AN IMMERSIVE TECHNOLOGY FOR INDOOR EXERCISE EQUIPMENT

by

Abraham A. Clements

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr.YangQuan Chen Major Professor	Dr. Donald Cripps Committee Member

Dr. Jacob Gunther
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2010

Abstract

An Immersive Technology for Indoor Exercise Equipment

by

Abraham A. Clements, Master of Science

Utah State University, 2010

Major Professor: Dr.YangQuan Chen
Department: Electrical and Computer Engineering

Indoor exercise equipment is used for many reasons. The primary reason people own exercise equipment is a desire to improve their health. While exercise equipment has many advantages, such as being able to exercise during inclement weather, using a machine usually involves performing highly repetitive actions in an unchanging environment. These two factors often lead to boredom which makes it difficult for people to continue using indoor exercise equipment.

Many devices and methods have been devised to try and reduce this boredom; a few of these include enabling interaction between users on different machines, watching television while exercising, and creating virtual environments in which to exercise. Google's Street View project has 360° images of a huge number of roads around the world. They make available an application programming interface that allows developers to access this data in their own applications. This large data set and its availablity creates an opportunity to create a realistic environment in which to exercise at a scale that has never been possible before.

The Indoor Escape system was created to take advantage of this opportunity. It creates an immersive environment that can be transversed while using a treadmill, elliptical, or bicycle. It was designed to be a low-cost, retrofitable, and easy-to-use system that helps

alleviate the boredom associated with using treadmills, ellipticals, and bicycles. While there are many other systems that create virtual environments, this is the first system, known to the author, which is adaptable to existing exercise equipment without original equipment manufacturer support. This report details the design, implementation, and testing of the Indoor Escape system.

(162 pages)

To my wife, Ann, for her support and encouragement in completing this project and continuing my education.

Acknowledgments

This project and my master's degree would not have been possible without the aid of a number of people. I would like to thank those people here.

Dr. YangQuan Chen provided very valuable suggestions and advice throughout the completion of this project. His advice on courses to take was also very valuable. I would also like to thank him for the use of his treadmill and making me a member of the Center for Self Organizing Intelligent Systems (CSOIS).

Members of the CSOIS also provided valuable input and resources in the development of the Indoor Escape system. I would particularly like to thank Cal Coopmans for taking the time to discuss ideas with me and helping me assemble the sensor circuit boards.

Two of the greatest contributors to the project are the late David G. Sant and his wife Diann. Their generosity enabled me to obtain my master's degree while providing for my family. It also enabled me to take one of my projects from an idea to a realized system. By creating the Sant Fellowship, they provided me with a very rare opportunity for which I am very grateful.

I would also like to thank my father, Lorin, for not answering all of my questions but instead teaching me how to find the answers. In addition, I would like to thank him for teaching me the value of an education. Finally, I would like to thank him for his encouragement in this project and help in proofreading this report.

I would also like to thank my mother, Trudy, for the help she has given me and my family; for teaching me to explore my own interests and allowing and encouraging me to pursue them.

To my sister, Connie, I would like to say thank you for your encouragement and for you help in formatting and editing this paper. You truly are the best sister ever.

To my children, Taggart and Avry, I would like to say thank you for letting me work the long nights I needed to and still begging me to play with you every day. As often as I have turned you down, I'm very honored that you still ask every day.

Finally, I would like to thank my wife, Ann. Thank you for your support, encouragement, and tolerance of the long hours this degree and project have taken. This accomplishment is as much yours as it is mine.

Abraham A. Clements

Contents

List of Tables

List of Figures

Acronyms

ADC	Analog-to-Digital Converter
AJAX	Asyncronous JavaScript And XML
ALP	Application Layer Protocol
API	Application Programming Interface
ASP	Active Server Page
CRC	Cyclic Redundancy Check
CSS	Cascading Style Sheet
DOM	Document Object Model
FIFO	First In First Out
HTML	Hyper Text Markup Language
ID	Identification
I/O	Input/Output
IR	Infared
ISO	International Organization for Standardization
ISP	In-Circuit Programming
ISR	Interupt Service Routine
LED	Light Emitting Diode
OEM	Original Equipment Manafacturer
PCB	Printed Circuit Board
PHP	PHP Hypertext Preprocessor
PRX	Primary Receiver
PTX	Primary Transmiter
RF	Radio Frequency
RTC	Real-Time Counter
SDK	Software Development Kit
SMD	Surface Mount Device

TCP/IP	Transmission Control Protocol/Internet Protocol
TX	Transmission
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Motivation and Prior Art

People use indoor exercise equipment for a variety of reasons. These reasons may include inclement weather, making outdoor exercise dangerous or uncomfortable; convenience of exercising at home; and a desire to exercise in private. While there are many advantages to indoor exercise equipment, using most exercise machines involves performing highly repetitive actions. In addition, by exercising indoors the changing environment experienced during outside physical activity is lost. The loss of the dynamic environment and performing highly repetitive actions causes many users to feel bored while using their treadmills, stationary bikes, rowing machines, etc. This boredom negatively influences consistent use of indoor exercise equipment [1, 2].

There have been many devices created in an effort to reduce the monotony of using indoor exercise equipment. These devices generally all work to make exercise equipment more interactive and can be loosely grouped into three broad categories: integrating exercise equipment into video games [3, 4], providing social interactivity between multiple users [1, 2, 5, 6], and virtual/augmented reality systems [7–15].

Video game systems attempt to relieve this boredom by using the exercise equipment as an input to a game. The advantage of this approach is it provides a dynamic interactive environment with rewards attached to effort. They also take advantage of gaming consoles or personal computers, thus reducing the need for systems dedicated solely for exercising. Two examples of these systems are created by Andrus et al. [3] and Waters [4].

Andrus et al. [3] claim a system which connects to a video game console and allows a large variety of exercise equipment to be used as controllers for the console. In addition,

it provides a method through which the resistance of the exercise equipment can be controlled. Waters [4] claims an Application Programming Interface (API) which enables users of the API to incorporate exercise equipment as controllers to applications. It allows the programmer to focus on developing the program with the need to implement an interface for the exercise equipment. The user of the program simply tells the program what exercise equipment is being used and the proper parameters of the equipment are mapped to the desired input of the program.

The two above approaches focus on interactive fitness equipment and making it more common and easier to implement. A second method, to reduce the monotony of using indoor exercise equipment, is to enable social interaction. Patents [1,2,5,6] primarily focus on enabling social interaction while exercising. They include methods to allow video and/or audio communication [1,2,5], communicating exercise equipment parameters such as speed and distance to multiple locations [1,5,6], and enabling central control of multiple exercise machines [1,2,5]. By enabling social interaction these systems enable a support group to be developed which can encourage consistent use of indoor exercise equipment. They do little to provide an interactive environment in which to exercise.

Particularly promising systems are those which introduce virtual reality. These systems allow the user to feel as if they are interacting with the real world. An example of one of these systems is made by Tacx [8]. It includes an indoor trainer for a bicycle that can adjust its resistance to match the terrain of a given route. The route can either be created via integration with Google Earth or by purchasing pre-recorded routes. The system also allows interactivity between remote or virtual opponents. Many other systems have been created that use prerecorded routes and a system to control the playback of the routes according to the speed of an exercise device [12–15]. Another virtual reality system called iFit Live [7,16] allows the user to create routes using Google Maps. These routes can then be traversed using iFit Live compatible equipment. Elevation data is used to control the resistance of the equipment to simulate the real environment. The routes can be visualized using Google Street View. A third system [10] was created by Aki Mimoto. Mimoto used

Google Maps API to create a system where he could bike through the images on Google Street View.

1.2 Areas for Improvement

All of the above mentioned systems reduce the monotony of using exercise equipment, but there is room for additional improvements. The virtual reality systems created by Tacx [8] require the use of their VR trainers, are targeted at competitive cyclists, and only work on bicycles. The systems that use prerecorded routes are all dependent upon recorded videos of the routes [12–15]. Prerecorded routes add to the expense of the system and the locations that have been recorded are usually limited. These systems also require the purchase of a compatible player for their videos. The iFit Live system requires iFit Live compatible equipment that was released in 2009. Thus, most people would need to purchase new exercise equipment to use this system. Many people already own indoor exercise equipment and the cost of purchasing special exercise equipment to use these virtual reality systems is prohibitive. Mimoto's system is the only system that does not require the purchase of specific equipment, but his system only works for bicycles and he readily admits that his system is not stable or easy-to-use [10].

Thus, there is still currently no system that reduces the monotony of using indoor exercise equipment that is easy-to-use, does not require the purchase of compatible equipment, and is low-cost.

1.3 Project Proposal

To address the short coming mentioned above, a system called the Indoor Escape was designed, built, and demonstrated that reduces the monotony of using indoor exercise equipment and meets the following criteria: easy-to-use; retrofitable to treadmills, bicycles, and elliptical machines; and is inexpensive. This report details the steps undertaken to design build and demonstrate the Indoor Escape.

1.4 Chapter Overview

This report is organized into chapters containing the information needed to understand the manner of system implementation and the how and whys of the same. Each chapter discusses critical elements of the design and implementation of the Indoor Escape system. A brief overview of each chapter is given below.

Chapter 2, Design Goals and Constraints, contains background information on the technologies used to create the Indoor Escape and the restraints they put on the design of the system.

Chapter 3, Design Over View, provides a high-level overview of the design of the entire Indoor Escape system. It includes the block diagram of the entire system and high-level discussion of the three major subsystems. The three major subsystems are the sensors, micro-server, and display subsystems. It discusses component and technology selection and the criteria used to choose the components and technologies.

Chapter 4, Communication Protocols, describes the communication protocols used between the subsystems of the Indoor Escape system.

Chapter 5, Sensors and Wireless Receiver Implementation, discusses the implementation of sensors and the wireless receiver used to detect the rate of travel for the treadmill, elliptical, and bicycle. This includes descriptions of both the hardware and software used to implement the sensors. It also discusses the methods and results of testing.

Chapter 6, Micro-Server Implementation, provides a detailed description of the micro-server in the Indoor Escape system, including its design and testing.

Chapter 7, Display Subsystem Implementation, details the functionality of the display system its functionality and its testing. The display system provides the user interface and controls the entire system.

Chapter 8, Complete System Testing, details testing of the entire system and contains a presentation of the performance of the system.

Chapter 9, Conclusion and Future Work, concludes the report with a discussion of future work, improvements that could be made to the system and lessons that were learned

from developing the Indoor Escape system.

Chapter 2

Design Goals and Constraints

The Indoor Escape system is designed to be a low-cost, easy-to-use system that creates an immersive exercise environment for treadmills, elliptical machines, and bicycles. A design constraint that could not be eliminated was time. The development and demonstration of the Indoor Escape system was to be completed by May 1, 2010. This chapter discusses the technologies that were chosen to meet the cost, ease-of-use, and immersive environment goals. Schedule restraints made it imperative to use off-the-shelf technologies and equipment as much as possible. These technologies restricted the design options and those constraints are also presented.

2.1 Goals

Cost Goal: The Indoor Escape system is designed to adapt to existing exercise equipment. This is the niche of the market that is yet uncaptured. Unit equipment costs were kept below $100 to make it feasible for adaptation to home/club equipment.

Ease-of-Use Goal: Adaptation to existing equipment requires some easily installed components that can be placed by an average ability non-technical person. Plug and play type devices are consistent with ease-of-use.

Immersive Exercise Environment: Creating artificial environments to exercise in, given the previous goals, constrains the possible applications. For instance, multi- projector virtual reality imaging systems would be possible but cost and ease-of-use eliminate that option. The selection parameters for this goal are: easily accessed, readily available, and commonly displayed. This component had to be the first selected to enable The Indoor Escape system to meet the design goals. To have an immersive environment, data for that environment is needed. This data has typically been created in two ways: producing a

virtual reality and/or recording the real-world. Both methods have their advantages and disadvantages. Virtual reality can provide limitless environments, but the quality of the environments has typically not been on par with the real-world. Using real-world data provides a very natural and realistic feeling environment but requires the collection of the data. This has in the past been the largest disadvantage of using real-world data. Google Street View changes this by providing a nearly limitless set of real-world images. Google Street View was considered and selected based on the criteria of access, availability, and display. The schedule impact of data collection was also reduced by this choice. The paragraphs that follow describe the other constraints of this choice.

2.2 Constraints from Using Google Street View

Google Street View is a feature of Google Maps that allows viewing of 360° panoramas of any street that Google has taken images of. Google, as of May 2010, has taken images of many of the streets in the United States, Canada, Japan, Australia, western Europe, and many other locations [17]. Google provides an API that enables their data to be used in custom applications. The large set of high quality images at street level from all over the world and the availability of an API to access them were deciding factors in the decision to use Google Street View to provide the immersive environment.

Using Google Street View places some constraints on the design of the system. First the use of Google's Maps API is dependent on Google's terms of service. The conditions of the terms of service that had the most influence on the design of the Indoor Escape system are described here. For the other conditions the reader is referred to the terms of service [18].

1. Data cannot be accessed by any method other than the approved API.
2. Implementations using the API must be publicly available and free to use.
   (a) An exception is made for development.
   (b) A license can be purchased allowing commercial use and/or fees to be charged to use the interface.

3. All property right notices must be displayed and Google's logos/copyright notices cannot be obscured.

The only method that the Google Maps API provides for accessing Street View panoramas requires Adobe Flash Player 10 or higher. This was determined by disabling Flash in a web browser and then visiting www.google.com\maps and attempting to activate Street View. Google Maps API with access to Street View is also only available in JavaScript thus restricting the language in which the Indoor Escape system could be implemented. The Flash Player and JavaScript dependencies require the Indoor Escape system to be implemented using a modern web browser. When implementation started in December 2009, the only available platforms which had a web browser that supported flash were personal computers running Linux, Windows, Mac OS X, and the Nintendo Wii. The Wii was considered, but by testing it was discovered that its version of Flash Player is incompatible with Street View. Since many homes have computer systems running Windows, Linux, or Mac OS X, the personal computer provides a logical choice for the platform to deliver the Indoor Escape system's immersive environment. By utilizing the personal computer the initial cost of the system is reduced.

2.3 Limitations From Using a Web Browser

Implementing the Indoor Escape system in a web browser adds restrictions and challenges to the process. Web browsers are designed to allow content to be displayed independent of the environment in which the web browser is operating. In theory a web page should look and behave identically on a computer regardless of the operating system or device the web page is being viewed on. In order for web pages to obtain independence from the environment of the browser, web standards essentially operate on the lowest common denominator of a system. This means web pages do not have access to system devices such as the file system, communication ports, etc. Extensions can be made for many browsers which provide an interface between system level devices and web pages, but doing so restricts the usefulness of the web page to a particular browser and operating system.

Another reason that web browsers restrict access to the system level devices is security. Content viewed on the web is not trusted, thus giving it access to your system level devices could be very dangerous. To implement this security web browsers use what are called security sandboxes.

Security sandboxes are isolated environments in which scripts are executed. A web page may contain content from several different domains. The content from each domain is placed in its own sandbox and not allowed to access scripts from a different domain unless that domain gives it explicit permission to do so. Since the display of the Street View data has to be done in a web browser, speed information from the exercise equipment will need to be accessed from within the web browser.

The restrictions placed by the web browser on access to system level devices complicate the transfer of speed information from a sensor to the web page in the web browser. There are ways of getting around the restrictions imposed by web browsers, and they are often used to exploit computers, but if a method to get speed information to the web page circumvents the security framework of the web browser it is likely to be removed by browser updates, or blocked by security software. Thus, to have a reliable system it is essential that the speed information is accessed by the web page via means that operates within the security and abstraction frame work web browsers impose.

2.4 Sensor Restrictions and Design Goals

The overall system design goals place restrictions on the design and choice of sensors used to detect the rate of travel of the exercise equipment. The goal to create a system that retrofits to existing exercise equipment without Original Equipment Manafacturer (OEM) support, and is easy-to-use requires that the sensor be easy to attach to most treadmills, elliptical machines, and bicycles. They also must be easy-to-configure and account for varying installation configurations caused by the range of designs in exercise equipment. Additionally, the system must not create a safety hazard to the user of the equipment.

Exercise equipment has many moving parts, thus exposed wires running from a sensor to the computer displaying the immersive environment pose a potential safety hazard.

Therefore, a requirement of the design is that the sensors be wireless. Using wireless sensors will also make the system easier to install. Using wireless sensors requires that the sensors be powered by batteries. This imposes power restrictions on the sensors. Constantly changing batteries would increase the complexity of using the system; thus, the power consumption, battery size, and cost of the sensors all need to be taken into account when designing the wireless sensors.

2.5 Summary

The Indoor Escape system will use data from Google Street View to provide an immersive exercise environment. Using Google Street View data restricts the platform in which the Indoor Escape system is delivered to a modern web browser with Flash Player version 10 or higher, it also requires compliance with Google's terms of service. The use of a web browser as the delivery platform imposes the security restriction of a web browser on the design of the system. These restrictions complicate obtaining speed information from the sensors, but appropriate means must be used to obtain the speed data for a reliable system. The design of the sensor must be wireless for safety reasons. To reach the goals of low-cost, and ease-of-use, power consumption, battery size, cost and installation variations need to be taken into consideration when designing the sensors. The final goal was to complete the system by May 1, 2010.

Chapter 3

System Design Overview

3.1 Design Method

The system was designed using three steps. First, system functionalities which were necessary to meet the design goals were identified. Second, the system was broken into subsystems by grouping common functionalities and defining interfaces between the subsystems. Third, design options for each subsystem were explored and components and technologies were selected via the criteria of cost, ease-of-use, ability to provide an immersive environment, and ability to retrofit to treadmills, ellipticals, and bicycles.

3.2 Require System Functionality

The Indoor Escape was designed to create a low-cost, easy-to-use system that retrofits to exercise equipment. The first step in creating this system was identifying the required functionality. The features selected were chosen to meet the design goals. As discussed in the previous chapter it was decided to use data from Google Street View to provide an immersive environment. To create the immersive environment using Street View, three things need to be accomplished. First, the images from Street View need to be obtained and displayed. Second, those images need to have natural succession, which is obtained by following a real-world path. Third, the changing of images needs to be connected to the distance traveled on the treadmill, elliptical, or bicycle. Thus, providing the dynamic exercise environments requires the following features:

- a method to create a route from which images are selected,
- a method to retrieve images,
- a method to control the display of images according to distance traveled,

- a method to obtain information to calculate the distance traveled.

Supporting the above list of features will meet the goal of having an immersive exercise environment, but additional functionality is needed to meet the ease-of-use requirement. The method of creating a route needs to be simple for both long and short routes. To meet this goal it was decided to support two methods of creating a route. The first method is clicking on a map to set way points. This will work for short routes but may be difficult for long routes; thus, the second method allows the typing of beginning and ending locations and automatically creating a route between them.

During development two other features were included to increase the ease of using the system. The first added feature is the ability to save routes. This allows previous routes to be reused reducing the effort required to start using the Indoor Escape. Two methods of using a saved route are to be supported, restarting from the beginning and resuming from the last location. Resuming a route further reduces the difficulty of using the system because it allows one long route to be created and then used over multiple sessions. For example, a user could create a route from Boston to Los Angeles and then run it over the course of several months. The second additional feature needed is a simple method to add, select, and configure sensors. The information about the last sensor used is also saved so that once the sensor is configured it can be automatically used again.

Obtaining the information about the distance traveled also requires several features. It requires that a sensor be capable of obtaining information about the distance traveled. By integrating the speed of the exercise equipment the distance traveled can be obtained. It is sufficient for the sensor to obtain only speed information, but the sensor must be capable of measuring the speed of the exercise equipment accurately enough so as to not diminish the quality of the immersive environment. The sensor must be able to obtain the speed of a range of different models of exercise equipment. It will be cost effective to design a different sensor for the different types of exercise equipment. For example, one sensor could be designed for treadmills, another for ellipticals, and another bicycles, but it would be too expensive to design a sensor for each model of treadmill, elliptical, or bicycle. Therefore,

another required feature of the system is a few sensors that can easily be installed on a many different models of exercise equipment and still accurately measure the speed of the system. The cost of the sensors must also be considered. The final requirement is the sensor needs to be able to communicate the speed to the immersive environment. As discussed in Chapter 2, this communication needs to be wireless and must work within the security restrictions placed by a web browser.

In summary, by providing the capabilities in the list below the Indoor Escape system can provide a cost effective, easy-to-use, retrofitable, immersive environment for indoor exercise equipment.

1. Ability to create a route by setting way points
2. Ability to create a route by entering start and stop locations
3. Ability to save and retrieved routes
4. Ability to restart a saved route from either beginning or last location
5. Ability to retrieve and display Street View images from the route
6. Ability to control the displayed images according to distance traveled
7. Ability to easily add, configure, and select a sensor
8. Ability to measure speed of the treadmill, elliptical, or bicycle
9. Ability for sensors to account for installation variations
10. Ability to communicate speed information wirelessly and within the security restrictions of a web browser.

3.3 Defining Subsystems and Communication Interfaces

The system capabilities described above were grouped together based on common requirements and functions to define subsystems. Breaking the design down this way allowed each subsystem to be designed, implemented, and tested in isolation. This reduced the

complexity that needed to be managed simultaneously. Three subsystems were defined as shown in fig. 3.1. The display subsystem is solely responsible to provide abilities 1-6 and partially responsible for capabilities 7 and 10 from the above list of required system capabilities. The micro-server subsystem facilitates communication between the sensors and the display subsystem within the security restrictions of a web browser. The sensor subsystem provides capabilities 8 and 9 from the above list and contributes to capabilities 7 and 10. Two communication interfaces were also designed. It was decided to provide communication between the micro-server and display subsystems by use of a Transmission Control Protocol/Internet Protocol (TCP/IP) socket and a custom command set. Communication between the micro-server and the sensors is provided via a custom wireless protocol. The design of each subsystem and the selection of components and technologies are discussed in the following sections. The communication interfaces are discussed in Chapter 4.

3.4 Display Subsystem

3.4.1 Component and Technology Selection

The display subsystem is the heart of the system and provides the majority of the required capabilities of the systems. This section will describe each feature the display subsystem supports and the components and technologies used. It will discuss how the selected components and technologies fulfill the design goals and why they were select over other alternatives.

The selection of Google Street View to provide the immersive environment restricts

Fig. 3.1: Block diagram of Indoor Escape system.

the platform in which the display subsystem can be implemented to web browsers. In theory, web browsers fully conform to standards and web pages function identically in all web browsers. In reality, this is not the case, but in general compliance is pretty good and constantly improving. Therefore, it was decided that the Indoor Escape would be developed to work with Google's Chrome browser and compatibility issues with other browsers would be left for future work. This was decided because the goal of the project was to create the Indoor Escape system, not to deal with web browsers' inconsistencies. Generally, modifications needed to obtain cross browser compatibility are minor. Chrome was selected because it is one of the most compliant browsers [19], has one of the fastest JavaScript engines [20], has capable built- in development and debugging tools, and is available on all three major operating systems. The selection of Chrome as the demonstration web browser imposes very few if any restrictions on the system.

