBoxLocation);
```

```
                    this->panelMain->Controls->Add(this->groupBoxControl);
                    this->panelMain->Location = System::Drawing::Point(2, 3);
                    this->panelMain->Name = L"panelMain";
                    this->panelMain->Size = System::Drawing::Size(457, 508);
                    this->panelMain->TabIndex = 0;
                    //
                    // groupBoxLocation
                    //
                    this->groupBoxLocation->Controls->Add(this->pictureBoxUserIcon);
                    this->groupBoxLocation->Controls->Add(this->pictureBoxMap);
                    this->groupBoxLocation->Location = System::Drawing::Point(5, 93);
                    this->groupBoxLocation->Name = L"groupBoxLocation";
                    this->groupBoxLocation->Size = System::Drawing::Size(450, 410);
                    this->groupBoxLocation->TabIndex = 1;
                    this->groupBoxLocation->TabStop = false;
                    this->groupBoxLocation->Text = L"Location";
                    //
                    // pictureBoxUserIcon
                    //
                    this->pictureBoxUserIcon->BackColor = System::Drawing::Color::White;
                    this->pictureBoxUserIcon->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Center;
                    this->pictureBoxUserIcon->Image =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"pictureBoxUserIcon.Image")));
                    this->pictureBoxUserIcon->Location = System::Drawing::Point(105,
304);
                    this->pictureBoxUserIcon->Name = L"pictureBoxUserIcon";
                    this->pictureBoxUserIcon->Size = System::Drawing::Size(75, 73);
                    this->pictureBoxUserIcon->TabIndex = 1;
                    this->pictureBoxUserIcon->TabStop = false;
                    //
                    // pictureBoxMap
                    //
                    this->pictureBoxMap->Enabled = false;
                    this->pictureBoxMap->Image = (cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"pictureBoxMap.Image")));
                    this->pictureBoxMap->Location = System::Drawing::Point(51, 24);
                    this->pictureBoxMap->Name = L"pictureBoxMap";
                    this->pictureBoxMap->Size = System::Drawing::Size(350, 388);
                    this->pictureBoxMap->TabIndex = 0;
                    this->pictureBoxMap->TabStop = false;
                    //
                    // groupBoxControl
                    //
                    this->groupBoxControl->Controls->Add(this->buttonUserProfile);
                    this->groupBoxControl->Controls->Add(this->textBoxCommStatus);
                    this->groupBoxControl->Controls->Add(this->comboBoxConnect);
                    this->groupBoxControl->Controls->Add(this->buttonConnect);
                    this->groupBoxControl->Location = System::Drawing::Point(5, 8);
                    this->groupBoxControl->Name = L"groupBoxControl";
                    this->groupBoxControl->Size = System::Drawing::Size(401, 79);
                    this->groupBoxControl->TabIndex = 0;
                    this->groupBoxControl->TabStop = false;
                    this->groupBoxControl->Text = L"Serial Communication";
                    //
                    // buttonUserProfile
                    //
```