Since the display subsystem will be implemented in a web browser it will be a web application. Therefore, it is beneficial to explain a few of the basics of how a web application works. First, it is important to understand what happens when a webpage is loaded. After receiving a webpage the web browser reads in the Hyper Text Markup Language (HTML) of the file. The HTML defines objects using tags, each set of tags defines an object that contains information either about the webpage or information to display. Most webpages contain a tag which instructs the web browser to load a Cascading Style Sheet (CSS). The CSS tells the browser how to display each tag. The web browser then creates a linked list. Each node of the linked list is a tag from the HTML and associated with it is the style information obtained from the CSS. The tags in the HTML are often nested, and thus most nodes in the list have a parent node and children nodes. This linked list is referred to as the Document Object Model (DOM). After the creating the DOM, the DOM is passed to a rendering engine which draws the webpage on the screen.

A web browser has two method of requesting content: synchronously and asynchronously. Synchronous requests remove the current web page then request a new one displaying the new page when it is received. Asynchronous requests use what is commonly is known

as Asyncronous JavaScript And XML (AJAX). Web sites that use AJAX usually use JavaScript to request additional information from a server after the webpage is loaded as triggered by events. When the data is returned JavaScript is used to update the DOM. When changes are made to the DOM the browser redraws the webpage as needed to reflect the changes.

Perhaps the most familiar use of this method is Google's auto suggest when performing a web search. When you type into the search input box, JavaScript is used to send the text you have entered to Google's servers. The servers then looks up common searches that match the text you have entered and return it. When the data is received by the web browser, JavaScript is used to insert the returned results so they appear just below the search input box. If this was done using synchronous requests every time you entered a letter, the entire page would have to be reloaded. The advantage of AJAX is that it allows a more responsive application, because the user does not have to wait for an entire page to load every time something is new is needed. Instead just the new content can be loaded and added to the current webpage.

The primary function of the display subsystem is to provide a user interface to the Indoor Escape. This includes methods to create, retrieve, select, and traverse a route. Since all of these methods depend on the creation of a route, how that route is created is one of the most critical decisions for the display subsystem. It was determined that to meet the ease-of-use goal that two methods of creating a route would be implemented. Clicking to set way points and entering beginning and ending locations. Clicking to set waypoints requires a map that can be clicked on with each click setting a way point. The second method requires a way to interpret the input locations and create a route between them.

To accomplish this, two options were considered: Bing Maps Software Development Kit (SDK) [21] and Google Maps API. Both options would have provided the needed functionality to create a route using the two methods. Since it was decided to use Google Street View, which requires the Google Maps API, this API was chosen. Using Bing Maps SDK, would have increased the complexity of the design and required compliance with a

second set of terms of service, without providing any additional benefits.

Google Maps API is available for JavaScript and ActionScript. ActionScript is the language used by Flash and is a compiled language and thus usually runs faster than JavaScript. However, only the JavaScript API allows interaction with Street View images. ActionScript does have the ability to call JavaScript functions so a wrapper could be written for the needed functions to allow ActionScript to be used to control the Street View data. However, ActionScript does not have access to the webpage's DOM, therefore changing any information on the webpage would also require calling JavaScript functions. The majority of the display subsystem requires changing content on the web page. Therefore, any speed advantages obtained from using ActionScript would be minimal, but the complexity of exchanging data between JavaScript and ActionScript would be added. It was decided to use the Google's JavaScript Maps API to enable the creation of routes.

Two types of scripting are used to allow dynamic content on web pages. One method is client side, the other is server side. Client side scripting delivers the code to the client where the code is executed. Client side scripts only have access to resources provided by the web browser. JavaScript and ActionScript are examples of client side scripting languages. Server side scripts run on the web server and only the results of the execution are returned to the client. Server side scripts often receive information about the user and then use this information to query a database and use the information from the database to customize the page to the user. This way only the web server needs access to the database improving the security of the database.

To save a route two options were considered saving it to the local computer or on a remote database. Saving it to the local computer could be accomplished by setting cookies. The problem with this approach is that cookies were designed to temporarily store small amounts of information. Created routes could potentially be very large, making saving them as cookies very inefficient. The other problem is that the routes could potentially be stored for very long periods of time. It would also be desirable to be able to use the Indoor Escape system on multiple computers and still have access to your saved routes. For this

reason, it was decided to use a database to save the information to a remote server. It was decided to use a MySQL database because of the creators familiarity with it, and its wide availability. MySQL is an open source database, thus the cost of setting up a server with it is minimal. It is also a very capable database with large websites, such as Facebook [22] and Wikipedia, being run with MySQL back ends [23]. In order to save information to a database server side scripting is used.

There are many languages available for server side scripting two of the most popular are Active Server Page (ASP) by Microsoft and PHP Hypertext Preprocessor (PHP) an open source language. It was decided to use PHP for all server side scripting because it is readily available, and it provides good support for querying MySQL databases. In addition, the author has experience with it. Using MySQL and PHP will allow the saving and retrieval of routes and sensor configuration. By using PHP for server side scripting, MySQL to store data, JavaScript for client side scripting, and Google Maps API, the technologies needed to enable the creation, retrieval, and traversal of route are available.

The last technology needed is a means of communicating with the sensors. Several options were considered. These included writing an extension for a web browser, using ActionScript's socket class [24], and AJAX requests. The first option of building an extension would limit the system to one browser; in addition, a different extension would have to be built for each operating system the Indoor Escape supported. To access a communication port through an extension, the extension would have to be written to expose components to an external program. Then a native program would have to be written that interfaced with the communication port and passed the data to the extension using the exposed components of the extension.

The second option uses ActionScript's socket class. The socket class creates a raw binary, bidirectional socket connection. The connection is made using TCP/IP. To uses this method a TCP/IP server that communicates with the sensor would have to be created. It would obtain data from the sensor and then push the data through the socket connection to a flash application. The flash application would then call JavaScript functions to communicate

that information to the display subsystem. The advantages of this method are that it would work on any browser that supports Flash Player 9 or higher and the implementation of a flash program is significantly simpler than a browser plug in. Flash Player 10 is required by Street View so this method introduces no new requirements for the system to work. It would also allow the sensor data to be pushed to the browser.

For the last option, using AJAX is very similar to the flash socket except that JavaScript would be used to make asynchronous requests to a custom web server that communicates with the sensors. The design of this server would be more complicated than the design of the server for the flash socket because it would have to handle web requests instead of just a binary stream. In addition, it would require the display system to query the server at intervals to obtain the speed information and require a request reply communication method.

Each method is about equal in terms of the design goals of low-cost, ease-of-use, and providing an immersive environment. The ActionScript socket class will allow lower latency communication with the sensor because it will not require creating a new TCP/IP connection for every sensor update or require waiting for the display subsystem to request the speed information. It is also a much simpler implementation and does not place restrictions on the browser used for the Indoor Escape, therefore it was selected to enable communication with the sensors.

3.4.2 Design

After selecting the needed technologies to create the display subsystem, the system was designed. The display system needs to enable the creation of a route and the retrieval of a route. It also needs to support the traversal of the route and communicating with the sensors. The algorithms to accomplish these are presented here. Creation of a route is accomplished though the use of a class provided by Google's Map API class called GDirections [25]. It accepts either beginning and end locations or a set of way points and returns directions from the beginning to the end or from the first way point to the next. If the inputs are not latitude longitude coordinates, the information is geocoded to obtain lati-

tude and longitude points for the input locations and then the directions are obtained. The directions include a list of latitude and longitude coordinates which define the vertices of a polyline that highlights the route on the map. These vertices are used to define a route. By drawing a straight line from a vertex to the next the route can be traced.

The algorithm to create the route is given in the flow chart shown in fig. 3.2. The algorithm is run when any of three events occur. These events are: the map is clicked, a marker is clicked, or the button create route is clicked. When the map is clicked, a marker is added to the map at that location and the location of the click is added to a list of way points. If there is more than one way point in the list, directions between the way points are requested using the GDirections class. When the directions are returned, the route is drawn on the map. When a marker is clicked, it is removed from the map and list of way points. If there is more than one way point in the list, then directions are requested between the remaining way points and the route updated on the map. When the button create route is clicked, the input boxes for the beginning and ending location are first checked to make sure they are not empty. If they are not empty, then the locations are geocoded to obtain latitude and longitude coordinates. The route between these coordinates is then obtained and displayed on the map. If an error occurs either while geo coding or because an input box is empty, an error is displayed and the algorithm terminated.

After creating a route there are several ways to start traversing it. Traversing a route is accomplished by first initializing the route for traversal and then traversing the route. The flow chart for initializing a route is shown in fig. 3.3 and the figure for traversing a route is shown in fig. 3.4. The initialize route loads a buffer of street view images and sets the values of some variables needed to traverse the route. These variables are FrameToLoad, DistTraveled DistToNextFrame, CurFrame, NextFrame, and NumInBuffer. FrameToLoad is used to indicate the next frame to load a Street View image into. DistTraveled keeps track of how much distance has been traveled, and DistToNextFrame keeps track of how much distance needs to be traveled before advancing to the next frame in the buffer. CurFrame and NextFrame are used to keep track of which frame is being displayed and which frame

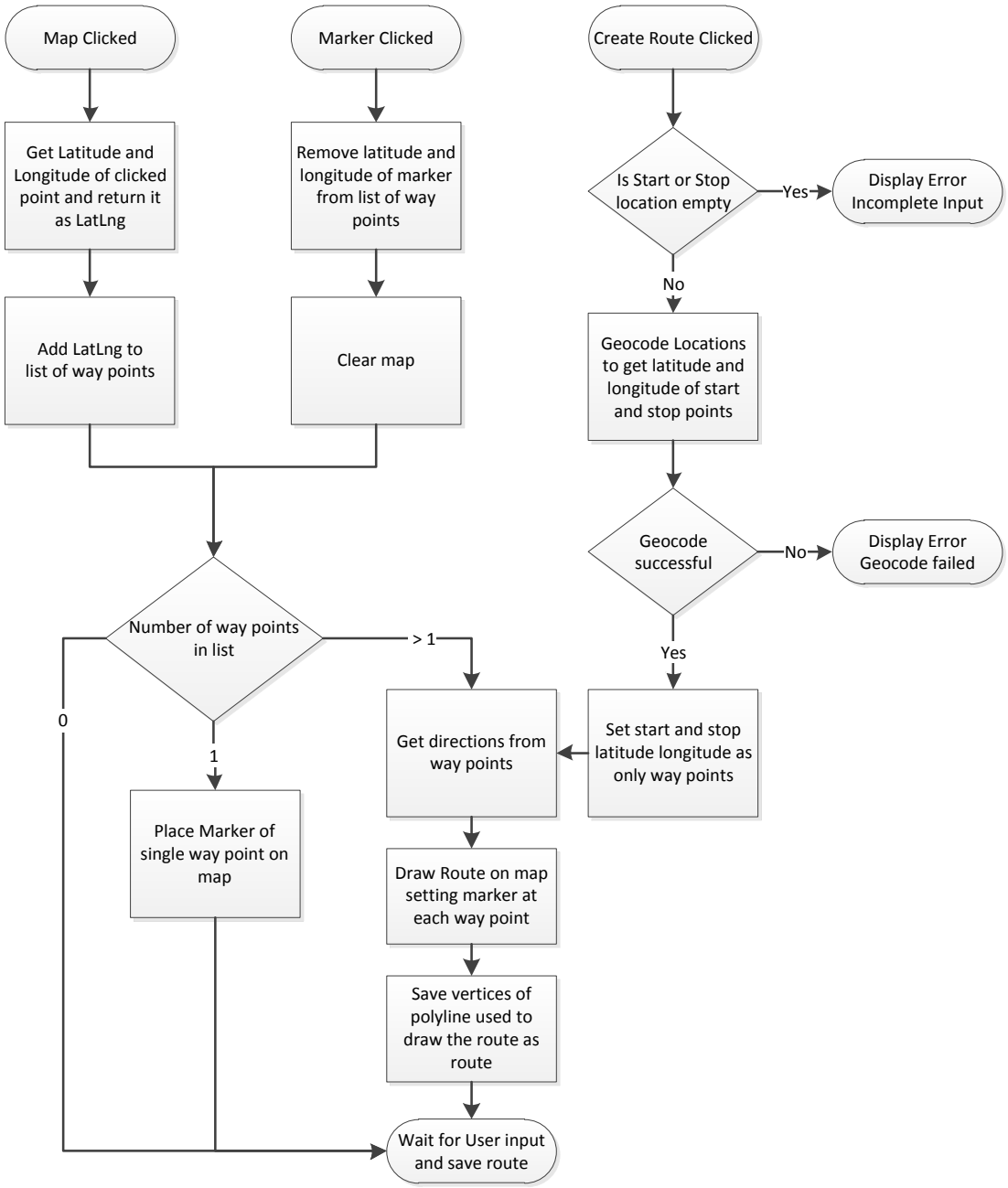

Fig. 3.2: Flow chart of algorithm used to create a route.

should be displayed next. NumInBuffer is used to keep track of the number of valid frames that are stored in the buffer. The initial values given to these variables depend on how the route was started.

There are three ways to start traversing a route: from a newly created unsaved route, restarting a saved route, and resuming a saved route. The first method is initiated by creating a route and then clicking start, the second and third options occur by clicking the saved routes associated resume link or restart link. The flow chart in fig. 3.3 shows how each of these different cases are handled. It also shows how values are assigned to the needed variables. The initialize route algorithm finishes by starting an interval timer. This is the interval at which the traversal algorithm shown in fig. 3.4 is run. The interval is set to 500ms, thus the traversal algorithm will check every half second to see if it needs to change the image that is being displayed. This value was chosen so that the display system would have good response to changes in images and that the integration on the speed information would be fairly accurate.

The traversal algorithm checks to see if it needs to advance frames, if it does then it advances the frame and requests a new image to load into the frame that was just passed. If the route was previously saved, it also stores the current location and distance traveled each time a new image is loaded. When the end of the route is reached, a message is displayed and if the route was not previously saved the user is asked if they would like to save the route. The traversal algorithm depends on the current speed of the exercise equipment. This is held in the variable CurSpeed, and the flow chart shown in fig. 3.4 assumes that this value is updated externally.

The final functionality that needs to be provided by the display subsystem is to configure, add, select, and remove sensors. The sensor options will be shown in a separate window from the window used to create a route. The new window will be launched by clicking a link. This display will show all of the sensors currently in the system. Each sensor can then be configured, removed, or selected as the active sensor. In addition, this window will provide the option of adding additional sensors to the system. The chain of events followed

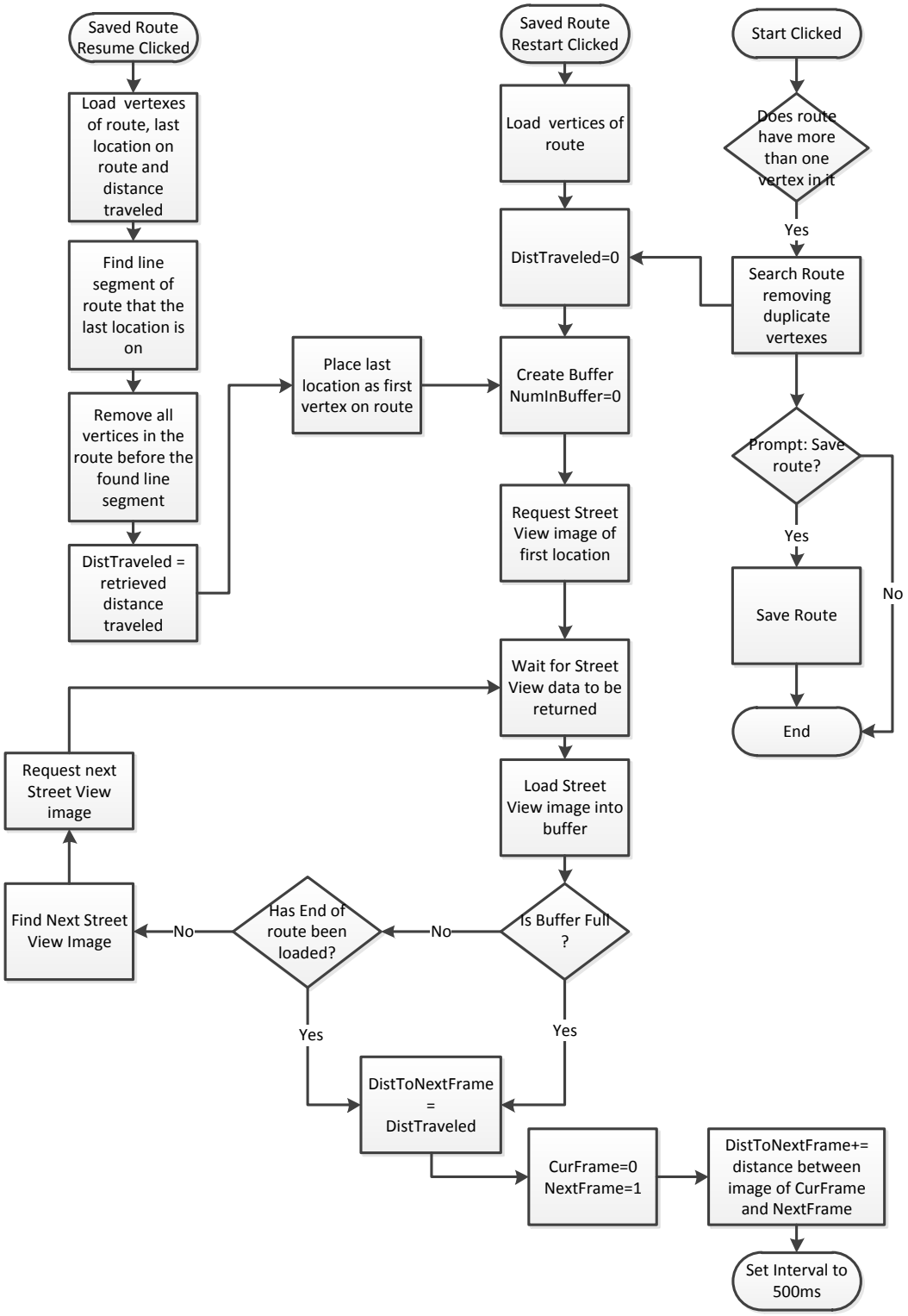

Fig. 3.3: Initialize route flow chart.

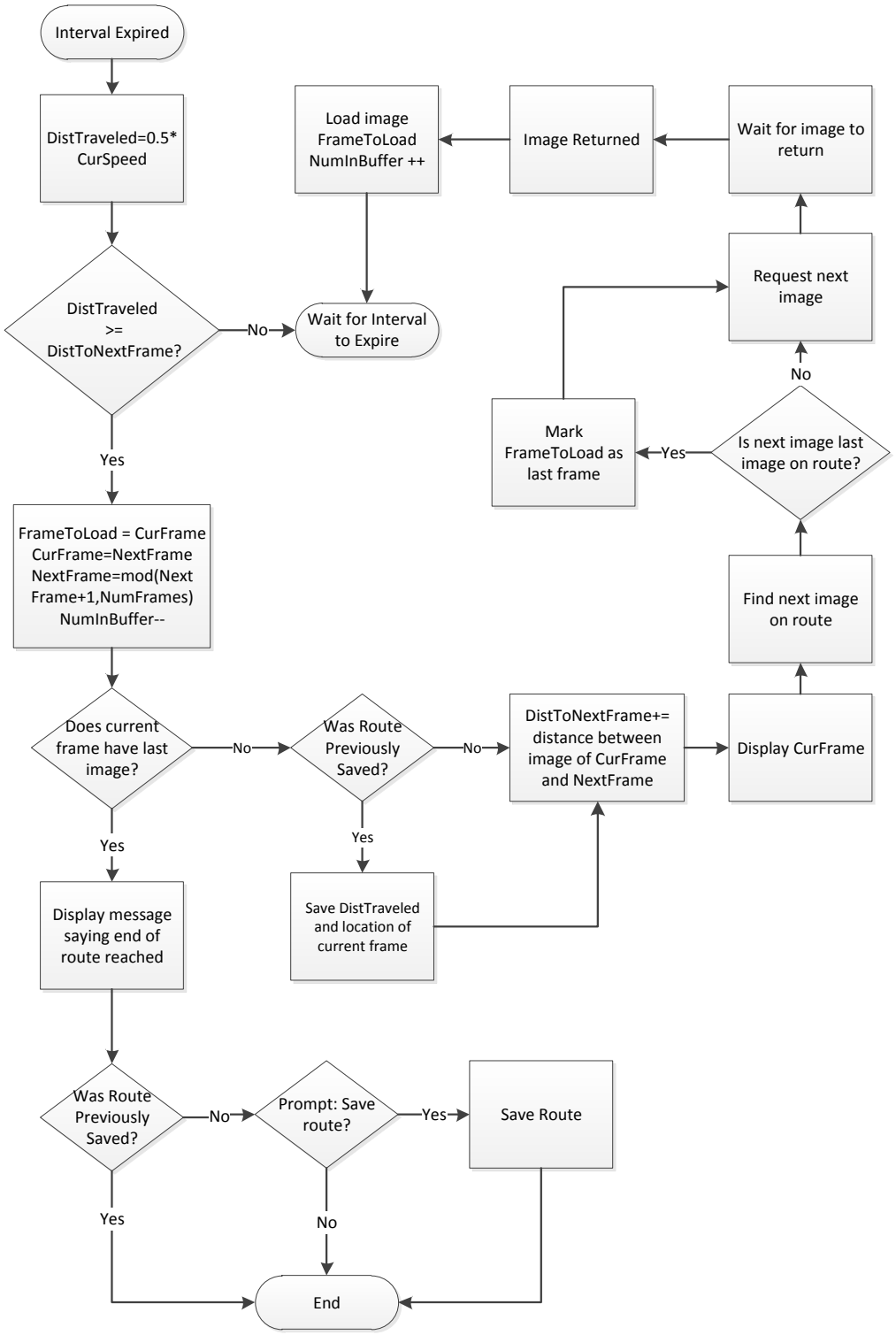

Fig. 3.4: Flow chart of traversal algorithm.

to perform these actions is shown in the flow chart shown in fig. 3.5.