```
                    this->buttonUserProfile->Location = System::Drawing::Point(262, 24);
                    this->buttonUserProfile->Name = L"buttonUserProfile";
                    this->buttonUserProfile->Size = System::Drawing::Size(97, 36);
                    this->buttonUserProfile->TabIndex = 3;
                    this->buttonUserProfile->Text = L"User Profile";
                    this->buttonUserProfile->UseVisualStyleBackColor = true;
                    this->buttonUserProfile->Click += gcnew System::EventHandler(this,
&Mainform::buttonUserProfile_Click);
                    //
                    // textBoxCommStatus
                    //
                    this->textBoxCommStatus->BackColor =
System::Drawing::SystemColors::InactiveCaption;
                    this->textBoxCommStatus->Location = System::Drawing::Point(105, 52);
                    this->textBoxCommStatus->Name = L"textBoxCommStatus";
                    this->textBoxCommStatus->ReadOnly = true;
                    this->textBoxCommStatus->Size = System::Drawing::Size(114, 20);
                    this->textBoxCommStatus->TabIndex = 2;
                    //
                    // comboBoxConnect
                    //
                    this->comboBoxConnect->FormattingEnabled = true;
                    this->comboBoxConnect->Location = System::Drawing::Point(104, 23);
                    this->comboBoxConnect->Name = L"comboBoxConnect";
                    this->comboBoxConnect->Size = System::Drawing::Size(76, 21);
                    this->comboBoxConnect->TabIndex = 1;
                    //
                    // buttonConnect
                    //
                    this->buttonConnect->Location = System::Drawing::Point(10, 23);
                    this->buttonConnect->Name = L"buttonConnect";
                    this->buttonConnect->Size = System::Drawing::Size(76, 38);
                    this->buttonConnect->TabIndex = 0;
                    this->buttonConnect->Text = L"Connect";
                    this->buttonConnect->UseVisualStyleBackColor = true;
                    this->buttonConnect->Click += gcnew System::EventHandler(this,
&Mainform::buttonConnect_Click);
                    //
                    // timerFallDetect
                    //
                    this->timerFallDetect->Interval = 500;
                    this->timerFallDetect->Tick += gcnew System::EventHandler(this,
&Mainform::timerFallDetect_Tick);
                    //
                    // Mainform
                    //
                    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                    this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                    this->ClientSize = System::Drawing::Size(459, 509);
                    this->Controls->Add(this->panelMain);
                    this->Name = L"Mainform";
                    this->Text = L"FallDetect";
                    this->Load += gcnew System::EventHandler(this,
&Mainform::Mainform_Load);
                    this->panelMain->ResumeLayout(false);
                    this->groupBoxLocation->ResumeLayout(false);
                    (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this-
>pictureBoxUserIcon))->EndInit();
```

```
                    (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this-
>pictureBoxMap))->EndInit();
                    this->groupBoxControl->ResumeLayout(false);
                    this->groupBoxControl->PerformLayout();
                    this->ResumeLayout(false);

             }
#pragma endregion

             /*****************************************************************
             Function Name: Mainform_Load
             Arguments: Object, EventArgs^ e
             Return: none
             Description: When program loads, determine which ports are available and
display them
             *****************************************************************/

             private: System::Void Mainform_Load(System::Object^  sender,
System::EventArgs^  e)
             {
                    pictureBoxUserIcon->Visible = false;
                    array<String^>^ serialPorts = SerialPort::GetPortNames();
                    for each(String^ port in serialPorts)
                    {
                           comboBoxConnect->Items->Add(port);
                    }
             }

             /*****************************************************************
             Function Name: buttonConnect_Click
             Arguments: Object, EventArgs^ e
             Return: none
             Description: When connecting, determine if userprofile has been filled out,
             and open savefiledialog box to specify where to save log data
             *****************************************************************/

             private: System::Void buttonConnect_Click(System::Object^  sender,
System::EventArgs^  e)
             {
                    //if user profile has been filled out, make sure data is entered
before connection
                    if(!lookUpUserProfileValues())
                    {
                           MessageBox::Show("Please enter user profile information before
connecting", "Fall Alert");
                    }
                    else
                    {
                           if(buttonConnect->Text == "Connect")
                           {
                                  sfd = gcnew SaveFileDialog();            //create
savefiledialog
                                  dt = gcnew DateTime();
       //timestamp to represent filename
                                  sfd->Filter = L"CSV Files (*.csv)|*.csv|"
                                                L"TXT Files (*.txt)|*.txt|"
                                                L"All Files|";
                                  sfd->FilterIndex = 1;
```

```
                                while (sfd->FileName == "")
                                {
                                       file_name = dt->Now + ".csv";      //define
filename as timestamp
                                       file_name = file_name->Replace(':','_');
                                       file_name = file_name->Replace('/','_');
                                       sfd->FileName = file_name;
                                       sfd->ShowDialog();                         //open
savefiledialog
                                }

                                file_name = sfd->FileName;
                                file_stream = gcnew StreamWriter(file_name);
                                //write headers to log file
                                file_stream->Write(dt->Now + "\n" + "Time, Location,
Steps, Distance, KCal\n");
                                file_stream->Close();

                                try
                                {
                                       //set properties for serial port and open it
                                       if(comboBoxConnect->SelectedItem == nullptr)
throw 1;