3.5 Sensor Subsystem

The sensors need to be able to measure the speed of the exercise equipment and communicate that speed to the display system. In addition, they need to be able to attach to many different models of exercise equipment and be tolerant of variations in the installation. It was decided that a different sensor would be built for treadmills, ellipticals, and bicycles. Thus three different sensors were to be designed. Each sensor would use a common communication interface and the sensors would be kept as similar as possible without compromising performance. By keeping the design of the sensors common the investment in the design and development is reduced. The method of detecting the rate of travel for each type of exercise equipment will be presented first, and then the components common to all of the sensors will be discussed.

Three types of sensors where considered for the three types of supported exercise equipment. They were a reflective Infared (IR) sensor, an accelerometer, and a Hall-effect sensor. The design and method of using each for a treadmill, elliptical, and bicycle are different, and the method of operation is described for each sensor and type of equipment below.

3.5.1 Treadmill Speed Sensing Method

Treadmills consist of a moving belt on which a person runs or walks. The track is typically made of black rubber. Beside the belt there are usually two rails that are meant for the user to stand on while the treadmill is starting up. These rails typically run the entire length of the track. The top or bottom of these rails provide a good place to attach a sensor because it is close to the track and does not pose a hazard to the user. The IR sensor would be attached to this rail such that the sensor is over the track. The black rubber of the track will absorb most of the IR light. By attaching a reflective object to the track the passing of the object under the sensor could be detected each revolution. This reflective object could be a sticker or painted stripe. This would provide a low-cost, easy-to-use sensor. By using an Analog-to-Digital Converter (ADC) to measure the signal

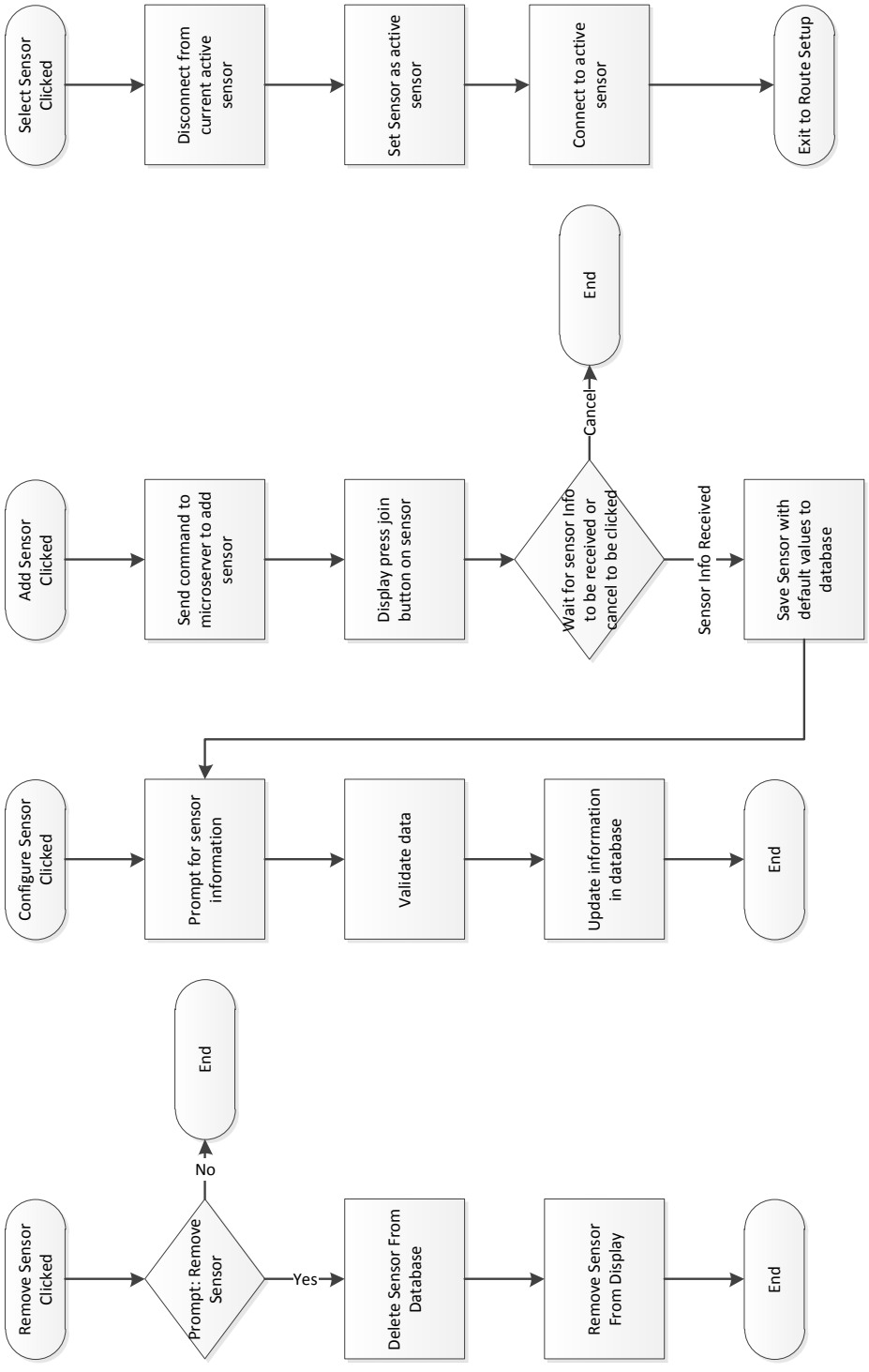

Fig. 3.5: Configure sensors flow chart.

on the sensor, variations in the distance between it and the track can be accounted for. This allows the sensor to work on a wide range of treadmills. The only information the user would need to provide is the length of the track, thereby making it easy to set up.

The Hall-effect sensor would work in a manner very similar to the IR sensor. A magnet would need to be attached to the track instead of a reflector. Attaching a magnet to the sensor presents challenges because the surface under the rubber track is typically ferrous, thus using two magnets would not work because the magnet under the track would just stick to the ferrous surface. Additionally, there can be very small clearances between the frame of the treadmill and the belt, particularly where the belt passes over the rollers at the ends of the track. Thus, any object sticking up could become caught in the treadmill. A small, flat magnet could be glued to most tracks, but the smaller the magnet the harder it is to detect it, thus reducing the robustness of the sensor to variations in treadmill designs.

Using an accelerometer to detect the speed of the treadmill would be much more complicated. Every time a foot strikes the treadmill it vibrates the track; by detecting this vibration and knowing an average stride length, an approximation to the speed of the treadmill could be determined. The amount of vibration caused by a foot strike would depend on many factors such as, the treadmill, the person running, and the surface on which the treadmill is on. Thus, the algorithm used to detect the foot strike would need to account for all of these variations. This could be accomplished by recording a few foot strikes and then running an auto correlation between the recorded signals and the current measurements. The peaks in the autocorrelation would identify a foot strike. Depending on the length of the autocorrelation that would have to be calculated significant computation power may be required to obtain real time performance. Thus the cost of using an accelerometer could be much higher than using a Hall-effect or IR sensor. In addition, the accelerometer itself is much more expensive than the other sensors.

A decision matrix comparing the three sensors is shown in Table 3.1. The IR sensor was selected because of its low-cost, ease-of-use, and ability to tolerate various treadmill designs.

Table 3.1: Decision matrix used to select sensor method for treamills, ellipticals, and bicycles.

	Treadmill			Elliptical			Bicycle		
	IR	Accel.	Hall-eff.	IR	Accel.	Hall-eff.	IR	Accel.	Hall-eff.
Cost	4	2	5	4	3	5	4	3	5
Ease of Use	5	5	2	2	5	1	4	4	4
Variations	4	3	3	2	4	1	4	4	4
Power	3	5	4	3	5	4	3	5	4
Total	16	15	14	11	17	11	15	16	17

3.5.2 Elliptical Speed Sensing Method

The same sensors were evaluated for the elliptical but the theory of operation is different. The design of elliptical machines is highly variable compared to treadmills. The common feature of all ellipticals is that they make the users feet move in some sort of elliptical shape. This is done by attaching one end of a foot rail to a wheel and then attaching the other to some sort of linkage. This linkage may have several joints which are used to modify the elliptical shape in which the foot rail moves. On some machines this wheel is easily accessible; on others it is covered. To use the IR sensor, a moving part of the machine would need to pass by a stationary part of the machine where a sensor or reflector could be attached. The stationary part of the machine would also need to be close enough (less than a cm or so) to the moving part of the machine. The moving part of the machine would need to have space to attach a sensor or reflector, whichever is not attached to the stationary part, without interfering with the operation of the machine. The Hall-effect sensor would have the same requirements except the distance between the two parts of the machine would need to be much less than the IR sensor. With the high variability in the design of elliptical machines it was determined that it would be very difficult to ensure that either the Hall-effect or IR sensor would work. Even when such a location exists on the elliptical, describing it to the end user without providing custom instructions for each model could be very difficult, complicating the installation.

The accelerometer takes advantage of the one common trait identified in ellipticals.

Since all ellipticals move the exercisers feet in an ellipse twice per revolution the acceleration vector along the forward axis of the exercisers foot will switch directions. Detecting one of these switches is used to indicate the completion of a revolution of the wheel or two steps. Attaching the sensors can be accomplished easily by placing it near the foot hold on the elliptical. Table 3.1 shows the decision matrix comparing the three sensors on how they meet the design goals. The accelerometer was selected mostly because it allows a single sensor to work for nearly all ellipticals.

3.5.3 Bicycle Speed Sensing Method

All three sensors could be used to detect the speed of a bicycle. The IR sensor and Hall-effect sensor require a reflector or magnet to pass by the sensor once per revolution, and the accelerometer would take advantage of the fact the that bicycle wheel moves in a circle and thus the acceleration vector will change directions twice per revolution. Installation of all the sensors would be pretty straight forward. For the IR sensor either a reflector would be attached to the wheel or the existing reflector would be used. The Hall-effect sensor would require attaching a magnet to the spokes of the wheel and the accelerometer would just need to be attached to the wheel such that the axis that is measured is roughly parallel with the plane containing the rim of the bicycle. The Hall-effect sensor was selected because it is the cheapest. The decision matrix used is shown in Table 3.1.

In summary, it was decided to use a reflective IR sensor to detect the speed of a treadmill, a Hall-effect sensor for the bicycle speed sensor, and an accelerometer for the elliptical. After selecting the method of detecting the speed for the various types of exercise equipment it was necessary to determine the exact components to be used.

3.5.4 Sensor Component Selection

For the reflective IR sensor it was determined to use the QRD1114 [26] from Fairchild Semiconductor. It meets all the goals and requirements of the design. It is available from Mouser for $1.04 for a single unit with the price dropping under 50 cents for quantities of 500 or more. Through testing, it was determined that using the circuit shown in fig. 3.6

that a reflective object about 2.5 cm from the sensor could be accurately detected. This provides enough distance to allow the sensor to perform reliably on many different models of treadmills.

The SS411P Hall-effect sensor from Honeywell was selected for use on the bicycle because of its low-cost and acceptable performance. It will operate at low voltages down to 2.7V requires 5.5 mA of current [27]. The accelerometer selected is the MMA7361LT from Freescale. It is a three-axis radio metric accelerometer. It has two ranges of sensitivity with maximum values of 1.5g or 6g and requires 400uA of current. In sleep mode, the power consumption is just 4uA meeting the goal of low power consumption [28].

3.5.5 Wireless Radio Selection

The selection of the sensors provides only part of the capability to detect the speed of the exercise equipment. All of the sensors require a microcontroller to read the sensor and detect the event which indicates the completion of a revolution. The microcontroller then needs a means of communication this information wirelessly. Before a microcontroller could be selected it was important to determine the wireless communication that would be used because the microcontroller needs to provide an interface between the radio and the sensor.

Several wireless protocols and devices where considered. These include Wi-Fi, Bluetooth, Zigbee, and Nordic Semiconductors nRF24L01+ class of radios. For the Indoor Escape application the range of communications does not need to be very large. In most applications the transmitter and receiver will be within a few feet of each other, at worst it would be on the opposite side of a room. Therefore, a range of about 20 feet will be sufficient. In addition, high communication reliability is not required. If some data is lost, it would not greatly affect the reliability or performance of the system. Sparkfun provides a useful comparison guide of the common wireless protocols and the Nordic wireless devices [29]. This guide was used to create a decision matrix shown in Table 3.2, by using the relative weightings between the wireless devices and applying weights to each of the selection parameters mentioned above. From Table 3.2 it can be seen that the Nordic nRF24L01+, shown as Nordic in Table 3.2, provides the best balance of cost, reliability, range, and power

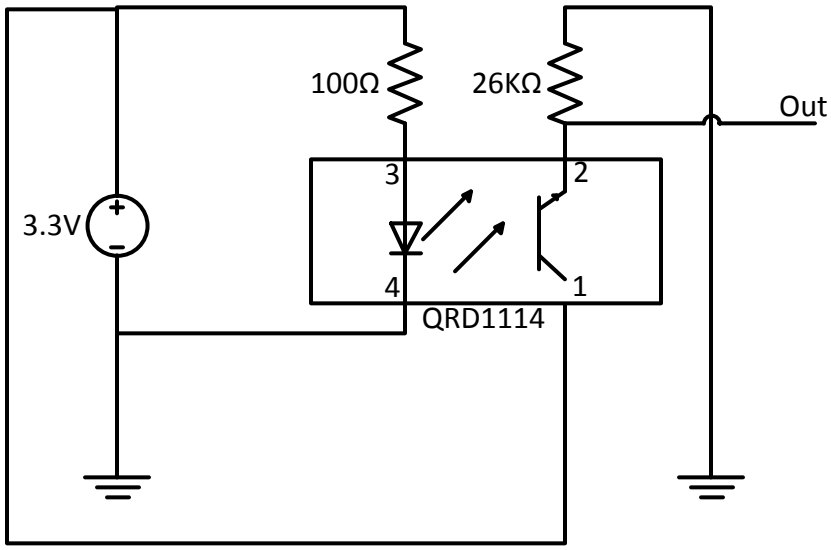

Fig. 3.6: Test circuit for QRD1114.

for the Indoor Escape system.

Nordic semiconductor has three products that use the nRF24L01+ radios; they are the nRF24L01+, nRF24LE1, and nRF24LU1+. The nRF24LE1 is a system on a chip (SOC) that combines an enhanced 8051 microcontroller, the nRF24L01+ radio, a Real-Time Counter (RTC), an ADC and other components. The nRF24LU1+ is another SOC that was designed to create a Universal Serial Bus (USB) adapter for the nRF24L01+ radios. It combines the nRF24L01+ radio with an enhanced 8051 microcontroller, a USB end device controller, and other components such as hardware Universal Asynchronous Receiver/Transmitter (UART) and timers [30]. The final product is the nRF24L01+ which is a radio that provides a serial interface and allows the use of other microcontrollers. The nRF24LE1 was selected because the combined micro controller ADC and radio made it a single chip solution for the sensors that meet the requirements for communication and achieved the goal of low-cost and low power.

The components selected for the three sensors along with their respective costs are listed in Table 3.3. Each sensor will use the nRF24LE1 and a QRD1114, MMA7361, or

Table 3.2: Wireless decision matrix.

	Power Weight:5		Distance Weight:2		Data Rate Weight:1		Data Delivery Weight:3		Cost Weight:4		
Protocol	Relative Rank	Weight Rank	Relative Rank	Weight Rank	Relative Rank	Weight Rank	Relative Rank	Weight Rank	Relative Rank	Weight Rank	Weight Total
Nordic	5	25	1	2	2	2	2	6	5	20	55
Wi-Fi	2	10	3	6	5	5	5	15	2	8	44
Zigbee	4	20	1	2	2	2	2	6	4	16	46
Bluetooth	3	15	3	6	4	4	4	12	3	12	49

SS114P. The current consumption of each of the components for the different modes is given in Table 3.4. From Table 3.3 and Table 3.4 it can be seen that the goals of low power and low-cost can be satisfied for these with the selected components.

One of the biggest disadvantages of using the nRF24LE1 is that the SOC only provides a physical layer for network communications. A wireless protocol will have to be implemented. In addition, a receiver will have to be made that connects to the personal computer. It was decided that a star topology would be supported because it would allow a single receiver to receive data from multiple sensors. Using this with the micro-server described in the next section would make the system much cheaper for a gym to incorporate. It could also potentially reduce the cost for home exercise equipment if more than one exercise machine was used in the home.

3.6 Micro-Server Subsystem Component and Technology Selection

The micro-server's purpose is to provide an interface between the sensors and the display subsystem. It acts as a bridge between the wireless communication protocol of the sensors and the TCP/IP communications used by the display subsystem. To the end user it would be a small program that runs on their computer and relays the sensor information into the web browser. To create the bridge several features are needed. On one end it needs the ability to connect to the ActionScript socket that the display system provides as an interface to receive sensor data. On the other end the micro-server needs the ability to communicate with the wireless receiver.

In addition, the micro-server needs to be able to determine what information to send where. A desirable feature for a gym would be to be able to have a single computer run

Table 3.3: Cost of major components for the sensors.

Component	Cost
nRF24LE1-Q48-T	3.70
QRD1114	1.04
MMA7361LT	2.73
SS411P	0.58

Table 3.4: Current consumption of major components of the sensors for differnt operating modes.

Component	Mode	Current
nRF24LE1	Memory Retention With Timers	1.6μA
nRF24LE1	Active	4mA
nRF24LE1	Transmitting	11.1mA
nRF24LE1	Receiving	13.3mA
QRD1114	As shown in fig. 3.6	19mA
SS114P	NA	5.5mA
MMA7361LT	Active	400μA
MMA7361LT	Sleep	3μA

multiple instances of the Indoor Escape, thereby reducing the number of computers it would need to purchase to add the Indoor Escape to all of its machines. Another model that could be used by gym's would be for them to provide the sensors and micro-server and then the gym members bring in their own laptops. The users would then connect to the Indoor Escape system and the gym's micro-server would provide the sensor information. Using this model, the gym would only need to install a micro-server. Thus the micro-server should be capable of connecting to multiple sensors and multiple display subsystems and have the ability for a display subsystem to receive data from a selected sensor.

It was decided that the connection to the display system would be made by using BSD sockets. The BSD socket interface is compatible with Linux, Windows, and Mac OS X, making it a good solution to create a cross platform micro-server. It was decided, for purposes of this project, that the micro-server would only be developed on Linux. While this limits the demonstration of the micro-server to Linux, the rest of the system can be demonstrated on any operating system by connecting over TCP/IP to the micro-server running on Linux. By only developing the micro-server on Linux, the development time needed to demonstrate the system can be reduced.

To connect to the sensors, it was decided to use the nRF24LE1 SOC and a RS232 serial connection. The nRF24LU1+ was considered, but the requirement to develop drivers for the USB device made this a development intensive choice. For commercialization, the

nRF24LU1+ would probably be the right choice. By using a serial port and the same SOC used for the sensors, the development efforts that would have been put into learning the USB protocol and the nRF24LU1+ were put into enhancing the Indoor Escape system. In addition, the availability of serial to USB converter chips such as FTDI's FT232R [31] and the needed drivers makes the use of a serial port a viable option even for computers without a serial port.

3.7 Schedule

One of the goals in the design of the Indoor Escape system was to have it completed by 05/01/2010. The schedule shown in fig. 3.7 was used to complete the Indoor Escape system. The major deadlines for stages of development were: system and design exploration to be performed from 9/1/09 to 12/4/09 with a preliminary design review on 11/11/09. Implementation of the system would begin on 12/01/09. The implementation would start with the display subsystem then move to the micro-server and then to the sensors. It would conclude with finishing a few components of the display subsystem on 3/31/2010. Thus meeting the May 1st deadline.

3.8 Summary

The Indoor Escape system was designed by dividing the system into three major subsystems. The display subsystem is to be implemented as a web application. It will provide the user interface to the system. This user interface provides the ability to create edit, retrieve, and traverse routes. The display subsystem will use Google's Map API to create routes and the immersive exercise environment. To save user data, the display subsystem will use a MySQL database. A web server will be used to deliver the display subsystem to the user's web browser. The web server will run server side scripts written in PHP and handle all interactions with the web browser. JavaScript will be used as the client side scripting language. Finally, communication to the micro-server subsystem will be provided by a flash application which uses ActionScript's socket class to create a bidirectional TCP/IP socket connection.

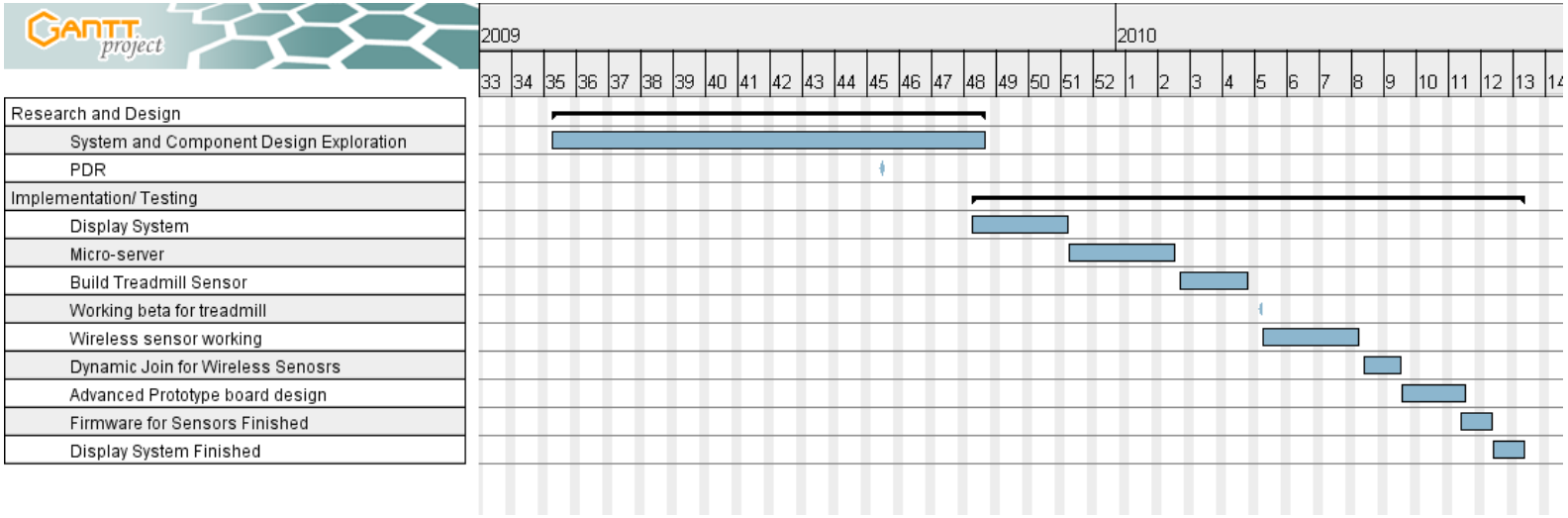

Fig. 3.7: Schedule for completing the Indoor Escape system.