                                       buttonConnect->Text = "Disconnect";
                                       pictureBoxUserIcon->Visible = true;
       //show the user icon
                                       timerFallDetect->Enabled = true;
       //enable the main timer
                                       _serialPort->PortName = comboBoxConnect-
>SelectedItem->ToString();

                                       _serialPort->BaudRate = 9600;
                                       _serialPort->ReadTimeout = 2000;
                                       _serialPort->WriteTimeout = 2000;
                                       _serialPort->Open();

                                }
                                catch(int i)
                                {
                                       switch(i)
                                       {
                                              case 1:
                                              {
                                                     textBoxCommStatus->Text = "No port
selected";
                                                     break;
                                              }
                                              default:
                                              {
                                                     textBoxCommStatus->Text = "Comm
error";
                                                     break;
                                              }
                                       }
                                }
                         }
                         else
                         {
```

```
                                //clean up instantiated objects and close the serial
port
                                buttonConnect->Text = "Connect";
                                timerFallDetect->Enabled = false;
                                _serialPort->Close();
                                delete dt;
                                delete sfd;
                        }
                }
        }

        /*****************************************************************
        Function Name: timerFallDetect_Tick
        Arguments: Object, EventArgs^ e
        Return: none
        Description: Main loop of the program occuring every 0.5sec; establishes
        serial communication with user tag, from which the current location of
        the tag is determined and displayed on the map, and current steps and
        related data are logged to a log file everytime that location changes
        *****************************************************************/
        private: System::Void timerFallDetect_Tick(System::Object^  sender,
System::EventArgs^  e)
        {
                array<unsigned char>^ buffer = {'D'};
                array<byte>^ steps_bytes = {0,0};
                //array<Char^> ^input = {0xAD, 0x01};
                try
                {
                        go = _serialPort->ReadChar();                   //read
incoming data from the user tag
                        _serialPort->DiscardInBuffer();

                        if(go == 'A')                                     //if
incoming character is an A begin send procedure
                        {
                                _serialPort->Write(buffer, 0, 1);  //let blind node
know that an A has been received
                                Sleep(50);
                                _serialPort->DiscardInBuffer();
                                Sleep(50);
                                state = _serialPort->ReadChar();   //read the current
position of the blind node

                                textBoxCommStatus->Text = "Comm Established";
                                //if for some reason a character other than 1-5 is
sent, keep the previous character
                                if(!((state >= 49 && state <= 53) || state == 0))
                                        state = prev_state;

                                if(state != prev_state)
                                {
                                        //write the current date to the text file
                                        file_stream = File::AppendText(file_name);
                                        file_stream->Write(dt->Now + ",");
                                        //specify what location to display the user tag
at, or if a fall has occurred
                                        //engage appropriate routine
                                        switch(state)
```

```
{
	case KITCHEN:
	{
		pictureBoxUserIcon->Location =
Point(50,50);
		file_stream->Write("Kitchen,");
		break;
	}
	case LIVING_ROOM:
	{
		pictureBoxUserIcon->Location =
Point(50,300);
		file_stream->Write("Living
Room,");
		break;
	}
	case BATHROOM:
	{
		pictureBoxUserIcon->Location =
Point(250,300);
		file_stream->Write("Bathroom,");
		break;
	}
	case BEDROOM:
	{
		pictureBoxUserIcon->Location =
Point(250,50);
		file_stream->Write("Bedroom,");
		break;
	}
	case FALL:
	{
		//fall routine, currently displays
a message box, in future iterations
		//should send fall alert to a
remote client PC or smart phone
		file_stream->Write("Fall
Occurred,");
		pictureBoxUserIcon->Image =
Image::FromFile(L"C:\\Users\\Mike\\Pictures\\Fall.png");
		timerFallDetect->Enabled = false;

		MessageBox::Show("Potentially
injurious fall has occurred! \nContacting emergency personnel.", "Fall Alert",
MessageBoxButtons::OK, MessageBoxIcon::Stop);
		pictureBoxUserIcon->Image =
Image::FromFile(L"C:\\Users\\Mike\\Pictures\\UserIcon.png");
		timerFallDetect->Enabled = true;

		break;