The mirco-server subsystem provides a communication bridge between the sensors and display subsystems. It provides the means to communicate with the display subsystem within the security frame work of a web browser. It will be developed for the Linux operating system for the purposes of demonstration, but written using BSD sockets to ease porting to other platforms. A serial port will be used to communicate with the wireless receiver.

Three sensors and a wireless receiver will be implemented for the Indoor Escape system. All of the sensors will use the nRF24LE1 SOC which provides an enhanced 8051 microcontroller, an nRF24L01+ radio, RTC, and ADC. In addition, the sensing method for each type of exercise equipment was decided upon. The treadmill with use a reflective IR sensor, the QRD1114, and a reflective stripe on the treadmill track to measure the treadmills speed. The elliptical's speed will be determined by using an accelerometer, the MMA7361LT, and measuring the time between zero crossings of the acceleration in the forward direction. Finally, the bicycle's rate of travel will be measured using a Hall-effect sensor, the SS114P, and a magnet attached to the spokes of the bicycle wheel. A detailed block diagram of the complete system is given in fig. 3.8.

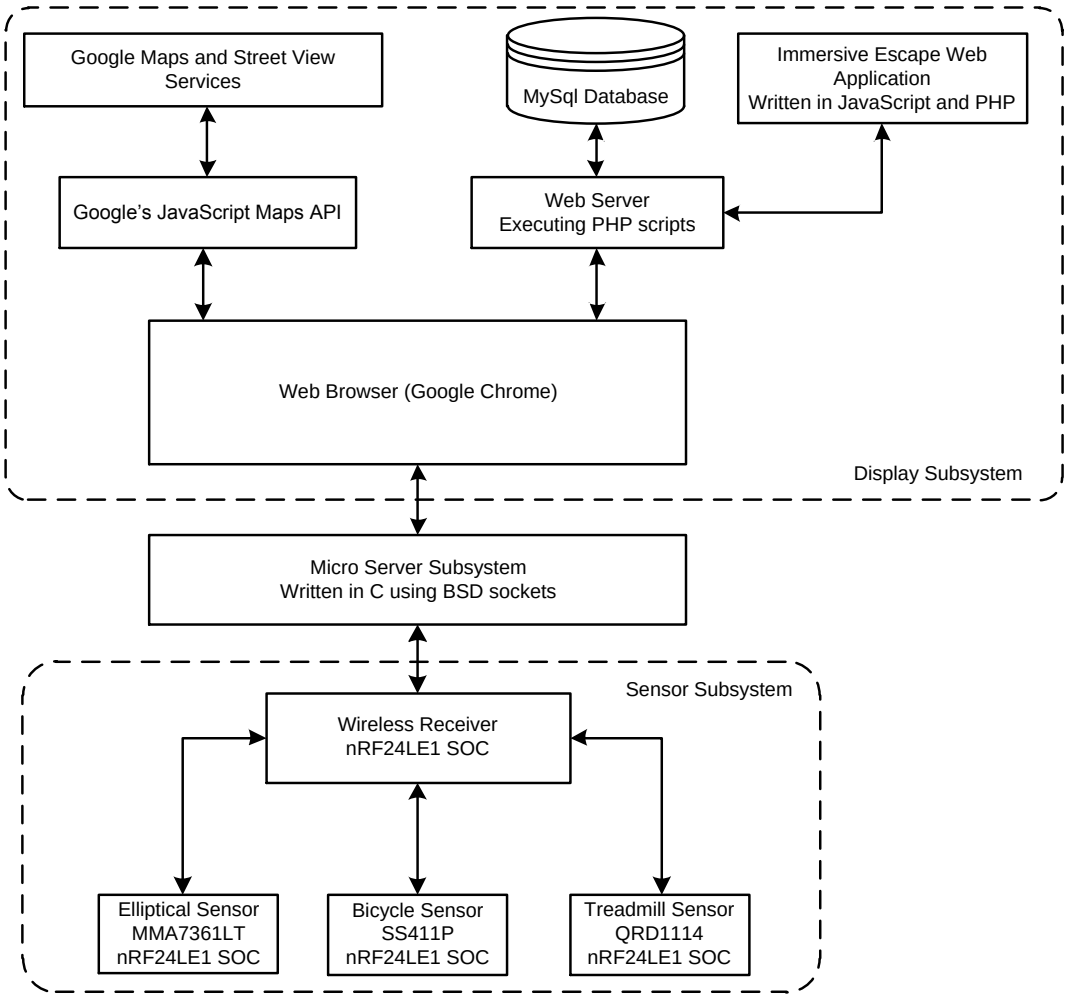

Fig. 3.8: Detailed block diagram of Indoor Escape system.

Chapter 4

Communication Protocols

4.1 Overview

Two communication protocols where defined to enable communication between the three subsystems of the Indoor Escape system. An Application Layer Protocol (ALP) was defined that enables communication between the display subsystem and the micro-server, and between the micro-server and the wireless receiver. This ALP is used on top of RS-232 serial protocol, and the TCP/IP stack. A custom Radio Frequency (RF) protocol was implemented that allows communication between the wireless receiver and the sensors. This protocol was developed to take advantage of the built in features of the nRF24L01+ radio. Figure 4.1 shows the communication interfaces used between the subsystems of the Indoor Escape. This chapter will describe the protocols that were developed.

This chapter makes use of communications diagrams similar to those shown in fig. 4.1. In the diagrams the arrows with text indicate the packet type and the direction of the arrow represent the direction the packet moves. The vertical bar represents the devices in the diagram. Each device is labeled at the bottom. Text on the vertical bar represents action or events performed by the respective device. These actions usually occur in response to either external events or the reception of a packet.

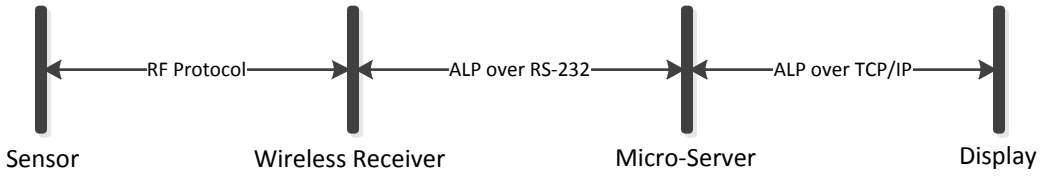

Fig. 4.1: Communication interfaces between the subsystems of the Indoor Escape system.

4.2 Application Layer Protocol

The ALP was implemented to allow communication between the micro-server and the display subsystem, and between the micro-server and wireless sensor. It consists of a packet format that is transmitted over both TCP/IP and RS-232. The packets have six fields that are depicted in fig. 4.2. The first field is a four-byte address that designates the destination of this packet. The next field is a four-byte address of the sender of the packet. The third field is one-byte designating the packet type. The packet type is followed by a one-byte length field. The length is the total length of the packet including all overhead. The fifth field is a 0 to 244 byte data field. The last byte is new line character, which has a hex value of 0x0A. It is used to designate the end of the packet.

The new line character is used to mark the end of a packet because the serial port on the micro-server operates in canonical mode. Which means it reads data one line at a time, and new data is not registered until a new line character is received. Thus, a new line character can only be transmitted at the end of a packet. Since the addresses and type fields are arbitrarily assigned this character is simply avoided. The packet overhead is 11 bytes so the length field will never contain 0x0A. Thus, the only potential problem is the data field. There is only one packet type that is used that sends arbitrary data and it provides a means to delimit the data and will be discussed later.

4.3 RF Protocol

A second protocol was defined for wireless communication. This protocol was designed to take advantage of the unique resources the nRF24L01+ radio provides. The nRF24L01+ is the radio component build into the nRF24LE1 and nRF24LU1+. It is also available as

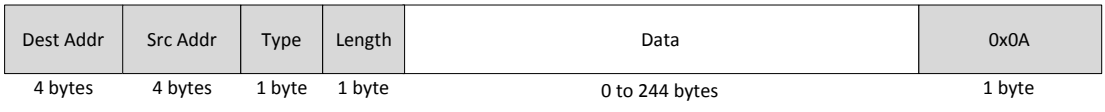

Fig. 4.2: TCP/IP application layer packet format.

a standalone integrated circuit.

The nRF24L01+ radio has a mode of operation that Nordic semiconductor refers to as an enhanced shock burst mode [32]. The enhanced shock burst mode provides International Organization for Standardization (ISO) physical and link layers. These layers take care of packet assembly, transmission, reception, error checking, acknowledgements, and retransmission. All of these features are handled via nRF24L01+ radio without interaction with the microcontroller.

The nRF24L01+ enhanced shock burst operation has two configurations, Primary Transmiter (PTX) and Primary Receiver (PRX). When using the enhanced shock burst mode communication is always initiated by a PTX sending data to a PRX. The PRX is intended to always be listening. Data is transmitted from the PRX to the PTX by piggybacking data on acknowledgements.

To use the enhanced shock burst mode of operation several parameters of the radio must be programmed. For two radios to communicate they must be configured to the same baud rate, frequency band, Cyclic Redundancy Check (CRC) length, and pipe address. Each of these parameters has several options.

The radios are capable of transmitting at 512Kbs, 1Mbs, and 2Mbs. They can operate on 120 different 1MHz wide frequency bands. The spacing required between frequency channels depends on the baud rate. At 2Mbs baud rate there must be at least 2 MHz of separation to avoid interference between channels. The other baud rates require 1 MHz of separation. The CRC can be either 8 or 16 bits.

The pipe address defines a virtual channel within the physical channel. Each PRX has the ability to listen to up to six pipe addresses concurrently. This enables a single PRX to be able to communicate with up to six different PTXs. However, the pipe addresses on the PRX are not independent of each other. The pipe address length is the same for all the pipes, and can be three, four, or five bytes. A further limitation is that pipe addresses 1 through 5 must be the same except the last byte of the address. For example pipe 1 could

have an address of 0x43-0x56-0x76-0x78 and pipe 2 could have and address of 0x43-0x56-0x76-0x54 but not 0x43-0x56-0x34-0x20.

For the RF protocol it was decided to use the 2Mbs band rate. This was selected because it has the shortest on air time; thus, reducing the probability of collisions and power used to transmit data. A four-byte pipe address length was chosen to maintain compatibility with the ALP packet format and a 16-bit CRC was chosen. It was decided to use multiple frequencies and a dynamic method of selecting one.

4.3.1 Frequency Selection

Noise in the environment many prevent a particular frequency band from being usable. Thus, to ensure reliable communication more than one frequency should be available for use. A common approach to solve this problem is frequency hopping. In frequency hopping the receiver and transmitter change frequencies periodically, thus if they are on a noisy frequency they are only there temporarily and communication is only temporarily interrupted. For the communication to occur both the receiver and transmitter must be on the same frequency while the transmitter transmits. To achieve this two methods of frequency hopping are commonly used synchronous and asynchronous.

With synchronous frequency hopping timing information is exchanged to synchronize clocks between the transmitters and receive. When the transmitter wants to send information, it determines the frequency to use based on how much time has passed and the agreed upon hopping pattern. The advantage of synchronous hopping is low latency. Its disadvantage is that timing information has to be exchanged fairly often unless high precision clocks are used. The higher precision the clock the more expensive the system is.

Asynchronous frequency hopping usually has higher latency than synchronous. With asynchronous frequency hopping the transmitter transmits on different frequencies until it finds the receiver. Thus, the same data is transmitted multiple times and it takes a while before the receiver is found.

The approach taken for the RF protocol is mix of the two. It was decided to use eight frequency bands. The receiver listens to a frequency band and changes to a different band

only if it does not receive any data for two seconds. The sensors transmit on every band until it receives an acknowledgement. The next time the sensor transmits, it starts from the frequency that was successful last time. Even in sleep mode the sensors transmit at least once every two seconds. Thus, once data is transmitted successfully both the receiver and sensor will stay on the same frequency providing a low latency link. If that channel where to become noisy and unsuitable for communication the receiver would stop receiving data and change frequencies until it started receiving data gain. By using eight bands spread across the available frequency spectrum, a sufficient degree of noise immunity is obtained. Each receiver would use its own set of frequencies and a means of communicating the frequencies used to the sensors is included in the protocol.

4.3.2 Communication Network

A useful feature of the nRF24L01+ is the ability for it to listen to up to six devices concurrently. The nRF24L01+ thus can support a star topology network. In this network the wireless receiver acts as the hub and each sensor is an end device. For the receiver to communicate with a device it is essential that only one device communicate on a pipe address. If two devices share a pipe address the data sent from the receiver to the devices would get confused. For example, if the receiver has data for device A, and device B transmits to on the same pipe address that A uses, the receiver has no way of knowing that it is device B not A and it will piggy back the data on acknowledgment to B. Thus, each sensor in the network needs a unique pipe address.

An intuitive approach would be to use the sensors unique Identification (ID) as the pipe address, but the receiver cannot listen to any six arbitrary pipe addresses. Therefore, it was decided to give each receiver a set of six pipe addresses that are unique from any other receiver and provide a means for the receiver to assign a pipe address to each sensor in its network. To accomplish this, a pipe address, called the join address, was reserved for adding sensors to a network. In addition, eight frequencies, called the join frequencies, were also designated for the purpose of adding sensors to a network. Each receiver and sensor has the join address and frequencies program into them and they use them to communicate

when in join mode. Join mode allows a sensor to be added to a wireless receiver's network by sending to the sensor a pipe address and the frequencies needed to communicate with receiver.

When a sensor is added to the network the sensor's ID is associated with the pipe address it was assigned. If the network is full and a sensor requests to join the network, the request is ignored unless one of the sensors in the network has not sent data in a long time. In this case, the inactive sensor is removed from the network and the new sensor added.

4.3.3 RF Packet Format

The RF protocol uses a packet format that allows it to take advantage of the enhance shock burst networking layers. The RF packets are designed to be small to reduce the probability of collisions and the power it takes to transmit. The packets have the format shown in fig. 4.3.

The gray blocks indicate data that is automatically added by the enhanced shock burst hardware. The preamble is added to the beginning of the packet so that the receiver can synchronize with the incoming packet. The pipe address defines the virtual channel that is used by the receiver to identify where the data is coming from, and allows the receiver to send data back to the sensor. The packet control field is used to indicate if the packet should be acknowledged. If packet should be acknowleged, the control field contains a count. The count is used by the receiver, to determine if the packet is a new or repeated packet. The sensor ID field is the unique ID given to each sensor and is used to route the packet. The packet type is used tell the receiver what type of packet this is. The low bat field is used to indicate the battery state of the sensor. If the battery is low, this field will be 0x01 otherwise it will be 0x00. The data field contains the data that is transmitted and the last field is a 16-bit CRC that is checked and removed by the enhanced shock burst layer.

4.3.4 RF Protocol Summary

The RF protocol was designed to take advantage of the nRF24L01+ built in features and to allow reliable wireless communication. It uses a star topology network with each

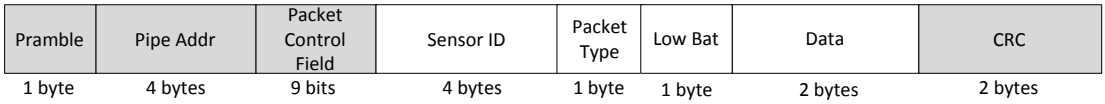

Fig. 4.3: The RF protocol packet format. Gray fields are added by the enhanced shock burst hardware.

receiver being capable of supporting up to six sensors. The receiver listens to eight channels and stays on one channel unless it does not receive data for over two seconds. The receiver also assigns each sensor in its network a pipe address that the sensor uses to communicate with the receiver. The pipe address and frequencies are exchanged with the sensors by temporarily using the join address and frequencies. The packets types were designed to be compatible with the ALP packet types for the Indoor Escape system.

4.4 Command and Data Packet Types

The Indoor Escape system requires that a set of commands and data types must be communicated between its subsystems. To facilitate the communication of the commands and data, packets are used. The packet types that are used are same across both the ALP and the RF protocol. Each device handles the packets it receives differently depending on their type and origin. Each packet type is specified by a one byte character. The command and data packets that are used in the Indoor Escape system are shown in Table 4.1. Table 4.1 includes the packet type's name, its associated ASCII character, and a brief description. Detailed descriptions of each packet and how it is handled are given below.

4.4.1 Join Packet

The join packet has three purposes. It is sent from the display subsystem to the wireless receiver to put the receiver into join mode. It is also sent by a sensor to the wireless receiver to request the information the sensor needs to connect to the wireless sensor. When the wireless receiver is in join mode, and it receives a join packet from a sensor, the receiver replies with a join packet. The data of this packet includes the pipe address and frequencies

Table 4.1: Command and data packet types.

Packet Type	Ascii Char	Description
Join	J	Sent to from display subsystem to PRX to put PRX in join mode
Join Success	s	Sent from PRX to indicate a new sensor has been added. The data field will the Sensor ID of the new sensor.
Sleep	S	Sent by display to put sensor in sleep mode
Time	T	Sent by sensor to display indicating the milliseconds passed since last revolution
Wake Up	W	Sent by display to put sensor into active mode
Awake	a	Send by sensor to display indicate that it is awake
Associate	A	Sent by display to micro-server to associate sensor to display
Unassociate	U	Sent by display to micro-server to remove association between sensor and display
Error	e	Indicates an error has a happened, Usually indicates a requested command carried out

needed to communicate with the wireless receiver when the receiver is in normal operation mode. Table 4.2 shows the data contents and action caused based on the sender and receiver of the packet for the join, join success, and time packets. In Table 4.2, it is assumed that any subsystem between the sender and receiver relays the packet without modification.

4.4.2 Join Success

The join success packet is sent by the sensor to the wireless receiver after the sensor receives the pipe address and frequencies from the wireless receiver. The data of this packet contains the sensor ID of the sensor. The wireless receiver passes the packet along to the micro-server which relays it to the display subsystem. The display system uses the information to add the sensor to a list of available sensors. The join and join success packets allow the adding of sensor to a display subsystem. The sequence of how these packets are exchanged is shown in fig. 4.4.

4.4.3 Sleep Packet

The sleep packet is sent from the display subsystem to a sensor. When the sensor receives the packet it enters sleep mode. The sleep packet has no data.

4.4.4 Time Packet

The time packet is sent by a sensor with the end destination being a display subsystem. The data in the packet indicates the number of milliseconds the most recent revolution took. By knowing the distance traveled per revolution the display system can then calculate the speed of the exercise equipment.

4.4.5 Wake Up Packet

The wake up packet is sent from the display subsystem to a sensor. When the sensor receives this packet it enters active mode. In active mode the sensor measures the amount of time it takes to complete a revolution and transmits this data using a time packet. The wake up packet has no data.

Table 4.2: Join, join success, and time packet data and action caused by source and destination.

Packet Type	Source	Dest	Data	Action Caused
J	Display	Receiver	none	Causes receiver to go to join Mode
J	Receiver	Sensor	Pipe Address and Frequencies	Data Saved to NV memory and join success sent
J	Sensor	Receiver	Sensor ID	Receiver responds with a join packet
s	Sensor	Receiver	Sensor ID	Receiver relays packet to Server and exits join mode
s	Receiver	Display	Sensor ID	Sensor info displayed and added to Database
T	Sensor	Display	ms for last revolution	Varies

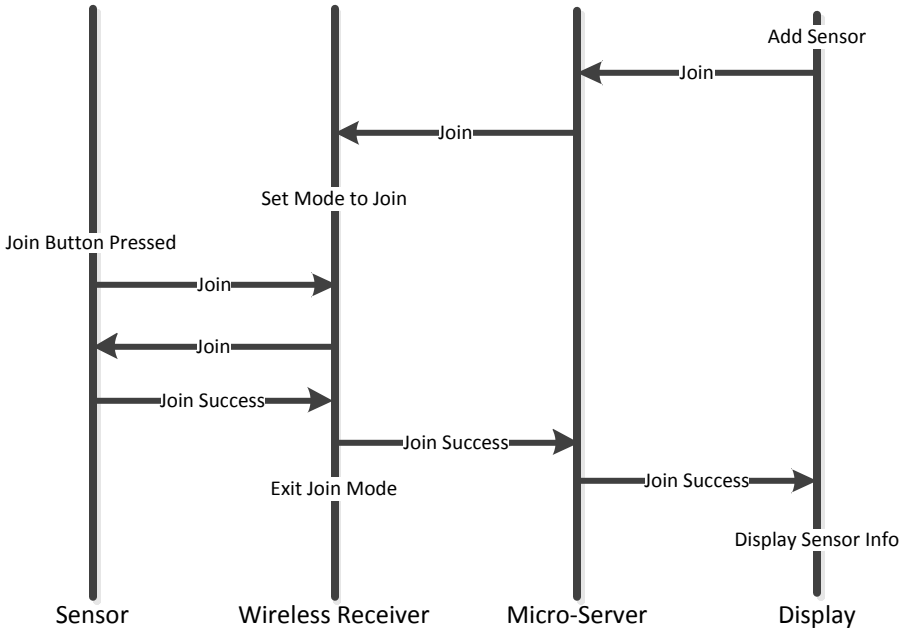

Fig. 4.4: Sequence of packets exchanged to add new sensor to the display subsystem.

4.4.6 Awake Packet

The awake packet is sent from a sensor to a display subsystem. When the sensor is in sleep mode it wakes up about every other second and sends this packet. By doing this it allows the sensor to receive data from the wireless receiver. This is critical because the sensor is put in active mode by receiving a wake up packet that originates from the display subsystem. The awake packet has no meaningful data. The data field is populated with dummy values when sent from a sensor.

4.4.7 Associate Packet

When the micro-server receives an associate packet from a display subsystem, it checks to see if it has received a packet from a sensor with an ID that matches the destination address of the associate packet. If it has, and the sensor is currently unassociated, it associates the sensor with that display subsystem. After associating the sensor with the display, all data received from the sensor will be sent to the display. If the sensor is not in the system or if it is already associated, the micro-server will return an error. If the associate is successful, nothing is returned. The associate packet does not contain data. The sensor that is to be associated is provided as the destination address.

4.4.8 Unassociate Packet

The unassociate packet removes the association of a sensor to a display subsystem. It is sent from the display subsystem to the micro-server. When the micro-server receives the packet it looks for the sensor with the sensor ID that matches the destination address of the unassociate packet and removes the association. The data field of this packet is empty.

4.4.9 Error Packet

Error packets are sent when a command cannot be handled. The data field consists of a one-byte error code followed by a human readable message. Error packets are primarily sent from the micro-server to the display subsystem.

4.5 Packet Translation

The ALP packet types are compatible with the RF protocol types. Translation of the packets is handled by the wireless receiver. The translation from the RF protocol to the TCP/IP application layer protocol will be discussed first followed by the reverse operation.

4.5.1 RF Protocol to TCP/IP Application Layer Protocol Packet Translation

When the wireless receiver receives a packet other than a join from a sensor, it translates that packet into the ALP packet format and sends it to the micro-server via the receiver's serial port. To do this the destination address of the ALP packet is given dummy values. The source address is set to the sensor ID which is obtained from the RF protocol packet. The packet type is taken directly from the RF protocol packet, and the first byte of the ALP data field is the low battery field from the RF protocol. Bytes one and two of the ALP packet data field are the data field of the RF packet. The wireless receiver then calculates the length of the packet and attaches the new line character to the end of the packet.

Packets from the wireless receiver are never intended for the micro-server thus they are either passed along to an associated display subsystem or discarded if the sensor is not associated.

If the wireless receiver receives a join packet over the RF protocol, the packet is either discarded or causes the receiver to send back the frequencies and pipe address the device should use to communicate with the receiver. The later action is only taken if the receiver is in join mode.

4.5.2 Application Layer Protocol to RF Protocol Packet Translation

The wireless receiver receives ALP packets over its serial port. These packets are translated to the RF protocol. The RF protocol packet is populated by setting the sensor ID field equal to the destination address of the ALP packet, the type fields are copied, and the data is ignored. The data is ignored because all of the packets sent to the sensor from the display subsystem are commands and do not contain data. The RF protocol packet could translate an ALP packet with two bytes of data. The two bytes of data would be

sufficient to transmit information such as elevation or slope. The last thing the receiver needs to do is determine what pipe to send the data on. This is obtained by using the sensor ID to find the pipe address that is associated with. It then places the RF packet on the Transmission (TX) First In First Out (FIFO) for that pipe. The hardware of the radio will finish populating the packet and transmit it when it acknowledges the next packet received on that pipe.

Chapter 5

Sensor and Wireless Receiver Implementation

After designing the system each subsystem was implemented independently. The subsystems were also tested independently to the extent possible. This chapter will discuss the implementation and testing of the sensors and wireless receiver. It also discusses design changes made as a result of the testing.

5.1 Development Tools

Nordic semiconductor makes available a set of development kits for their nRF24LE1 SOC's. For the development of the sensors an nRFgo Starter Kit was purchased and two nRF24LE1-F16Q48-DK development kits were purchased. The nRFgo Starter Kit include two mother boards with sockets to connect various nRFgo modules. The motherboards were used to program and debug the sensor modules. Each development kit contains three nRF24LE1 modules. Each module uses the nRF24LE1 SOC with 48 pins. This provides more than enough Input/Output (I/O) ports for this application. Each module provides half of a functioning radio link. Two of the modules in each kit use Printed Circuit Board (PCB) antennas and the other module provides a SMA connector. The development kit also includes five nRF24LE1 part samples and a license for Keil μVision with enables the compiling of programs up to 4KB. It provides example codes and a complete hardware abstraction layer for the nRF24LE1 written in C. Thus, Keil μVision and C were used to develop all of the software on for the sensors and wireless receiver.

The motherboards provided in the starter kit are about five inches by five inches and thus too large to attach to the exercise equipment. Therefore, it was decided that for the prototypes of the sensors small printed circuit boards would need to be designed. To reduce the manufacturing costs, it was decided to design one board that could be used for all three

sensors. The PCB could be designed in two ways: design the PCBs for the nRF24LE1 chip with needed signals to the various sensors, or design the board to use the modules provided in the development kit. The later was chosen because it did not require the designing of radio frequency circuitry. The design of the PCB will be discussed in the following section.

5.2 Sensor PCB Design and Testing

5.2.1 Schematic Design

The sensor PCB was primarily designed to give the radio modules access to the various sensors needed and to provide a form factor that could easily be attached to the exercise equipment. It was not intended to provide a commercially viable design, but rather a platform for developing and testing the proposed sensors. It was also important for time constraints that the PCB be manufactured only once. Therefore, the design of the board needed to be flexible so that if any errors were made in its design that they could be fixed without fabricating another revision of the board. This was accomplished by bringing all the input output ports of the microcontroller out to pin headers along with power and ground. In this way if connections were incorrect they could be cut and then attachment boards made via wire wrapping components and connecting them to the pin headers.

To ease the attachment of the sensors to the exercise equipment it was decided to power the sensors from two AAA batteries. A step-up converter was used to regulate this voltage to 3.3V. This voltage was chosen because there is a wide range of components available to operate at this voltage and all of the components selected for the sensors were capable of operating at this voltage. To help reduce the power consumption of the sensors a voltage controlled switch was included to allow turning off the QRD1114 IR sensor and the SS411P Hall-effect sensor. It was also decided to include two Light Emitting Diodes (LEDs) and two buttons to provide basic interaction with the sensors directly for debugging and I/O options.

The PCB was designed using EAGLE 5.7.0 layout editor from CadSoft. The schematic of the design is shown in fig. 5.1. The schematic can be broken into five different sec-

tions centered on the major components of the design. The first section centers on the L6920DBTR. The L6920DBTR is a high efficiency set up converter from ST microelectronics. This component was selected because of its high efficiency, approximately 90 percent; its sufficient current output, 800mA; and low-cost, 2.70 in single quantities [33]. The L6920DBTR also includes a configurable low battery indicator which is used to help notify the end user of the need to change batteries. The circuit used to configure the L6920DBTR was derived from the datasheet [33]. Jumpers JP1 and JP2 where included in the design of the sensor PCB to allow the measurement of the current required by the sensor PCBs. JP1 allows measurement of the current after the L6920DBTR and JP2 allows the measurement of the current before the L6920DBTR.

The second area of the PCB is focused on the MMA736LT accelerometer. The accelerometer was connected to port 0 of the nRF24LE1 model. Outputs were connected to allow controlling the accelerometer. This enables the ability to select the sensitivity of the sensor and to change between active and sleep modes. In addition, all three axes were routed to ADC inputs of the nRF24LE1 module. 3.3 nF capacitors where selected as filtering capacitors on all of the axes and a 0.1F capacitor was used to decouple the MMA7361LT from the power supply. The values of the capacitors were selected from an example circuit in the MMA7361LT datasheet [28].

The third area of the design includes the QRD1114, the SS411P, and a NC7WB66K8X. The NC7WB66K8X is dual normally open analog switch that can source 128 mA of current and operating at 3.3 volts and has a rise time of about 30ns [34]. It is used to turn on and off the QRD1114 and SS411P.

The fourth area of the PCB is the LEDs and switches. The switches are configured so that they are normally high and pulled low when they are closed. Since the nRF24LE1 can source at most 5mA from a general purpose I/O pin, a transistor is used to switch on and off the LEDs. The LEDs are active high.

The fifth and final area of the board is the pin headers and module connectors. The modules use two female 40 pin M50-432 connectors from Harwin [35]. The matching male

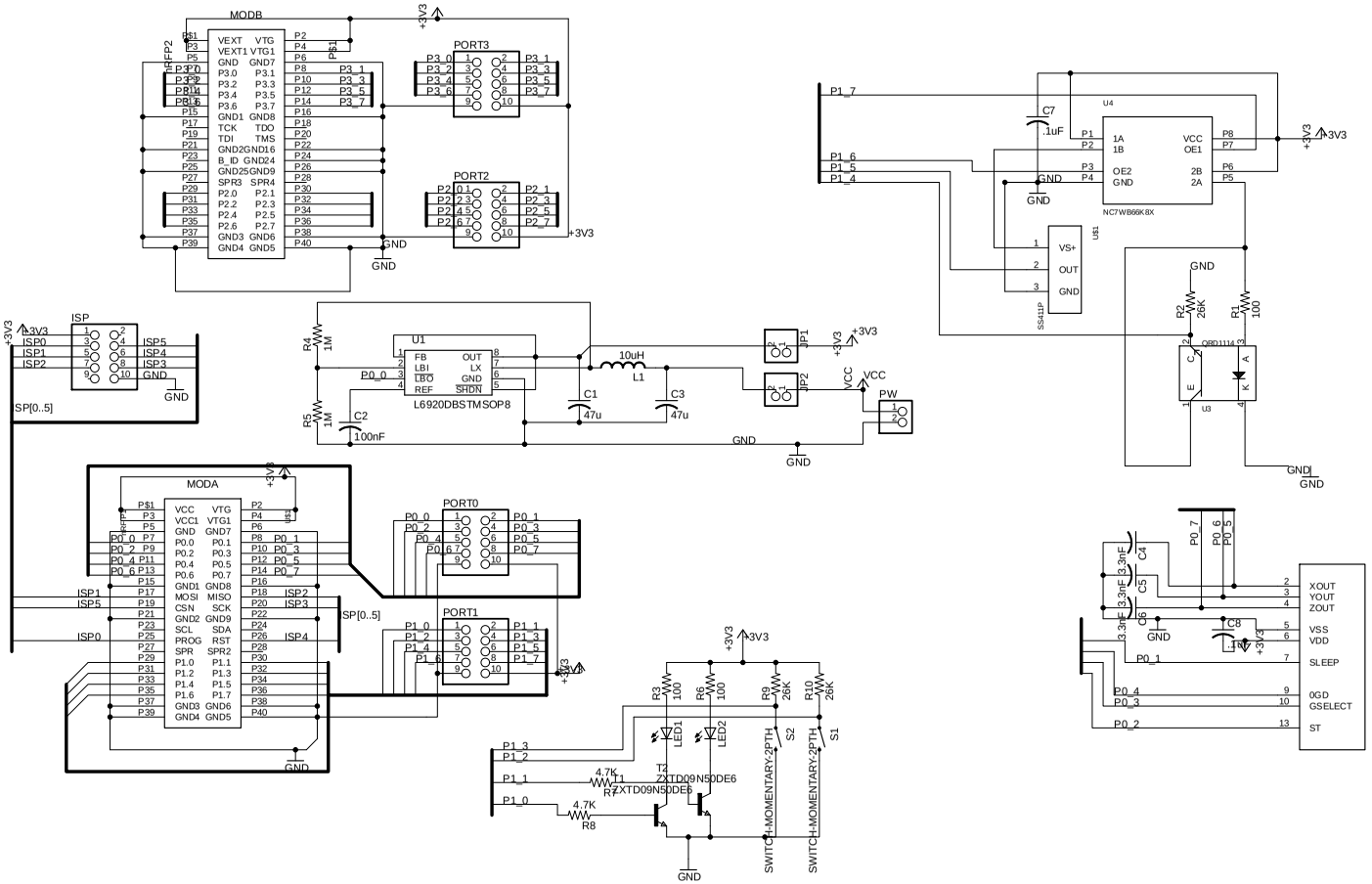

Fig. 5.1: Schematic used to create sensor PCB.

connectors are the M50-492 and the M50-360. The M50-492 contains a plastic guard around the pin which helps in the connection of the module, but was discontinued and replaced with an incompatible part. Therefore, the M50-360 connector was selected. The appropriate connections where made to the modules by using the pin out provided in the documentation provided with the development kit [36]. Five 10 pin 0.1" pitch headers where included on the board. Each of the four general purpose I/O ports were routed to the header. Power and ground are also available at each header. These ports allow the prototype board to be easily extended. In addition it allows the sensors attached to the board to be tested independently from the nRF24LE1. The fifth header is an In-Circuit Programming (ISP) port. This port allows the module to be programmed while attached to the prototype PCB.

5.2.2 Printed Circuit Board Layout

After creating the schematic shown in fig. 5.1, the board was laid out using EAGLE. A custom parts library along with a library available from SparkFun [37] and libraries included with EAGLE were used to create the layout for the PCB. The created parts library and the Eagle design files are available in the Appendix. The most crucial layout constraint was to get the spacing of the two connectors for the nRF24LE1 modules correct. The development kit comes with the complete module layout. These layout files were used to match the spacing of the connectors. It was also decided to use surface mount discrete components to reduce the size of the board needed. Wherever possible, passive parts were selected in the 1206 package. This size was selected because it is much smaller than leaded components and is large enough to allow for easy hand soldering. The board was designed with 8mil min feature sizes and 10mil spacing. The only exception to these was between Surface Mount Device (SMD) pads where 6mil spacing was used. This was because some of the components had 6mil spacing between their pads.

Five boards where ordered from Advanced Circuits using their $33 each special [38]. Only three boards where needed, but there was a minimum quantity of 4 and they included the 5th board free of charge. Having five boards also prevented the need to order the boards again if one was damaged during assembly. The special allows 2-layer PCB with minimum

spacing of 6 mil, up to 60 square inches, Green solder mask, and top and bottom silk screens. The final board was 3 inches by 2 inches. Figure 5.2 shows the layout of the components on the board.

The PCB uses 21 unique components with a total cost of $27.89. To have the PCBs manufactured cost $33.00, and the nRF24LE1 modules cost $33.00 each, making the total cost per sensor $93.89. This is the cost of the prototype board if the sensors were to be produced commercially this price would be significantly reduced. The PCBs could be designed to be much smaller and in mass production become very cheap. The component cost will also drop with volume. In addition, the pin headers and connector for the nRF24LE1 module account for over $9.00 of the components. Finally, the cost of the sensors would be reduced because the nRF24LE1 modules would not be used. The nRF24LE1 costs $3.70 in single quantities and requires only a few external components to make work. The complete bill of materials for the sensors is included in the Appendix.

5.2.3 PCB Assembly and Testing

After receiving the PCBs, three boards were assembled using a fine-point soldering iron and heat gun. Figure 5.3 shows an assembled board. After assembling each board the boards where tested. The first test was the power supply circuit. This was done by inserting batteries into the battery holder and then measuring between the terminals on the board using a multi-meter. After ensuring that power batteries were properly connected JP2 was closed. This activated the L6920LBTR step-up converter. The output of the step-up converter was then measured using an oscilloscope. From fig. 5.4 it can be seen that the output of the step-up converter has a mean value near 3.3V, but it contains a saw wave. The frequency of the saw wave varies depending upon the power being drawn from the step-up converter. Later testing showed the board functioned satisfactory, but this noise in the supply voltage adversely affected sensor readings. After testing the step-up converter, the other components on the board where tested.

This testing was done using an independent power supply. The pin headers were used to test the sensors, the LEDs, and the buttons. This was done by using a breadboard and a

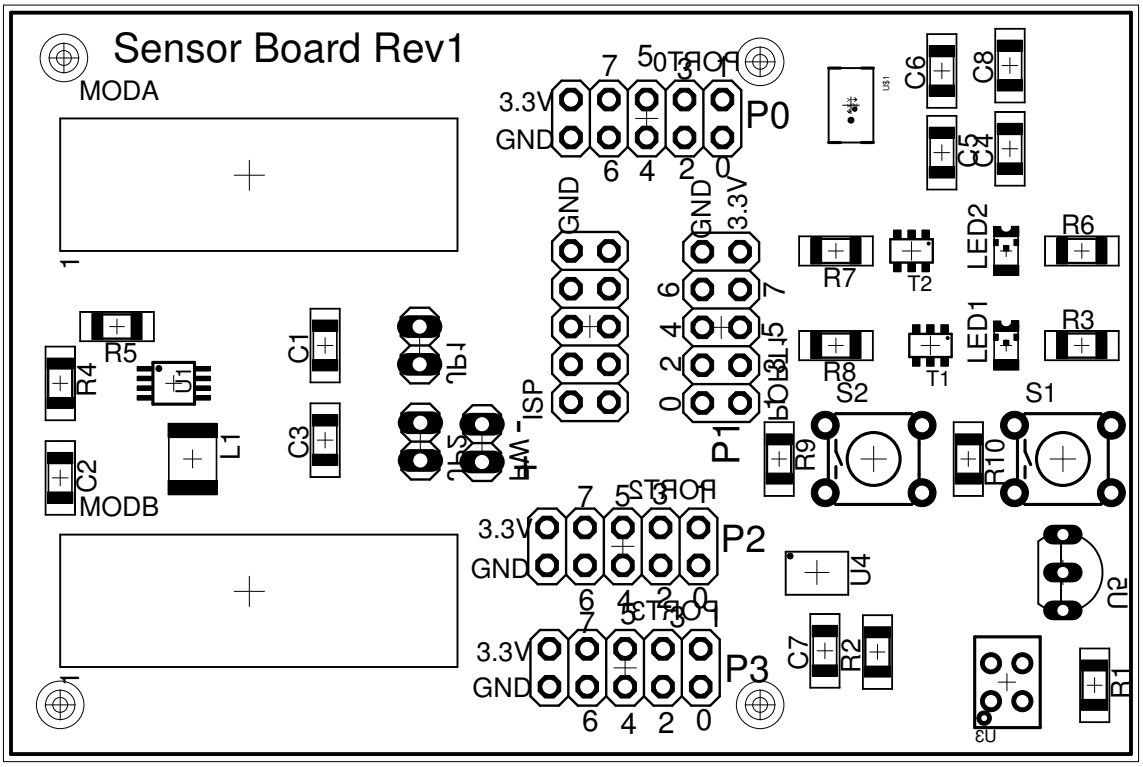

Fig. 5.2: Layout of sensor PCB showing position of components.

ten-pin patch cable. Connections were then made in the breadboard to activate and measure the response of the various components on the PCB. To test the accelerometer, P0.2 and P0.3 were connected to ground and P0.1 was set high. This set the accelerometer to active with 1.5G sensitivity. The axes where then measured one by one using an oscilloscope as the sensor was rotated. By rotating the measured axis so it pointed toward the earth a value of 1G should be detected. On two of the three assembled PCBs, the accelerometer functioned properly. The third board it appeared that the accelerometer did not power up. This was probably caused by either connections not being made properly while soldering or damage caused during soldering. Since the accelerometer is only needed for the elliptical, it was decided that one of the boards with a functioning accelerometer would be used for the elliptical and the problem was not corrected.

While the accelerometer worked on two of the boards, its signal was rather noisy. To reduce the noise, the filtering capacitor on the y-axis of the board used for the accelerometer

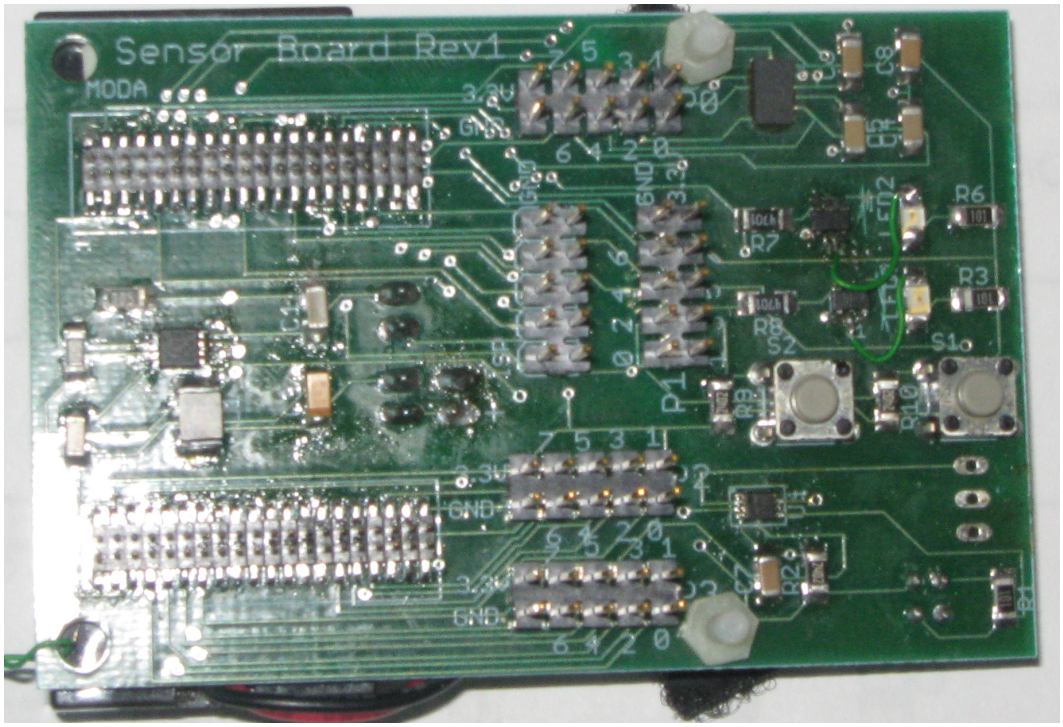

Fig. 5.3: Assembled PCB.

was changed to a 0.1μF. Figure 5.5 shows the output of the y-axis of the accelerometer when attached to the elliptical using a 3.3nF capacitor. By changing to a 0.1μF capacitor, the noise in the signal was significantly reduced as shown in fig. 5.6. Both fig. 5.5 and fig. 5.6 were obtained with the sensor attached to the elliptical and the elliptical traveling at about 60 RPM. It can be seen that this generates about a 1Hz sine wave verifying the method of sensing the speed of the elliptical.

The QRD1114 IR sensor was tested by powering it on by connecting P1.6 to power and then measuring P1.4 using an oscilloscope. A box with aluminum tape attached was then moved past the sensor. On all three boards the IR sensors detected the moving of the box in front of the sensor. P1.6 was then connected to ground, turning off the IR sensor. In all cases the sensor output measured by the oscilloscope fell to zero, thus the IR sensor on all three boards turned off and on, and functioned as expected when turned on.