	}

	default:
	{
		pictureBoxUserIcon->Location =
Point(0,0);
		break;
```

```
                                        }


                                    }

                                    //send other log data, including steps, mileage,
and calories
                                    _serialPort->Read(steps_bytes, 0, 2);
                                    steps = bytesToInt(steps_bytes);
                                    mileage = getMiles();
                                    file_stream->Write(Convert::ToString(steps) +
",");
                                    file_stream->Write(Convert::ToString(mileage) +
",");
                                    file_stream-
>WriteLine(Convert::ToString(getKCal()));
                                    prev_state = state;
                                    file_stream->Close();
                                    _serialPort->Write(buffer, 0, 1);
                                }


                        }
                }


                catch(TimeoutException^ ex)
                {
                        textBoxCommStatus->Text = "Waiting for data.";
                }
        }
        /*****************************************************************
        Function Name: buttonUserProfile_Click
        Arguments: Object, EventArgs^ e
        Return: none
        Description: Activate the UserProfile form
        *****************************************************************/
        private: System::Void buttonUserProfile_Click(System::Object^  sender,
System::EventArgs^  e)
        {

                UI->ShowDialog();
        }
        /*****************************************************************
        Function Name: lookUpUserProfileValues
        Arguments: none
        Return: char
        Description: search the registry, if any biometric data is undefined,
        return a 0, else take that data and use it to calculate calories, mileage,
etc.
        *****************************************************************/
        private: char lookUpUserProfileValues()
        {
                RegistryKey^ currentUser;
                RegistryKey^ softwareKey;
                currentUser = Registry::CurrentUser;
                softwareKey = currentUser->
                        OpenSubKey("Software\\Fall Alert");
```

```
                    if (softwareKey->GetValue("age") == nullptr || softwareKey-
>GetValue("height") == nullptr
                           || softwareKey->GetValue("weight") == nullptr || softwareKey-
>GetValue("gender") == nullptr)
                    {
                           return 0;
                    }
                    else
                    {
                           UI->age = Convert::ToInt32(softwareKey->GetValue("age"));
                           UI->height = Convert::ToDouble(softwareKey-
>GetValue("height"));
                           UI->weight = Convert::ToDouble(softwareKey-
>GetValue("weight"));
                           UI->gender = Convert::ToBoolean(softwareKey-
>GetValue("gender"));
                           return 1;
                    }
             }
             /*****************************************************************
             Function Name: bytesToInt
             Arguments: none
             Return: int
             Description: Converts two 8 bit bytes into a 16 bit integer
             *****************************************************************/
             private: int bytesToInt(array<unsigned char>^ input)
             {
                    int output = 0;
                    for(int i = 1; i>=0; i--)
                    {
                           output = (output << 8) + input[i];
                    }
                    return output;
             }
             /*****************************************************************
             Function Name: getMiles
             Arguments: none
             Return: double
             Description: determine mileage based on steps taken
             *****************************************************************/
             private: double getMiles()
             {
                    double miles;
                    if(UI->gender)
                           miles = steps*(0.415*UI->height/2)/160934.4;
                    else
                           miles = steps*(0.413*UI->height/2)/160934.4;

                    return miles;
             }
             /*****************************************************************
             Function Name: getKCal
             Arguments: none
             Return: double
             Description: determine KCal based on mileage, assuming 1 mile walking = 100
             Calories
             *****************************************************************/
             private: double getKCal()
```

```
		{
			return 100*mileage;
		}
};

}

//
//    FILE: UserInput.h
//       AUTHOR: Mike Putnam
//       PURPOSE: Defines user input sub form for Fall Alert, which takes biological data
//(age, gender, height, and weight) and stores it in the registry for use in calculating
//calories and mileage in the main form

#pragma once

namespace FallDetect {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	using namespace Microsoft::Win32;
	/// <summary>
	/// Summary for UserInput
	/// </summary>
	public ref class UserInput : public System::Windows::Forms::Form
	{

	public:

		double height;
		double weight;
		int age;
		bool gender;
		//registry key instantiations for accessing the PC registry
		RegistryKey^ currentUser;
		RegistryKey^ softwareKey;


		UserInput(void)
		{
			InitializeComponent();
			//
			//TODO: Add the constructor code here
			//
		}

	protected:
		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		~UserInput()
		{
			if (components)
			{
```