The Hall-effect sensor was tested in a similar manner to the IR sensor. Except P1.7 was used to turn it off and a magnet was passed by the sensor. During all testing, the

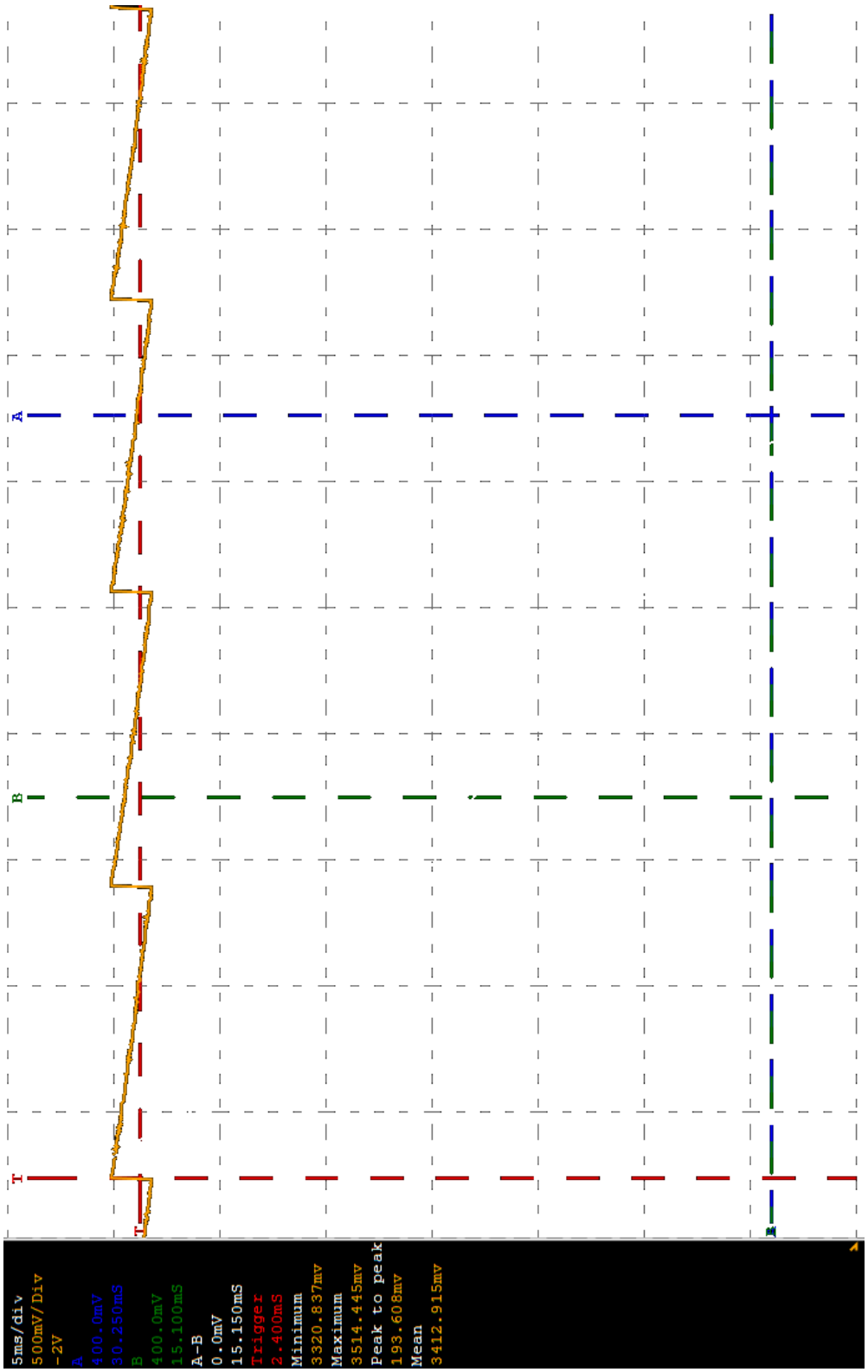

Fig. 5.4: Oscilloscope screen shot of noise in output of L6920LBTR.

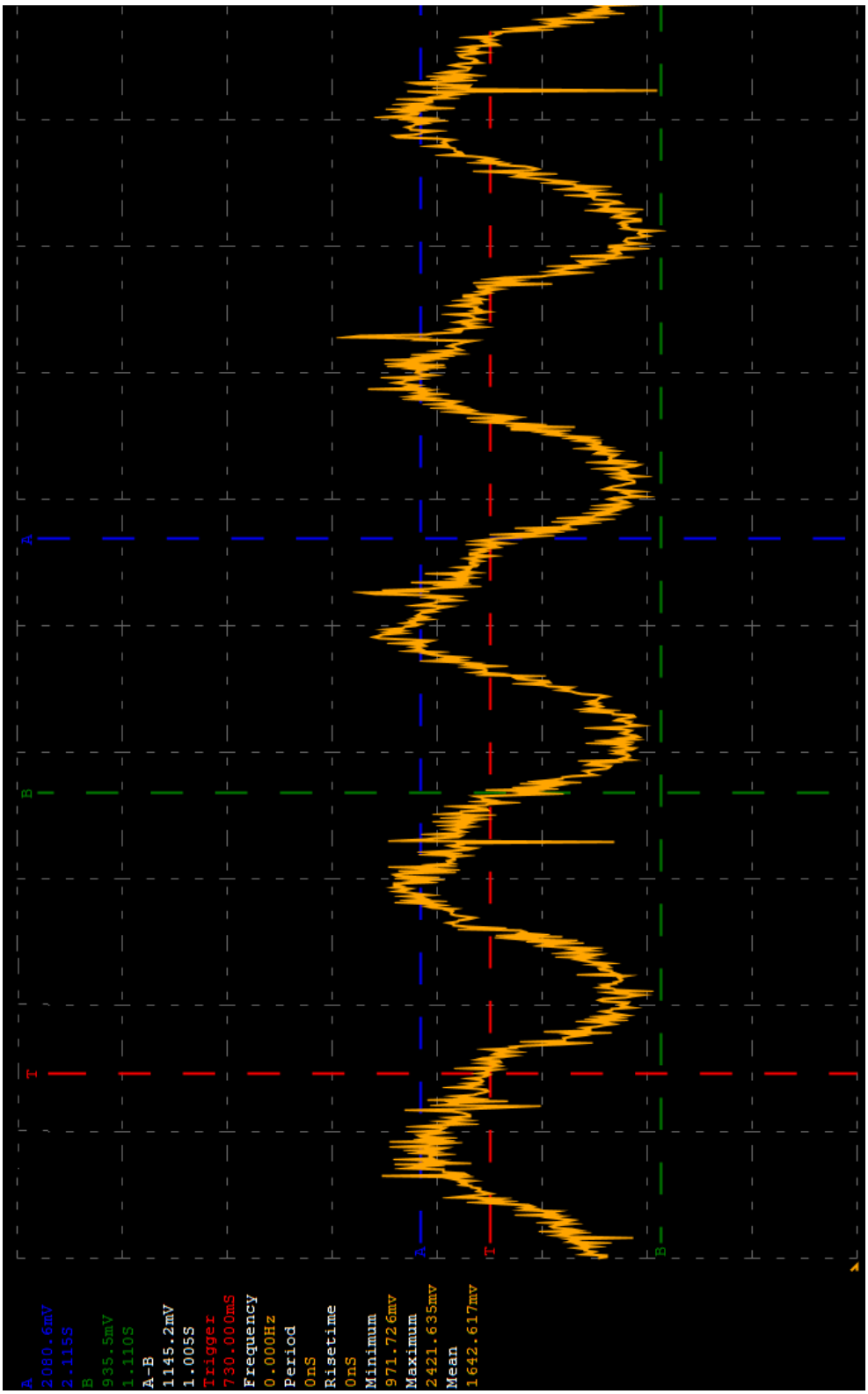

Fig. 5.5: Y-axis of accelerometer with 3.3nF capacitor.

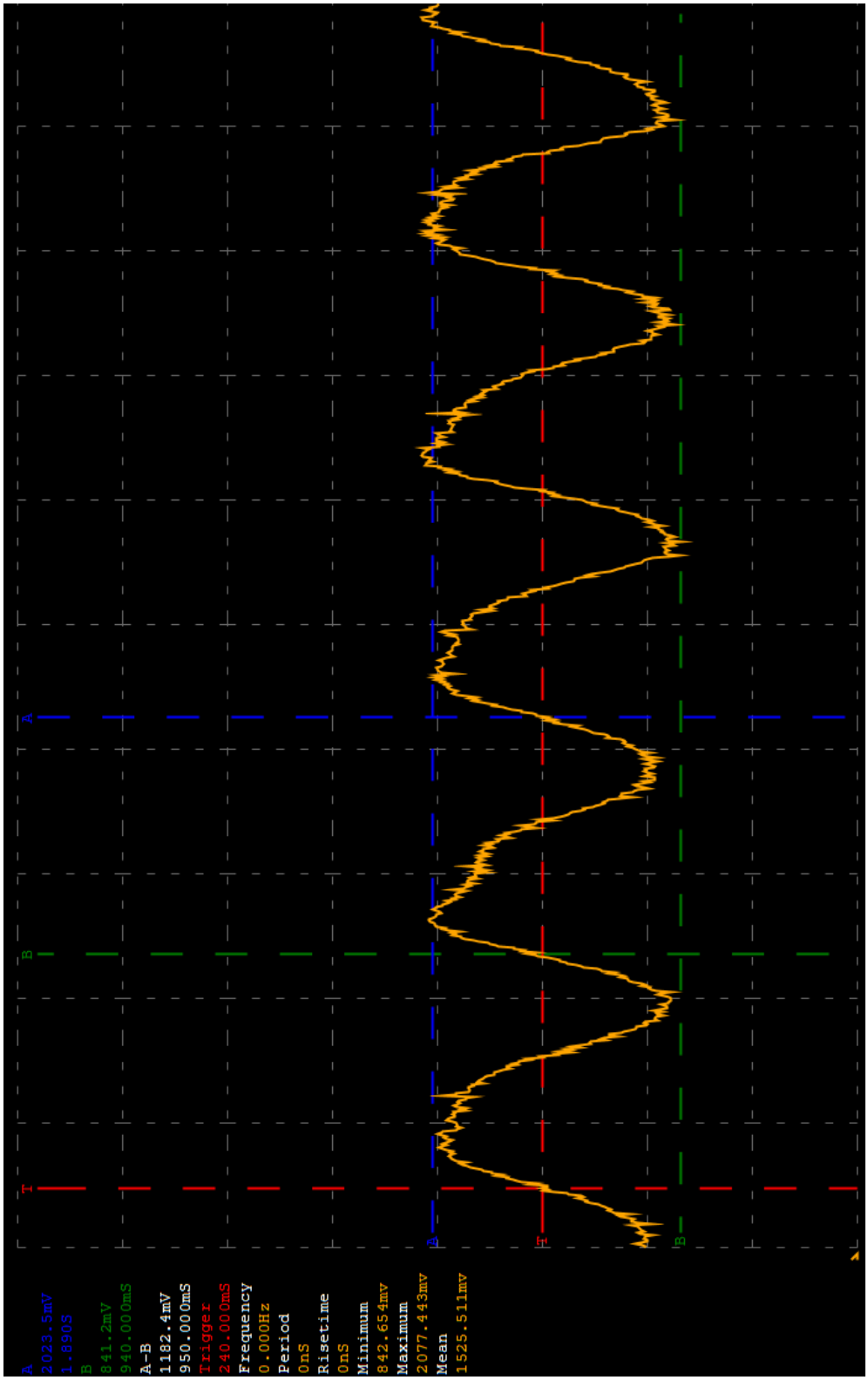

Fig. 5.6: Y-axis of accelerometer with 0.1μF capitor.

output of the sensor read zero. After consulting the datasheet, it was discovered that the ground and output pins had been switched in the layout. Thus, the output was attached to ground and the ground pin was floating. To correct the problem, it was decided to use wires and a socket to connect the part allowing the two pins to be swapped. This correction was only made to one of the boards as the Hall-effect sensor is only used on the bicycle. This also provided the advantage of allowing flexibility in placing the board on the bicycle while still allowing the sensor to be close enough to detect the passing of the magnet attached to the spokes of the bicycle.

5.3 Sensor and Wireless Receiver Software

After testing the PCBs, software was written to enable the sensor to detect the rate of travel of the exercise equipment. All software for the sensors and wireless receiver was written in C using Keil μVision 4. The μVision projects for all of the sensors and the wireless receiver can be found in the Appendix. This section describes the software that was implemented for the sensors and wireless receiver. The implementation of the wireless receiver is discussed first. Then the common elements of the sensor software will be discussed and this section will conclude with a discussion of the specifics of each sensor.

5.3.1 Wireless Receiver Software

Frequency Hopping Test

Prior to implementing the wireless receiver, testing of the frequency hopping pattern was performed. This testing was done by writing a program for a transmitter to send a packet at regular intervals. Each packet consisted of a count value; a receiver was then programmed to receive the packet using the modified frequency hopping pattern described in Chapter 4. It would then increment its own counter and send a packet to the transmitter. When either device received a packet it would write the packet out its serial port. A program Real-Term was then used to record all the data received on the serial ports. The data was then analyzed to see how many packets were dropped.

Several parameters were varied to determine the best values to use in the hopping. These parameters are the time between the interval at which packets where sent (Interval), and the number of times a packet was send on a frequency before changing frequencies (Auto Retries). For all the tests the timeout interval at which the receiver would change the frequency it was listening to was 500ms. The tests also used eight channels, and the delay between retries on the same channel (ARD) was 250s. The actual implementation of the protocol uses a receiver timeout interval of 2 seconds. The interval was shortened to reduce testing time. The timeout interval does not matter as much as the ratio between the time out interval and the transmission interval. Table 5.1 shows that the transmitting and receiving of packets using the described hopping works very well. In all of the tests every packet that was sent was received.

Receiver Software

The receiver is responsible to provide a wireless connection between the micro-server and the sensors. Thus, it needs to be able to receive date from both the micro-server and the sensors. The receiver is implemented using an nRF24LE1 module from the developer kit plugged into the nRFGo motherboard. The communication between the micro-server and the receiver is accomplished using a RS-232 connection running at 38400bps. The desire was to use 57600bps, but the baud rate generator on the nRF24LE1 was unable to derive an accurate enough baud rate for reliable communication at this speed. The 38400bps provided a good match and testing showed that it performed reliably. Communication to the sensors is provided using the RF protocol described above.

The software on the receiver constantly checks to see if data has been received via either the serial port or the radio. After receiving the packets several algorithms are used. When a packet is received from the radio, Algorithm 5.1 is used to handle the packet. Algorithm 5.1 first checks to see if the packet is a join packet. If it is and the receiver is in join mode, it identifies an available pipe to assign the sensor to. It then associates the sensor that sent the join packet with that pipe and sends the pipe address and frequencies to the sensors. If the receiver is not in join mode when it receives a join packet, the packet is ignored. If the

Table 5.1: Frequency hopping test results.

Interval (ms)	Auto Retries	Packets TX-RX	Percent Received	Packets RX-TX	Percent Received
125	3	4780	100	4780	100
125	2	4949	100	4949	100
500	2	65535	100	65535	100
1400	2	7674	100	7674	100

packet type is a join success and the receiver is in join mode the packet is converted to the ALP format and sent to the micro-server. If the packet is any type besides a join or join success it is forwarded to the micro-server after converting it to the ALP format.

When the wireless receiver gets a packet from its serial port, Algorithm 5.2 is used. This algorithm checks to see if the packet type is a join packet. If it is it sets the receiver to join mode, otherwise it checks to see if there is a pipe that is associated with a sensor ID that matches the destination address of the packet. If there is a pipe associated, the packet is converted to the RF protocol format and placed on the TX FIFO for that pipe.

The receiver uses two more algorithms. Algorithm 5.3 is used to change modes, and Algorthm 5.4 is an Interupt Service Routine (ISR) that is called every two seconds. It is used to keep track of the activity of sensors that are associated to pipes, and to timeout a join. If a sensor is inactive for over a minute the pipe is marked as available and will be given to another sensor if no other pipes are available when a new sensor requests to join the network. The wireless receiver was tested after implementing the sensors. Its testing will be described later in this chapter.

5.3.2 Common Sensor Software

The operation of all of the sensors is very similar. Each type of exercise equipment uses a different type of sensor, and thus some variations in the software are required to acquire and interpret the sensor readings. All of the sensors support at least three modes, sleep, active, and join. In sleep mode the sensors go to a low-power state (memory retention mode, timers on) and wakes up from a RTC about every 2 seconds. When the sensors

Algorithm 5.1 Receiver process RF packet.

Input:
Received Packet $Packet$,
Mode m
Output:
Received Data Flag $Flag$,
Mode m
Begin
$Flag = true$
Switch$\{Packet_{Type}\}$
Begin
Case JOIN:
Begin
If$\{m = JOIN\}$
Begin
Find Available Pipe
Associate Available Pipe with $Packet_{SensorID}$
Send Pipe Address and Frequency List Over Radio
End
End
Case JOIN SUCCESS:
Begin
If$\{m = JOIN\}$
Begin
Find Available Pipe
Associate Available Pipe with $Packet_{SensorID}$
Send Pipe Address and Frequency List Over Radio
End
End
Default:
Begin
Translate $Packet$ to Serial Protocol
Send Translated $Packet$ on Serial Port
End
End
End

Algorithm 5.2 Process serial packet.

Input:
Received Packet $Packet$,
List of Sensor Ids Associated to Pipes $List$
Output:
Mode m,
Begin
If{$Packet_{Type}$ = JOIN}
Begin
m =SetMode(JOIN)
End
Else
Begin
For Each{Sensor ID in $List$ **as** $sensorId$}
Begin
If{$sensorId = Packet_{DestId}$}
Begin
Convert $Packet$ to RF Protocol as $RFPacket$
Place $RFPacket$ on FIFO for Pipe Associated with $sensorId$
End
End
End
End

Algorithm 5.3 Receiver set mode.

Input:
Mode m,
Frequency Table Pointer $freqPtr$,
Output:
Mode m,
Begin
If{m = JOIN}
Begin
Set $freqPtr$ to Join Frequency List
Change Radio Frequency to First Frequency in Join Freq List
m =JOIN
End
Else
Begin
Set $freqPtr$ to Regular Frequency List
Change Radio Frequency to First Frequency in Regular Freq List
m = ACTIVE
End
End

wake up a single awake packet is transmitted. This enables the sensor to be woken up from commands issued in the display subsystem, while using minimal power consumption. Memory retention with timers on uses 1.8μA of current [32]. When the system wakes up from this mode it causes a system reset. If the sensor receives a wake up packet the sensor will go to active mode.

In active mode the sensor sets up an interrupt that samples an analog input looking for an event indicating a revolution has completed. It measures how long, in milliseconds, it takes to complete a revolution. After every revolution, it updates the time the revolution took. The RTC is used to generate a second interrupt that wakes up the sensor and transmits the time the last revolution took. In active mode the RTC is set to trigger every half second. When none of the interrupts are triggered the sensor enters a standby mode. This mode allows the microcontroller to be woken up from the ADC and requires 1 mA of current.

The third common mode is join. This mode is entered by pressing the join button on

Algorithm 5.4 Receiver RTC ISR.

Input:
Received Data Flag $Flag$,
Mode m,
List of Frequencies $List$,
Array of Pipe Structures $Pipes$,
Elapsed Time $Time$

Output:
Received Data Flag $Flag$,
Mode m, Array of Pipe Structures $Pipes$,
Elapsed Time $Time$,

Begin
If$\{Flag = false\}$
Begin
Set Radio Frequency to next Frequency in $List$
End
If$\{m = JOIN\}$
Begin
$Time = Time + 1$ /* Increments Every 2 seconds*/
If$\{Time > 15\}$ /* Join Mode times out after 30sec */
Begin
m =SetMode(ACTIVE)
End
End
Else
Begin
For Each{Structure in $Pipes$ **as** $pipe$}
Begin
$Pipe_{Count} = Pipe_{Count} + 1$
If$\{Pipe_{Count} > 30\}$
Begin
$Pipe_{Available} = True$ /*Sensor inactive for 1 min */
End
End
End
$Flag = false$
End

the sensor. This mode allows the sensor to be connected to a receiver. To accomplish this, the sensor switches to the join frequencies and address. The receiver should also be in join mode causing it to use the same frequencies and address. The RTC is then used to wake the sensor up 4 times a second and transmit a join packet. The receiver should respond with a join packet that will have the frequencies and address that this sensor should use to communicate with the receiver. The sensor sends back a join success packet. It stores the address and frequencies in nonvolatile memory and then changes the radio to use those frequencies and addresses. It then goes to active mode. These three modes comprise the essential functions of the sensors. Some of the more involved algorithms will be discussed here. The complete source code for each sensor is included in the Appendix.

The implementation of the software was complicated by the use of the different low-power modes on the nRF24LE1. One of the biggest challenges was maintaining constancy with the I/O ports of the microcontroller. When the microcontroller is put into a low-power mode besides standby, the I/O port registers are reset. This includes the registers that setup the direction and configuration of the ports. For the ports to maintain their current state while in a sleep mode an output latch must be set. But setting the latch makes it possible for the I/O port registers and the outputs of the pins to contain different values. Also if the latch is open before the registers are reprogrammed, the outputs will be changed. This could cause potential problems with sensors being turned on or off when they should not be. To maintain consistency between the pin values and the registers the Algorithm 5.5 is used to put the microcontroller into its low-power state and Algorithm 5.6 is called immediately after the microcontroller wakes. For the other common algorithms including the transmitter side of the RF protocol the reader is referred to the Appendix.

5.3.3 Treadmill Sensor

Software

The most important routine of the treadmill sensor is the interrupt that handles the ADC. The software initializes the ADC to sample at 4Khz. This allows the treadmill to

Algorithm 5.5 Sensor power down.

Input:
Micro Controller Sleep Mode *sleep*
Output:
Saved State of Port0 *Port*0, /* Port Variables are stored */
Saved State of Port1 *Port*1, /* in data retentive memory data */
Saved State of Port2 *Port*2, /* retentive memory maintains it */
Saved State of Port3 *Port*3 /*data in all sleep modes */
Begin
Store data of I/O Port0 into *Port*0
Store data of I/O Port1 into *Port*1
Store data of I/O Port2 into *Port*2
Store data of I/O Port3 into *Port*3
Set Output Latch
Set microcontroller state to *sleep*
End

Algorithm 5.6 Sensor open latch.

Input:
Port0 Last Value *port*0, /* Port Variables are stored in data retentive */
Port1 Last Value *port*1, /* memory data retentive memory maintains */
Port2 Last Value *port*2, /* it data in all sleep modes */
Port3 Last Value *port*3
Output:
Begin
Set Pin Direction of Each Port
Set Pin configurations of Each Port /*i.e pullup resistors, analog input */

Write *port*0 to I/O Port 0 /* These write the registers connected to the */
Write *port*1 to I/O Port 1 /* pins of the microcontroller. The output will */
Write *port*2 to I/O Port 2 /* not change until the latch is opened */
Write *port*3 to I/O Port 3
Open Output Latch
End

detect a 1 cm reflective strip moving at 20 meters per second, which well exceeds the speed of treadmills. Most of Nordic Tracks current offering of treadmill have a top speed of 5.4 meters per second (12mph) [39]. Algorithm 5.7 is run during sensor sampling. The algorithm samples the IR sensor and determines if the value is above a threshold. If is above the threshold then it is assumes that the reflective strip is under the sensor. It then outputs the time that has elapsed since the last crossing. The sensor then goes to sleep. The elapsed time is broadcast when the RTC wakes up the microcontroller.

For accurate speed measurement, it is essential that the threshold be set correctly. The values measured by the ADC are dependent on several factors: the distance of the IR sensor from the track, the speed of the treadmill, and variations in the reflectivity of the track. Since the distance between the track and the sensor can be heavily influenced by the installation of the sensor, treadmill sensors have the ability to determine this threshold after they are installed. This is accomplished by including a calibration mode. To activate the calibration the user starts the treadmill at a speed so that it makes at least one revolution per second and half. For most treadmills this is greater than about 2 meters per second (4.2 mph). The calibration uses the knowledge that the reflective stripe will be a very small portion of the total track. It sets the threshold to about 0.125 V and then for the next 1.5 seconds (6000 samples) counts the number of values below the threshold. If the number of counts is greater than 5950, then it keeps that threshold; if not, then it increases the threshold by another 0.125 Volts. This continues until either a satisfactory threshold is found or every threshold has been tried. Requiring 5950 samples out of 6000 be below the threshold, limits the width of the stripe to about 0.83 percent of the track. For a track length of 1.4m the stripe must be less than 2.3cm (track length is half the belt length thus 2.3cm is about 0.83% of 2.8m).