```
                        delete components;
                    }
                }
        private: System::Windows::Forms::Panel^  panelUserInput;
        private: System::Windows::Forms::Label^  labelHeight;
        private: System::Windows::Forms::Label^  labelLbs;
        private: System::Windows::Forms::TextBox^  textBoxLbs;
        private: System::Windows::Forms::Label^  labelWeight;
        private: System::Windows::Forms::Label^  labelInches;
        private: System::Windows::Forms::Label^  labelFeet;
        private: System::Windows::Forms::TextBox^  textBoxInches;
        private: System::Windows::Forms::TextBox^  textBoxFeet;
        private: System::Windows::Forms::TextBox^  textBoxAge;
        private: System::Windows::Forms::Label^  labelAge;
        private: System::Windows::Forms::Button^  buttonUICancel;
        private: System::Windows::Forms::Button^  buttonUIOK;
        private: System::Windows::Forms::GroupBox^  groupBoxGender;
        private: System::Windows::Forms::RadioButton^  radioButtonFemale;
        private: System::Windows::Forms::RadioButton^  radioButtonMale;

        private:
                /// <summary>
                /// Required designer variable.
                /// </summary>
                System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                void InitializeComponent(void)
                {
                        this->panelUserInput = (gcnew System::Windows::Forms::Panel());
                        this->groupBoxGender = (gcnew System::Windows::Forms::GroupBox());
                        this->radioButtonFemale = (gcnew
System::Windows::Forms::RadioButton());
                        this->radioButtonMale = (gcnew
System::Windows::Forms::RadioButton());
                        this->textBoxAge = (gcnew System::Windows::Forms::TextBox());
                        this->labelAge = (gcnew System::Windows::Forms::Label());
                        this->buttonUICancel = (gcnew System::Windows::Forms::Button());
                        this->buttonUIOK = (gcnew System::Windows::Forms::Button());
                        this->labelLbs = (gcnew System::Windows::Forms::Label());
                        this->textBoxLbs = (gcnew System::Windows::Forms::TextBox());
                        this->labelWeight = (gcnew System::Windows::Forms::Label());
                        this->labelInches = (gcnew System::Windows::Forms::Label());
                        this->labelFeet = (gcnew System::Windows::Forms::Label());
                        this->textBoxInches = (gcnew System::Windows::Forms::TextBox());
                        this->textBoxFeet = (gcnew System::Windows::Forms::TextBox());
                        this->labelHeight = (gcnew System::Windows::Forms::Label());
                        this->panelUserInput->SuspendLayout();
                        this->groupBoxGender->SuspendLayout();
                        this->SuspendLayout();
                        //
                        // panelUserInput
                        //
                        this->panelUserInput->Controls->Add(this->groupBoxGender);
```

```
                    this->panelUserInput->Controls->Add(this->textBoxAge);
                    this->panelUserInput->Controls->Add(this->labelAge);
                    this->panelUserInput->Controls->Add(this->buttonUICancel);
                    this->panelUserInput->Controls->Add(this->buttonUIOK);
                    this->panelUserInput->Controls->Add(this->labelLbs);
                    this->panelUserInput->Controls->Add(this->textBoxLbs);
                    this->panelUserInput->Controls->Add(this->labelWeight);
                    this->panelUserInput->Controls->Add(this->labelInches);
                    this->panelUserInput->Controls->Add(this->labelFeet);
                    this->panelUserInput->Controls->Add(this->textBoxInches);
                    this->panelUserInput->Controls->Add(this->textBoxFeet);
                    this->panelUserInput->Controls->Add(this->labelHeight);
                    this->panelUserInput->Location = System::Drawing::Point(6, 10);
                    this->panelUserInput->Name = L"panelUserInput";
                    this->panelUserInput->Size = System::Drawing::Size(270, 251);
                    this->panelUserInput->TabIndex = 0;
                    //
                    // groupBoxGender
                    //
                    this->groupBoxGender->Controls->Add(this->radioButtonFemale);
                    this->groupBoxGender->Controls->Add(this->radioButtonMale);
                    this->groupBoxGender->Location = System::Drawing::Point(10, 149);
                    this->groupBoxGender->Name = L"groupBoxGender";
                    this->groupBoxGender->Size = System::Drawing::Size(230, 39);
                    this->groupBoxGender->TabIndex = 12;
                    this->groupBoxGender->TabStop = false;
                    this->groupBoxGender->Text = L"Select Gender";
                    //
                    // radioButtonFemale
                    //
                    this->radioButtonFemale->AutoSize = true;
                    this->radioButtonFemale->Location = System::Drawing::Point(129, 16);
                    this->radioButtonFemale->Name = L"radioButtonFemale";
                    this->radioButtonFemale->Size = System::Drawing::Size(59, 17);
                    this->radioButtonFemale->TabIndex = 1;
                    this->radioButtonFemale->TabStop = true;
                    this->radioButtonFemale->Text = L"Female";
                    this->radioButtonFemale->UseVisualStyleBackColor = true;
                    this->radioButtonFemale->CheckedChanged += gcnew
System::EventHandler(this, &UserInput::radioButtonFemale_CheckedChanged);
                    //
                    // radioButtonMale
                    //
                    this->radioButtonMale->AutoSize = true;
                    this->radioButtonMale->Location = System::Drawing::Point(9, 16);
                    this->radioButtonMale->Name = L"radioButtonMale";
                    this->radioButtonMale->Size = System::Drawing::Size(48, 17);
                    this->radioButtonMale->TabIndex = 0;
                    this->radioButtonMale->TabStop = true;
                    this->radioButtonMale->Text = L"Male";
                    this->radioButtonMale->UseVisualStyleBackColor = true;
                    this->radioButtonMale->CheckedChanged += gcnew
System::EventHandler(this, &UserInput::radioButtonMale_CheckedChanged);
                    //
                    // textBoxAge
                    //
                    this->textBoxAge->Location = System::Drawing::Point(100, 112);
                    this->textBoxAge->Name = L"textBoxAge";
```

```
                    this->textBoxAge->Size = System::Drawing::Size(50, 20);
                    this->textBoxAge->TabIndex = 11;
                    //
                    // labelAge
                    //
                    this->labelAge->AutoSize = true;
                    this->labelAge->Location = System::Drawing::Point(10, 112);
                    this->labelAge->Name = L"labelAge";
                    this->labelAge->Size = System::Drawing::Size(79, 13);
                    this->labelAge->TabIndex = 10;
                    this->labelAge->Text = L"Enter your age:";
                    //
                    // buttonUICancel
                    //
                    this->buttonUICancel->Location = System::Drawing::Point(167, 208);
                    this->buttonUICancel->Name = L"buttonUICancel";
                    this->buttonUICancel->Size = System::Drawing::Size(59, 27);
                    this->buttonUICancel->TabIndex = 9;
                    this->buttonUICancel->Text = L"Cancel";
                    this->buttonUICancel->UseVisualStyleBackColor = true;
                    this->buttonUICancel->Click += gcnew System::EventHandler(this,
&UserInput::buttonUICancel_Click);
                    //
                    // buttonUIOK
                    //
                    this->buttonUIOK->Location = System::Drawing::Point(22, 209);
                    this->buttonUIOK->Name = L"buttonUIOK";
                    this->buttonUIOK->Size = System::Drawing::Size(63, 27);
                    this->buttonUIOK->TabIndex = 8;
                    this->buttonUIOK->Text = L"OK";
                    this->buttonUIOK->UseVisualStyleBackColor = true;
                    this->buttonUIOK->Click += gcnew System::EventHandler(this,
&UserInput::buttonUIOK_Click);
                    //
                    // labelLbs
                    //
                    this->labelLbs->AutoSize = true;
                    this->labelLbs->Location = System::Drawing::Point(158, 64);
                    this->labelLbs->Name = L"labelLbs";
                    this->labelLbs->Size = System::Drawing::Size(24, 13);
                    this->labelLbs->TabIndex = 7;
                    this->labelLbs->Text = L"Lbs";
                    //
                    // textBoxLbs
                    //
                    this->textBoxLbs->Location = System::Drawing::Point(100, 63);
                    this->textBoxLbs->Name = L"textBoxLbs";
                    this->textBoxLbs->Size = System::Drawing::Size(50, 20);
                    this->textBoxLbs->TabIndex = 6;
                    //
                    // labelWeight
                    //
                    this->labelWeight->AutoSize = true;
                    this->labelWeight->Location = System::Drawing::Point(10, 63);
                    this->labelWeight->Name = L"labelWeight";
                    this->labelWeight->Size = System::Drawing::Size(92, 13);
                    this->labelWeight->TabIndex = 5;
                    this->labelWeight->Text = L"Enter your weight:";
```