Algorithm 5.7 is used to determine both the speed of the treadmill and calculate the threshold. The algorithm uses five inputs. The first is the current mode the sensor is in. This can be either Active or Calibrate. The current threshold to use is input. Two variables used to handle transient readings are also input. Ideally, the sensor would give low values

Algorithm 5.7 Treadmill ADC ISR.

Input:

Mode m,
Threshold Value *threshold* ,
Number of Low Samples before High Again *numLow*,
Number of Consecutive Low Sample *adcLowCount*,
Elapsed Time *time*

Output:

Threshold Value *threshold*,
Milliseconds Since Last Crossing *crossingTime*,
Number of Consecutive Low Sample *adcLowCount*,
Elapsed Time *time* ,

Begin

Open Latch
Read ADC into *adcOut*
$time = time + 1$
If$\{m =$ ACTIVE$\}$
Begin
If$\{adcOut > threshold$ **and** $adcLowCount > 50\}$
Begin
$crossingTime = time \div 4$ /*ADC samples at 4KHz*/
$time = 0$
End
If$\{time > 0XFFFF\}$ /* hasn't detect a crossing in ∼16 sec*/
Begin
Set Mode to SLEEP
End
ElseIf$\{time > 36000\}$ /* ∼9 sec*/
Begin
$crossingTime = 0$ /*Indicates speed is zero */
End

If$\{adcOut > threshold\}$
Begin
$adcLowCount = 0$
End
Else
Begin
$adcLowCount = adcLowCount + 1$
End
End

algorithm continued below

Algorithm 5.7 cont. Treadmill ADC ISR.

```
    ElseIf{m =CAL}
    Begin
        If{adcOut < threshold}
        Begin
            adcLowCount = adcLowCount + 1
        End If{time < 6000}                          /* sampled for1.5 seconds */
        Begin
            time = 0
            If{adcLowCount > 5950}                   /* high < 125ms */
            Begin
                Write threshold to Flash Memory
                Set Mode to Calibrate Done
            End
            Else
            Begin
                If{threshold + 10 ≤ 255}             /* high < 125ms */
                Begin
                    threshold = threshold + 10
                End
                Else
                Begin
                    Set Mode to Calibrate Done
                End
            End
        adcLowCount = 0
        End
    End
End
```

then when the stripe passed under the sensor it would go high and stay high until the stripe passed. In reality, the sensor is noisy and may cross the threshold multiple times on both the rising and falling edges.

Therefore, before the sensor can detect a rising edge, it must read the sensor as being low for a specified number of consecutive readings. The first variable is the number of consecutive readings required; the second is the number of consecutive low samples that have been read. This number of consecutive readings is 50 which requires 12.5ms of low readings. This number was determined to work effectively and does not reduce the maximum

speed that can be detected by the sensor. The final input is a counter that is incremented every time the Algorithm 5.7 is called. This counter keeps track of the elapsed time with each count being 0.25ms.

Testing

Each mode of the treadmill sensor was tested to ensure that it worked properly. This was done in connection with the receiver. For testing, the receiver was connected to a serial port and a terminal program was used to send and receive data from the receiver. The sensor was then connected to the receiver using the dynamic join. After the join was successfully working it was checked to see if the sensor operated correctly. The rest of the treadmills sensors abilities were tested. It was determined that the sensor behaved as expected in each of its operating modes and responded to the necessary commands. In addition, it was able to reliably measure the speed of the treadmill, using the threshold determined by using the built in calibration method.

Power measurements were obtained for each of its operating modes, both before the step-up converter, by applying power with a 1 ohm resistor connected to the respective JP2 pins. The voltage across the resistor was then measured using an oscilloscope.

Measurements were obtained when the sensor was in active and sleep modes. Each of these modes has two states idle and transmitting. For active mode the idle state still reads the sensors, in sleep mode the sensor shuts everything off except the RTC. The current consumption was then measured for each mode and state combination. The total average current consumption was then estimated by assuming that the sensor was in active mode for one hour each day and sleeping the rest of the time. During all testing no receiver was active thus the transmissions all timed out and the sensor was transmitting as long as possible. Table 5.2 shows the results of the power measurement along with the estimated number of days a 1100mAh battery would last. 1100mAh is a common amp hour rating of alkaline AAA batteries [40]. From Table 5.2 the estimated battery life is about 19 days. Methods to improve battery life will be discussed in Chapter 9.

Table 5.2: Current consumption of treadmill sensor and estimated battery life.

Mode	Action	Cur. (mA)	Time(s)	Ave. Cur.(mA)
Sleep	Idle	1.00	1.79	0.99
Sleep	Transmitting	17.20	0.01	0.13
Sleep Total				1.13
Active	Idle	31.80	0.49	30.92
Active	Transmitting	46.40	0.01	1.28
Active Total				32.20
Average mAh				2.42
Battery mAh	1100.00		Days of operations	18.93

5.3.4 Elliptical Sensor

Software

The software for the elliptical is similar to the treadmill software. It varies in two respects it does not have a method for the end user to calibrate the sensor and the ISR for the ADC is different.

A method for calibration for the end user is not provided because the sensor detects the zero crossing of the acceleration vector. The MMA7361LT has radiometric outputs, with it zero value always being half way between the maximum and minimum voltage. When it is sampled this will be the value 0x7F. In reality the zero value varies slightly, but it will be close enough that an accurate enough speed measurement can be determined without end user calibration. In the worst case scenarios the sensors could be calibrated when they are manufactured.

The algorithm run by the ADC ISR is Algorithm 5.8. Algorithm 5.8 is executed at 2KHz. Each time it runs, it samples the y-axis of the accelerometer. It then calculates the average of the last 128 samples. This is done to help smooth the signal further. The averaging is performed by taking the previous sum and subtracting the oldest sample and adding the new sample then shifting the value right by 7. The average is calculated in this way to accelerate the execution of the ADC ISR. The previous average is also kept. When

the previous average is below the zero value and the current one above it the time since this last crossing is recorded. The RTC will fire every half second causing the last recorded time to be transmitted.

Testing

Each of the modes of the Elliptical Sensor were tested by sending the appropriate packets to the wireless receiver which relayed them to the sensor. In each case the mode behaved as expect with the correct packets being transmitted at the appropriate intervals. The sensor was then attached to a Welso Momentum 630 elliptical by using velco straps. The elliptical was then used at varying speeds while the measured time between crossings was recorded using a terminal program that received data from the wireless receiver. It was determined that for the most part the sensor worked appropriately, but it occasionally sent errant readings of very short time periods. This was corrected by adding a hold off time. Thus, if the crossing time will be less than 100ms the detected crossing is ignored. This limits the elliptical sensors effective speed range to a maximum of 10 Hz or 600rpm. This is acceptable because it would be extremely difficult for a person to reach 600 rpm on an elliptical.

Current measurements where obtained for the Elliptical Sensor in the same manner as the Treadmill Sensor. The results of the measurements are given in Table 5.3. From Table 5.3 it can be seen that the Elliptical Sensor is estimated to get about 27 days of use out of a pair of AAA batteries compared to 19 for the Treadmill. This is because the MMA7361LT uses much less power than the QRD1114.

5.3.5 Bicycle Sensor

Software

The last sensor implemented was the Bicycle Sensor. It uses the SS411P to detect the passing of a magnet on the spokes of a the bicycle wheel. The software for the Bicycle Sensors is identical to the Treadmill Sensor except it samples and turns on the SS411P instead of

Algorithm 5.8 Elliptical ADC ISR.

Input:
Mode m,
Array of Previous Samples $adcSamples$,
Sum of Samples $sampleSum$,
Index of Oldest Sample $oldestIndex$,
Elapsed Time $time$,

Output:
Milliseconds Since Last Crossing $crossingTime$,
Sum of Samples $sampleSum$,
Index of Last Sample $oldestIndex$,
Elapsed Time $time$

Begin
Open Latch
Read ADC into $adcOut$
$time = time + 1$
If$\{m = ACTIVE\}$
Begin
$oldestIndex = mod(oldestIndex + 1, 128)$
$previousSampleSum = sampleSum$

/*Calculate new sum*/
$sampleSum = sampleSum - adcSamples[oldestIndex]$
$sampleSum = sampleSum + adcOut$
$adcSamples[oldestIndex] = adcOut$

If$\{(previousSampleSum \div 128) \leq 127$ **and**$(sampleSum \div 128) > 127$
and $time > 200\}$
Begin
$crossingTime = time \div 2$ /* ADC samples at 2KHz*/
$time = 0$
End
If$\{time = 0xFFFF\}$ /*16 seconds*/
Set Mode to SLEEP
ElseIf$\{time > 36000\}$ /* 9 seconds*/
$crossingTime = 0$ /*Indicates speed is zero */
End
End

Table 5.3: Current consumption of elliptical sensor and estimated battery life.

Mode	Action	Cur. (mA)	Time(s)	Ave. Cur. (mA)
Sleep	Idle	1.47	1.79	1.46
Sleep	Transmitting	16.20	0.01	0.13
Sleep Total				1.58
Active	Idle	4.28	0.49	4.16
Active	Transmitting	13.48	0.01	0.37
Active Total				4.53
Average mAh				1.71
Battery mAh	1100.00		Days of operations	26.84

the QRD1114 and the calibration algorithm is not used. The calibration algorithm is not used because the SS411P outputs digital signal. This means that its output does not need to be sampled by the ADC. Two other options could have been used, having the change in the sensor activate a pin change interrupt, or polling the pin as a digital input.

Having the pin change activate an interrupt was not possible because the system goes into sleep mode and when doing so latches the I/O ports. This means that change in the pin is not detected by the micro-controller to trigger the interrupt. The latch could remain open so it could detect the pin change but in doing so there is no way to ensure that P1.7 remains high. P1.7 is used to turn on the SS411P if it doesn't stay high the sensor will turn off.

The second method of polling the pin was considered but all the timers, except the RTC, on the nRF24LE1 sample much faster than is needed for this application. The ADC of the nRF24LE1 enables a sample rate to be set and then it samples at this rate. The available sample rates are 2KHz, 4KHz, 8KHz, and 16Khz. These are all power of two multiples of 1KHz, thus using them to sample the pin provides both the time value needed and the means to sample the sensor. Therefore, the ADC was used and the same algorithm as the treadmill was used with minor modifications.

Testing

The Bicycle Sensor was tested in the same manner as the Treadmill Sensor and Elliptical Sensor, by first checking that the communication and different modes worked. It was then attached to the bicycle and the accuracy of the measurements tested by recording them using a terminal program. The sensor performed adequately over a range of speeds from 2 meters per second to 13 meters per second. Current measurements were also obtained in the same manner as before and are given in Table 5.4. It is estimated that its battery would last about 21 days.

5.4 Summary

Three sensors and a wireless receiver were implemented for the Indoor Escape system. The sensors were tested and they were able to measure the speeds of their respective type of exercise equipment with an acceptable degree of reliability. Power estimates for each of the sensors were obtained and it is estimated with two AAA batteries the sensors could operate for between 19 and 28 days. This meets the power requirements that they be able to operated long enough to not to detract from the system. If rechargeable batteries were used, charging the sensors twice a month would be acceptable to most users. With the implementation of the wireless receiver the sensors are able to communicate wirelessly. In summary, the sensors as implemented were able to meet the design goals of low-cost,

Table 5.4: Current consumption and estimated battery life for the bicycle sensor.

Mode	Action	Current (mA)	Time(s)	Average Current
Sleep	Idle	1.00	1.79	0.99
Sleep	Transmitting	17.20	0.01	0.13
Sleep Total				1.13
Active	Idle	8.94	0.49	8.69
Active	Transmitting	23.50	0.01	0.65
Active Total				9.34
Average mAh				1.47
Battery mAh	1100.00		Days of operations	31.22

ease-of-use, reliability, and retrofitablity.

Chapter 6

Micro-Server Implementation

The micro-server is a small application that facilitates communication between the display subsystem and the wireless receiver. It works as a router between the wireless receiver and the display subsystem. For the Indoor Escape system to work, the micro-server application needs to be running on a computer that is connected to a wireless receiver. For a home user, this would most likely be the computer on which the display subsystem is run. In a gym environment it would likely be provided by the gym and either run the display subsystem or allow users to bring in their own laptops that would connect to the micro-server to obtain sensor information.

The micro-server supports two different connections. It connects to a wireless receiver via a serial port and uses TCP/IP sockets to connect to display subsystems. The micro-server was implemented in such a way as to support communication between multiple sensors and multiple display subsystems. The micro-server was written in C and uses POSIX threads and BSD sockets. It was compiled for the Linux operating system for the purpose of this project, but the selection of C, POSIX threads, and BSD sockets minimizes the effort required to port the application to other operating systems. The complete source code for the micro-server can be found in the Appendix.

This chapter describes the implementation of the micro-server and is organized as follows. The micro-server uses three threads. These threads in the order they are presented are: the initialization and TCP/IP listener thread, the TCP/IP connection handler thread, and the serial port handler thread. Combined these three threads comprise the implementation of the micro-server. The micro-server uses two structures to keep track of important information about the sensors in the system and the TCP/IP connections. These structures are the Serial Devices Structure and TCP Connection Structure. The structures will

be discussed first followed by the discussion of the three threads. The last section details how the micro-server was tested independent of the complete system.

6.1 Structures

6.1.1 Serial Devices Structure

The Serial Devices Structure keeps track of the information that is needed to route the information received from TCP/IP connections to the correct device and vice versa. It contains six properties: srcAddr, type, lastReceivedTime, displayFd, mutexPtr, and mutex. The first property, srcAddr, is the four-byte address of the sensor or other serial device. The type was intended to be used to identify what type of device it is, but the field is never used in the micro-server. The lastReceivedTime is used to keep track of the last time data was received from this device and is used to determine if this device can be over written. DisplayFd is the file descriptor written when sending data from this device to the associated display. The mutexPtr points to a mutex that is used to protect access to the displayFd. The last property mutex is a POSIX mutex that is used to protect access to the instance of the device structure.

6.1.2 TCP Connection Structure

The TCP connection structure contains information used to access the connections made over TCP/IP and to control access to those connections. A new structure is created for each TCP/IP connection that is made. It contains four parameters: srcAddr, fd, fdMutex, and active. Currently only display subsystems connect over TCP/IP thus for the remainder of this discussion the devices that connect will be referred to as displays. The property srcAddr, is the four-byte address of the display that is connected on this TCP/IP connection. Fd is the file descriptor used to read and write to this connection. fdMutex is a POSIX mutex that is used to control access to the file descriptor. The final field is a boolean indicating whether this structure contains information about an active TCP/IP connection.

6.2 Initialization and TCP/IP Listener Thread

The first thread is the initialization and TCP/IP listener thread. This is the entry point of the micro-server. It makes the necessary initializations and spawns a thread to listen to the serial port. It then listens for connections over TCP/IP. When a connection is requested over TCP/IP, the intialization thread creates the connection and then spawns another thread to listen to this connection; it then returns to listening for new connections. The micro-server as implemented allows up to 16 sensors and five displays to be connected to the system simultaneously. These parameters can be easily changed and were arbitrarily chosen. Algorithm 6.1 shows the routine executed by the initialization and TCP/IP listener thread.

6.3 TCP/IP Connection Handler

The TCP/IP connection handler is a routine that is runs in its own thread. It listens to and handles the commands and data sent from one display to the micro-server. Thus, a new thread is generated for each TCP/IP connection that is made to the micro-server. The connection handler listens for data on the socket and when it receives it determines where the packet needs to be sent and what it needs to do. All of the data received except one command is expected to follow the ALP. The exception is sent while establishing a connection with the ActionScript socket for security purposes.

6.3.1 ActionScript Socket Security

To prevent connections from being made via ActionScript Sockets to unauthorized servers, the ActionScript Socket class requests a cross domain policy file from the server. If the server does not provide this file or the computer making the request is not authorized and the ActionScript socket class does not make the connection. Thus, it was required that the micro-server respond with the cross domain policy file as specified in Adobe's cross domain policy file specification [41]. The ActionScript Socket class requests this file by sending the string <*policy-file-request* />, When this is received the micro-server responds with the following Extensible Markup Language (XML) policy file.

Algorithm 6.1 Initialization and TCP/IP listener routine.

Input:

Array of 16 Serial Devices Structures $Sensors$,
Array of 5 TCP Connection Structures $Connections$,
Serial Thread Mutex $serialMutex$,
TCP/IP port number$port$

Output:

Begin

Initialize serial port to 38400 baud in canonical mode
Get serial port file descriptor as $seriaFd$
Initialize $serialMutex$
For each{$sensor$ *in* $Sensors$ **as** $sensor$}
Begin
$sensor_{displayFd} = -1$ /*-1 designates it as invalid*/
$sensor_{srcAddr} = 0$
Initialize $sensor_{mutex}$
$sensor_{last_receivedtime} = 0$
$sensor_{mutexPtr} = NULL$
End
/*First sensor structure is reserved for Join Address */
Set srcAddr of first sensor in $Sensors$ to Join Address
For each{ Structure in $Connections$ **as** Con}
Begin
$Con_{Active} = false$
$Con_{fd} = -1$ /*-1 designates it as invalid*/
Initialize $Con_{fdMutex}$
$Con_{scrAddr} = 0$
End
Create serial listener thread with $serialFd$ as input
Initialze TCP/IP socket
$socketFd$= Bind TCP/IP Port Number $port$
Forever
Begin
Listen to $socketFd$ /*Blocks until connection made */
$newFd$ =Get File Descriptor of New Connection
If{$newFd < 0$}
Begin
Display Error
End

algorithm continued below

Algorithm 6.1 cont. Initialization and TCP/IP listener routine.

```
        Else
        Begin
            AvailableCon =First inactive connection in Connections
            AvailableCon_fd = newFd
            AvailableCon_Active = true
            Create new TCP/IP handler thread with AvailableCon as input
        End
    End
End
```

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
<allow-access-from domain="*" to-ports="51321" />
</cross-domain-policy>
```

This policy allows connections to port 51321 from any domain. In actual deployment of the system, this would be changed to only accept connections from the domain serving the Indoor Escape system.

6.3.2 TCP/IP Connection Handler Routine

The TCP/IP connection handler routine handles one connection. It uses a blocking read to receive data from the socket and new data is read when a new line is received. When data is received, it is checked to see if it is a policy file request. If it is, the policy file is sent. Otherwise it is decoded to see if it is an associate or unassociated packet. If it is either of these packets, it associates or unassociates the requested sensor. If the packet is any other packet, it checks if the sensor is in the system, and if the sensor is, the micro-server writes the packet to the serial port. If the sensor is not known in the system the packet is discarded. Algorithm 6.2 shows the routine performed by the each TCP/IP connection handler created.

Algorithm 6.2 TCP/IP connection handler routine.

Input:
Array of 16 Serial Devices Structures $Sensors$,
Mutex for serial port $serial_{mutex}$,
TCP Connection Structure Con
Output:
Begin
Forever
Begin
Wait for packet to be received on Con_{fd}
$packet$ = packet received from Con_{fd}
If{ Connection closed by client}
Begin
Remove all associations this connection has in $Sensors$
$Con_{Active} = false$
Exit Thread
End
If{$packet =< policy - file - request >$}
Begin
Send policy file
Remove all associations this connection has in $Sensors$
$Con_{Active} = false$
Exit Thread
End
Else If{$packet_{type}$ =Associate}
Begin
Search $Sensors$ for sensor that has $srcAddr = packet_{destAddr}$
If{sensor found}
Begin
Associate the sensor with this TCP/IP connection
End
Else
Begin
Return Error "Sensor not in system"
End
End
Else If{$packet_{type}$ =Unassociate}
Begin
Search $Sensors$ for sensor that has $srcAddr = packet_{destAddr}$
If{sensor found **and** sensor is associated to this connection}
Begin
Unassociate the sensor from this TCP/IP connection
End

algorithm continued below

Algorithm 6.2 cont. TCP/IP connection handler routine.

```
                Else
                Begin
                        Return Error "Sensor not in system"
                End
            End
            Else
            Begin
                Search Sensors for sensor that has srcAddr = packet_destAddr
                If{sensor found}
                Begin
                        Write packet to the serial port
                End
                Else
                Begin
                        Return Error "Destination Unknown"
                End
            End
        End
End
```

6.4 Serial Port Handler

The serial port handler routine shown in Algorithm 6.3 is responsible to listen to the serial port for packets from the wireless receiver. Its primary purposes are to relay packets to the appropriate display system and keep track of what devices have communicated with the micro-server. When it receives a packet, it checks to see if it has received a packet from the device before. If it has, it updates the time it last received data from that device, and looks to see if there is a display associated with the device. If there is an associated display, then the packet is forwarded on to the display. If this is the first time the micro-server has received data from this device, the micro-server finds an available Serial Device structure to save the devices information to. It saves the source address of the packet and the last time that it received a packet from this device. It then returns to listen for additional packets. Algorithm 6.3 shows how this is done.

Algorithm 6.3 Serial port handler routine.

Input:
Array of 16 Serial Devices Structures $Sensors$,
Array of 5 TCP Connection Structures $Connections$,
Serial Port file descriptor fd
Output:
Begin
Forever
Begin
Wait for packet to be received on serial port
$packet$ =received packet from serial port
n = length of received packet
If$\{n = packet_{length}\}$
Begin
$deviceFound = false$
$insertSensorStruct = NULL$
/*first sensor is reserved for Join*/
For Each{ Sensor in $Sensors$ **as** $sens$ except first sensor }
Begin
Lock $sens_{mutex}$
If$\{packet_{srcAddr} = sens_{srcAddr}\}$
Begin
$deviceFound = true$
$sen_{lastTimeReceived} = NOW$
If$\{sens_{displayFd} \geq 0$ **and** $sens_{mutexPtr} \neq NULL\}$
Begin
Lock $sens_{mutexPtr}$
/* Check to ensure displayFD is still valid*/
If$\{sensor_{displayFd} \geq 0\}$
Begin
Write $packet$ to $sens_{displayFd}$
End
Unlock $sens_{mutexPtr}$
End
End
If$\{insertSensorStruct = NULL\}$
Begin
Determine if this $sens$ is inactive
If{$sens$ is inactive}
Begin
$insertSensorStruct = sens$
End
End

algorithm continued below

Algorithm 6.3 cont. Serial port handler routine.