```
                    //
                    // labelInches
                    //
                    this->labelInches->AutoSize = true;
                    this->labelInches->Location = System::Drawing::Point(223, 13);
                    this->labelInches->Name = L"labelInches";
                    this->labelInches->Size = System::Drawing::Size(39, 13);
                    this->labelInches->TabIndex = 4;
                    this->labelInches->Text = L"Inches";
                    //
                    // labelFeet
                    //
                    this->labelFeet->AutoSize = true;
                    this->labelFeet->Location = System::Drawing::Point(139, 13);
                    this->labelFeet->Name = L"labelFeet";
                    this->labelFeet->Size = System::Drawing::Size(28, 13);
                    this->labelFeet->TabIndex = 3;
                    this->labelFeet->Text = L"Feet";
                    //
                    // textBoxInches
                    //
                    this->textBoxInches->Location = System::Drawing::Point(173, 12);
                    this->textBoxInches->Name = L"textBoxInches";
                    this->textBoxInches->Size = System::Drawing::Size(44, 20);
                    this->textBoxInches->TabIndex = 2;
                    //
                    // textBoxFeet
                    //
                    this->textBoxFeet->Location = System::Drawing::Point(100, 13);
                    this->textBoxFeet->Name = L"textBoxFeet";
                    this->textBoxFeet->Size = System::Drawing::Size(33, 20);
                    this->textBoxFeet->TabIndex = 1;
                    //
                    // labelHeight
                    //
                    this->labelHeight->AutoSize = true;
                    this->labelHeight->Location = System::Drawing::Point(10, 13);
                    this->labelHeight->Name = L"labelHeight";
                    this->labelHeight->Size = System::Drawing::Size(90, 13);
                    this->labelHeight->TabIndex = 0;
                    this->labelHeight->Text = L"Enter your height:";
                    //
                    // UserInput
                    //
                    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                    this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                    this->ClientSize = System::Drawing::Size(284, 264);
                    this->Controls->Add(this->panelUserInput);
                    this->Name = L"UserInput";
                    this->Text = L"UserInput";
                    this->Load += gcnew System::EventHandler(this,
&UserInput::UserInput_Load);
                    this->panelUserInput->ResumeLayout(false);
                    this->panelUserInput->PerformLayout();
                    this->groupBoxGender->ResumeLayout(false);
                    this->groupBoxGender->PerformLayout();
                    this->ResumeLayout(false);
```

```
            }
#pragma endregion
            /*****************************************************************
            Function Name: buttonUIOK_Click
            Arguments: Object^, EventArgs^ e
            Return: none
            Description: Ensure that all data has been saved to the registry
            before returning to the main form
            *****************************************************************/
            private: System::Void buttonUIOK_Click(System::Object^  sender,
System::EventArgs^  e)
            {

                  if (textBoxAge->Text != "" && textBoxFeet->Text != "" &&
textBoxInches->Text != ""
                        && textBoxInches->Text != "")
                  {
                        height = getHeightCM();           //convert height in feet,
inches to cm
                        weight = getWeightKG();           //convert height in lbs to
kg
                        age = Convert::ToInt32(textBoxAge->Text);
                        try
                        {
                              //write data to the registry
                              softwareKey->SetValue("height", height);
                              softwareKey->SetValue("weight", weight);
                              softwareKey->SetValue("age", age);
                              softwareKey->SetValue("gender", gender);
                        }
                        catch(Exception^ ex)
                        {
                              MessageBox::Show(ex->Message);
                        }
                        __finally
                        {
                              if (softwareKey) softwareKey->Close();
                              if (currentUser) currentUser->Close();
                        }
                        this->Close();
                  }
                  else MessageBox::Show("Please fill in all fields before
continuing.", "Fall Alert");

            }
            /*****************************************************************
            Function Name: buttonUICancel_Click
            Arguments: Object^, EventArgs^ e
            Return: none
            Description: Clear data before closing and returning to main form
            *****************************************************************/
            private: System::Void buttonUICancel_Click(System::Object^  sender,
System::EventArgs^  e)
            {
                  textBoxFeet->Text = "";
                  textBoxInches->Text = "";
                  textBoxLbs->Text = "";
                  textBoxAge->Text = "";
```

```
            this->Close();
        }
        /*****************************************************************
        Function Name: getHeightCM
        Arguments: none
        Return: float
        Description: Convert height from text field into centimeters
        *****************************************************************/
        private: float getHeightCM()
        {
            double feet;
            double inches;

            feet = Convert::ToDouble(textBoxFeet->Text);
            inches = Convert::ToDouble(textBoxInches->Text);

            inches = (feet*12) + inches;
            return inches*2.54;
        }
        /*****************************************************************
        Function Name: setHeightFtIn
        Arguments: height in cm
        Return: none
        Description: Convert height to ft, inches to display in text fields
        *****************************************************************/
        private: void setHeightFtIn(double hgtCM)
        {
            int feet;
            int inches;

            inches = hgtCM/2.54;
            feet = inches/12;
            inches = inches - (feet*12);

            textBoxFeet->Text = Convert::ToString(feet);
            textBoxInches->Text = Convert::ToString(inches);
        }
        /*****************************************************************
        Function Name: setWeightLbs
        Arguments: weight in kg
        Return: none
        Description: Convert weight in kg to lbs to display in text field
        *****************************************************************/
        private: void setWeightLbs(double wgtKG)
        {
            textBoxLbs->Text =
Convert::ToString(int(Math::Round(wgtKG/0.454,1)));
        }
        /*****************************************************************
        Function Name: getWeightKG
        Arguments: none
        Return: float
        Description: Take weight in lbs from text field and convert to kg
        *****************************************************************/
        private: float getWeightKG()
        {
            return Convert::ToDouble(textBoxLbs->Text)*0.454;
        }
```

```
		/*****************************************************************
		Function Name: radioButtonMale_CheckedChanged
		Arguments: Object^, EventArgs^ e
		Return: none
		Description: set gender to male when radio button is selected
		*****************************************************************/

		private: System::Void radioButtonMale_CheckedChanged(System::Object^
sender, System::EventArgs^  e)
		{
			if(radioButtonMale->Checked)
				gender = 1;
		}
		/*****************************************************************
		Function Name: radioButtonFemale_CheckedChanged
		Arguments: Object^, EventArgs^ e
		Return: none
		Description: set gender to female when radio button is selected
		*****************************************************************/
		private: System::Void radioButtonFemale_CheckedChanged(System::Object^
sender, System::EventArgs^  e)
		{
			if(radioButtonFemale->Checked)
				gender = 0;
		}
		/*****************************************************************
		Function Name: UserInput_Load
		Arguments: Object^, EventArgs^ e
		Return: none
		Description: grab data from registry and display it in the form, unless it
		it is undefined
		*****************************************************************/
		private: System::Void UserInput_Load(System::Object^  sender,
System::EventArgs^  e)
		{
			currentUser = Registry::CurrentUser;
			softwareKey = currentUser->
				CreateSubKey("Software\\Fall Alert");
			if (softwareKey->GetValue("age") != nullptr && softwareKey-
>GetValue("height") != nullptr
				&& softwareKey->GetValue("weight") != nullptr && softwareKey-
>GetValue("gender") != nullptr)
			{
				textBoxAge->Text = Convert::ToString(softwareKey-
>GetValue("age"));
				gender = Convert::ToBoolean(softwareKey->GetValue("gender"));
				if(gender)  radioButtonMale->Checked = 1;
				else  radioButtonFemale->Checked = 1;
				setHeightFtIn(Convert::ToDouble(softwareKey-
>GetValue("height")));
				setWeightLbs(Convert::ToDouble(softwareKey-
>GetValue("weight")));

			}
		}
};
}
```