```
                End
                /* Device wasn't found but available structure was*/
                If{deviceFound = false and  insertSensorStruct ≠ NULL}
                Begin
                    Lock insertSensorStruct_mutex
                    insertSensorStruct_srcAddr = packet_srcAddr
                    insertSensorStruct_lastReceivedTime = NOW
                    Unlock insertSensorStruct_mutex
                End
            End
        End
End
```

6.5 Testing

After implementing the micro-server, it was tested by designing a simple ActionScript socket program that could send each of the communication packet types. The program made its connection to the micro-server over TCP/IP and provided an output to display the data that was received. At the same time the serial port was connected to another computer using a null modem cable. The other computer was running a terminal program. The terminal program was used to simulate the wireless receiver sending and receiving packets.

With this setup each packet type was sent from the terminal and the response of the ActionScript program noted. When the sensor was unassociated the test program receive no data from the simulated sensor, but data could be received from the test program to the terminal. This was expected but this feature may want to be removed in future work. When the sensor was associated all packets from the terminal were received at the test program. The sensor could then be unassociated and the test program would not receive the packets from the terminal any more. In summary, the micro-server was able to provide the capabilities to allow communication between an application running in a web browser and the sensors within the security frame work. In addition, it was able to support multiple simultaneous TCP/IP connections and multiple sensors. The final subsystem is the display

subsystem. It will be discussed in Chapter 7.

Chapter 7

Display Subsystem

7.1 Overview

The display subsystem is really the heart of the Indoor Escape system. After installing the sensors the only portion of the system that the user should be aware of is the display subsystem. The display subsystem provides the user interface and immersive environment. As discussed in Chapter 3, it is a web application that runs in a web browser. It is also by far the largest and most complicated subsystem of the Indoor Escape system.

Implementation of the display subsystem required the used of several programming languages and technologies. Web pages can be broken into three components: content, layout, and scripts. The content of the web pages of the Indoor Escape system is generated using PHP and HTML. PHP also uses SQL to access a MySQL database to add content to the web pages. How the display of the content is presented is controlled by CSS and the scripts used by the pages are written in JavaScript and ActionScript. The display subsystem will be presented as it is experienced by the user. Each page will be presented in the order in which they would be used by a typical user. As the pages are presented the features of each page will be presented along with the actions that they cause to happen behind the scenes.

7.2 Main Page

The first page that the user will come to is shown in fig. 7.1. This is the main page it provides several main features, the ability to create a route, the ability to resume/restart a previous route, identification of the current active sensor, and a link to configure the sensors. The bottom right corner shows two additional features that are displayed only for debugging purposes. The white box is the flash application that uses the ActionScript

Socket to provide communication with the micro-server. The little black box with three buttons visible is a console that was used for debugging and will be discussed later.

7.3 Configure Sensors

The first time a user uses the Indoor Escape system they would need to configure a sensor. This is done by clicking on the configure link as shown in fig. 7.2. This will use AJAX to load the sensor configure page shown in fig. 7.3. AJAX is used to load the page so that information on the main page is not lost. For example, this would allow a user to create a new route then realize they need to change the sensor they are using. They could then change the sensor without losing the newly created route.

The configure sensor page lists all the sensors in the system. From the configure page sensors can be added, removed, and modified. In addition, the active sensor can easily be changed to any other sensor in the system. Five properties of each sensor are saved in the database. These include the four-byte sensor ID used to communicate with the sensor, a user defined name for the sensor, the distance traveled per revolution, and the user the sensor is associated with. The properties are displayed and modifiable on the configure sensor page.

7.3.1 Adding Sensor

The configure sensor page has three icons at the top of the page just under the Add Icon heading. These icons represent the types of devices that are supported by the Indoor Escape system. Clicking on one of them initiates the addition of a new sensor. If the bicycle icon is clicked, the new sensor is assumed to be connected to a bicycle, likewise the T represents a treadmill and the E represents an elliptical. With the exception of the type of sensor added identical actions are taken. When it is requested to add a sensor the display system sends a join command to the micro-server which relays it to the wireless receiver. When the wireless receiver gets this command it changes to join mode. The display also sends an associate command to the micro-server with the join address as the requested device. This allows the display system to receive all the data sent by the wireless receiver on the

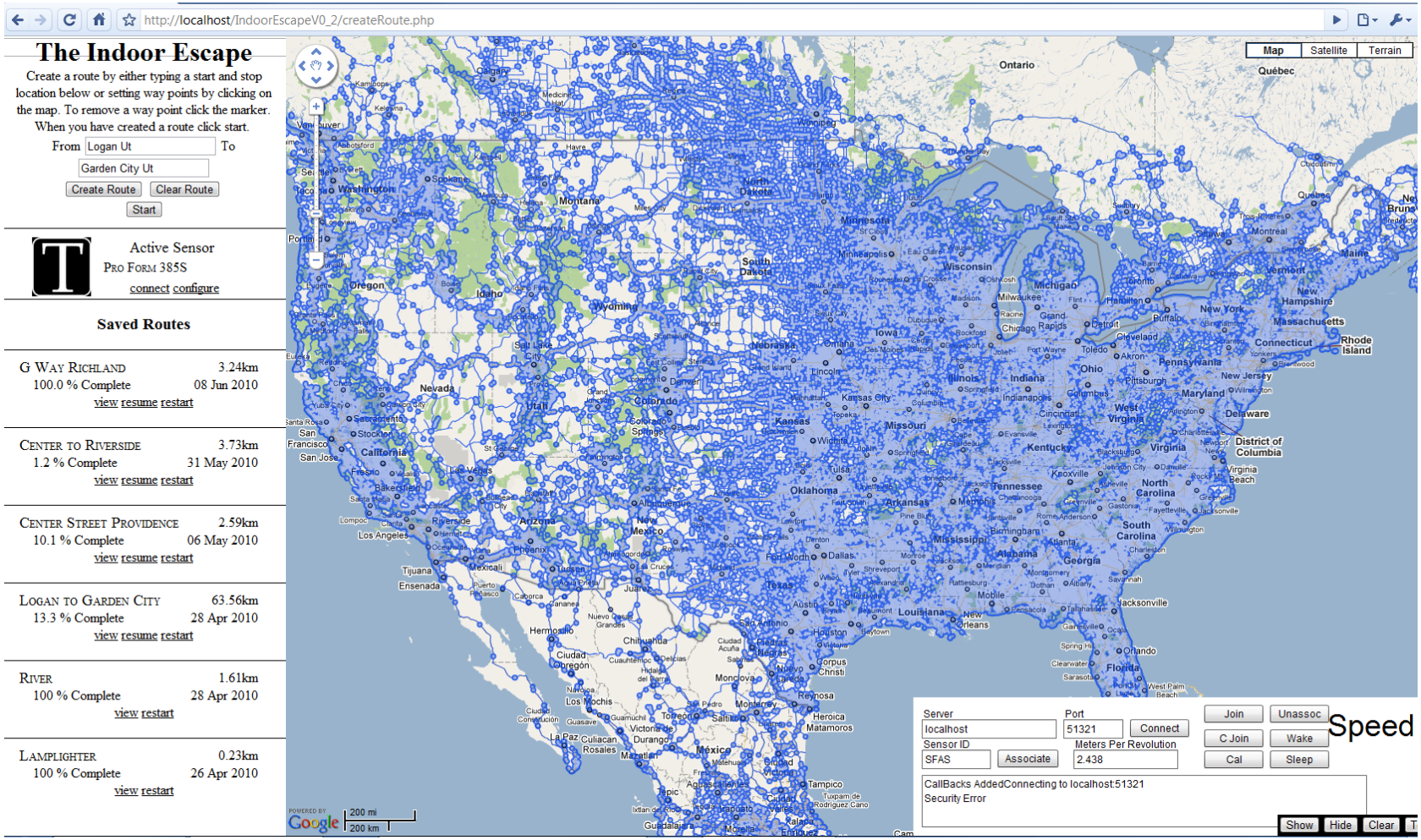

Fig. 7.1: Screen shot of the main page of the display subsystem.

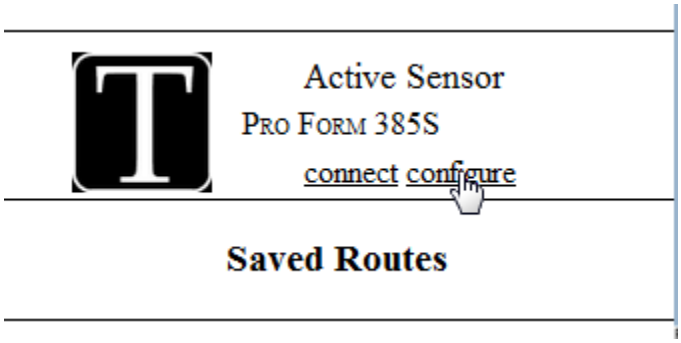

Fig. 7.2: Link to display page to configure sensors.

join address. The display system then displays a message instructing the user to press the join button on the sensor. This message also contains a button to allow the cancelation of adding a sensor.

When the join button on the sensor is pressed, the wireless receiver gives the sensor the frequencies and address needed to communicate with the wireless receiver. The sensor then sends back a join success and the wireless receiver relays the join success up to the display system. The join success contains the sensor ID needed to communicate with the sensor. At this point the message is cleared, the display unassociates from the join address, and the new sensor is added to the display with a default distance per revolution based on the type of device that is added. The distance per revolution is displayed as tire diameter, track length, or wheel radius depending on the device that is added. The sensor info is also added to the database using an AJAX request.

7.3.2 Change Sensor Info

To change sensor info the user clicks on the wrench icon in the sensor box as shown in fig. 7.3. This will cause the sensor to enter a modifiable state and displays the box to change the distance per revolution as shown in fig. 7.3 by the elliptical sensor. The user can then modify the information and click the save or cancel links. The save link sends an AJAX request to the web server which contains the sensor info. When the web server receives this request it updates the sensor info in the database. The cancel returns all settings to their previous values and returns the display to an unmodifiable state.

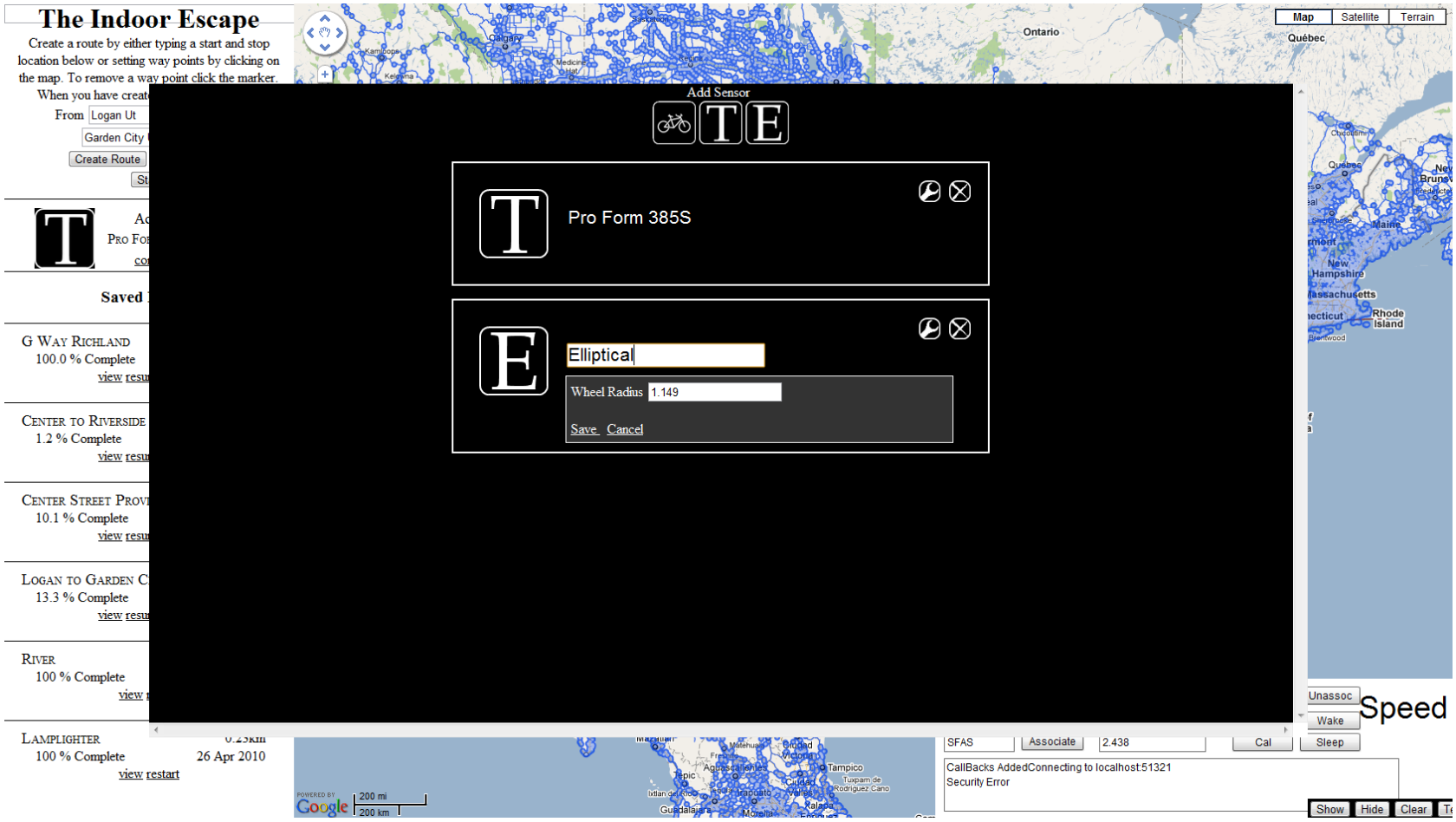

Fig. 7.3: Display system showing configure sensors page.

7.3.3 Removing Sensor

Sensors are removed by clicking the X icon in the sensor box. The system will then display a message to confirm the deletion of the sensor. If the user confirms the deletion an AJAX request is sent to the web server telling it to delete the sensor from the database and the sensor is removed from the display.

7.3.4 Selecting Different Sensor

The Indoor Escape system only uses one sensor at a time. The sensor currently being used is referred to as the active sensor. The active sensor is displayed on the main page, but can be changed by going to the sensor configure page and selecting a different sensor. The sensor is selected by clicking on the big icon in the sensor box. In fig. 7.3, this would be either the Bicycle, or the large T or E icons. When the icon is clicked on, the sensor selected is set as the active sensor and the sensor configure window is closed.

7.4 Creating a Route

One of the main features the home page provides is the means to create a route. For the purposes of this chapter a route is an ordered list of waypoints. Each waypoint is a latitude longitude coordinate pair that marks a change in direction. By drawing straight lines between the waypoints, the route can be drawn on a map. The two means of creating a route are clicking on a map to set way points and entering start and stop location.

7.4.1 Creating Route by Setting Markers

The main portion of the main page shown in fig. 7.1 is a map. This map can be manipulated by use of a mouse. Clicking and dragging will allow moving around the map and the zoom of the map is controlled by the scroll wheel. When the route is clicked a marker is set at that location. When the map is clicked again, a second marker is set and the directions between the first and second markers are obtained via Google Maps API. The directions returned by the maps API includes a list of vector coordinates, called by Google

a polyline, which is used to draw the route from the first point to the second point on the map. This polyline is used as the route.

Additional points can be set by adding more markers to the map by clicking. With each additional marker the directions from the first marker placed to the second are obtained followed by the directions from the second to the third continuing until the directions from the second to last marker to the last marker are obtained. Google limits the number of markers that can be used to obtain directions to 25.

Markers can be removed by clicking on the marker. When a marker is removed the directions are obtained again with that marker removed from the list of markers. Using this method of creating a route a high level of control is given to the user.

7.4.2 Creating Route by Entering Start and Stop Locations

The second method of creating a route is entering start and stop locations. The locations are entered as common text, such as Logan UT. The locations are then composed into a string with format Start Location to End Location. This string is passed to Google's Maps API GDirection class [25]. This class accepts the string and geocodes the locations and then obtains directions between the two points. If the geocoding is successful the route is displayed on the map and a polyline is returned. This polyline is used as the route. If the geocoding is not successful a message is displayed informing the user.

This method of creating a route allows a route to be created very quickly, but does not give the user very fine control over the path taken from the start location to the end point. After a route is created it can be traversed. Before discussing traversing a route the use of saved routes will be discussed.

7.5 Saved Routes

When a new route is created and the user clicks the start button traversal of the route begins. While the route is loading, the user is given the opportunity to save the route. When a route is saved, the user gives it a name. This name, along with all the waypoints in the polyline defining the route, are stored in the database. In addition, the latitude and

longitude of the last Street View image shown, the total distance of the route and distance traveled are saved in the database in the route table.

The main page displays the saved routes and provides up to three options for each route. The first option is viewing the route. When this option is selected, the route is drawn on the map and a marker placed at the current location on the route. The second option is resuming. Resuming a route is only available if the end of the route was not reached the last time it was used. When resume is selected, traversal of the route begins from the last location on the route. This allows the user to create a large route and over the course of several sessions traverse the route. The final option is to restart from the beginning. When this is selected, traversal of the route is started from the beginning.

7.6 Traversing a Route

Traversing a route is accomplished by displaying the Street View images from along the route in succession and switching the images as the real-world distance between them is passed. Google's Map API provides several classes that are very important to implementation of the traversal algorithm. These classes will be discussed first, followed by the initialization of a route for traversal. The initialization of a route varies depending on if the route is starting from the beginning or in the middle. Both of these will be discussed in their own subsections. Following those subsections, the discussion of loading the buffer will be presented. This section will then conclude with a subsection describing traveling along a route.

7.6.1 Google Maps API Classes

The first class that is important to understand is the GStreetviewPano. This class contains an instance of the Streetview flash applications. When it is initiated it is given an HTML div object in which it displays the selected panorama. The most important method of this class sets the location and point of view of the panorama displayed in the Street View application.

The second important class is the GStreetviewClient. This class provides methods to

obtain data about Street View images. It operates independently of all GStreetviewPano classes. This allows information about Street View images to be obtained without displaying the images. The data returned is the GStreetviewData class. The GStreetviewData class contains information about the location of the images such as it latitude and longitude, the panorama's ID, links to neighboring Street View images, and the neighboring images's IDs. An important characteristic of the ID is that they are only unique and stable during a session. Thus, saving them is not useful because on later uses of the Indoor Escape system the IDs may have changed and would no longer be valid. How these classes are used to traverse a route will be discussed in the following subsections. First the initialization of a route will be discussed.

7.6.2 Traversal Algorithm State Variables

The traversal algorithm uses several variables to maintain its state and display information. The traversal algorithm is implemented in a JavaScript class and the complete code for the algorithm is included in the Appendix. The class member variables needed to understand the algorithm are described here. The algorithms in the subsections below contain a new section called state. The class member variables that are used in the particular algorithm are listed in this section and they retain their values across algorithms and execution of the algorithms.

The class member variables for the traversal algorithm can roughly be grouped into three categories: display element variables, display control variables, and buffer loading variables. The display element variables are used to access elements that are displayed on the screen. Manipulation of these variables usually changes what is displayed on the computer monitor. The display control variables are used to keep track of what is being displayed, what should be displayed next and when it should be displayed. The final sets of variables are used to load the frame buffer. They include information about what frame should be loaded and where the information that needs to be loaded into that frame is located. The variables are listed below along with a description of the variables properties and a description of what the variable is used for.

Display Element Variables

- route: This is a list of longitude latitude coordinate pairs that mark each location where the desired path changes direction. The list is ordered with the first element being the starting location and a straight line is followed to the second location and so forth.
- speedometer: This is an HTML div object that is used to display the current speed. Other information, such as low battery, can also be displayed in this object.
- frames: This is an array of frame objects. Each frame object contains an instance of the GStreetviewPano class and several properties. The properties include the panoId of the image in the frame, the distance between this Street View image, and the Street View image in the previous frame, The length property of frames is the number of objects in the array. The frames compose a buffer that allows Street View images to be loaded and rendered in the back ground.
- map: A map that is displayed showing the route and a marker of the current location.
- marker: This is the marker that is displayed on the map. Changing its location moves the marker on the map.

Display Control Variables

- curSpeed: This variable contains the current speed of the exercise equipment. It is updated every time a time packet is received from the micro-server. For purposes of the algorithms below, it is assumed to always contain up to date information.
- distTraveled: This variable contains the total distance traveled on the route.
- distToNextFrame: This variable contains the total distance from the beginning of the route at which the next frame should be displayed.
- activeFrame: This is an index into the frames variable that is used to reference the frame that is currently being displayed on the screen.

- routeId: Contains the Id that is used to reference the route in the database. If the route is not saved it is set to -1.

Buffer Loading Variables

- curLocation: Marks the current location on the earth where Street View data should be loaded from.
- corners: An array of longitude and latitude coordinate locations where the route changes direction and a Street View image is available. These locations are referred to as corners.
- curCorner: An index into corners of the last corner that will be reached in corners.
- frameToLoad: An index into frames that marks the frame that the next Street View image should be loaded into.
- lastFrameLoaded: An index into frames that marks the last frame that was loaded.
- routeIndex: An index into route indicating the last coordinate in the route that was passed by the loading algorithm.
- numInBuffer: Contains the number of frames in the buffer that contain valid content.
- streetviewClient: An instance of the GStreetviewClient that is used to check the availability of Street View data and to obtain that data.

7.6.3 Initialization for Traversing a Route

The initialization of the traversal algorithm sets all of the state variables to their desired values and then starts traversing the route. The initial values for the state variables vary depending on how the route was obtained. Algorithm 7.1 shows the procedure used to initialize the traversal algorithm. The initialization algorithm has one required input, a path. It also has several optional inputs these are input if the route to follow was previously saved. If the route is being restarted, only the route and routeId are given. If the route is

being resumed, then additionally, the distTraveled and curLocation from the previous run are input. After initializing the variables, it calls the algorithm to load the buffer. After the buffer is loaded, an interval timer is set to 500ms. The timer causes the follow route algorithm to be executed every 500ms.

7.6.4 Loading the Buffer

Loading the buffer is accomplished by using two algorithms, Algorithm 7.2 and Algorithm 7.3 Algorithm 7.2 queries Google's server for Street View data at the coordinate in route indexed by routeIndex. When the query returns it, data is checked to see if a Street View image is available. If an image is available, it is checked to see if its panoId matches the panoId of the previous image. If it is a match, the data is discarded. If the image is new, it is appended to the corners array. routeIndex is then advanced. Algorithm 7.2 is necessary because Algorithm 7.3 uses the links returned with the image data to identify the next image on the route to load. The correct link is identified by determining the bearing link that follows the bearing from one point in route to the next point. Often locations in the route are close enough together that they return the same Street View images. This is particularly common on curves. Calculating the bearing between two points when they are the same gives an undefined result. Thus, to avoid this, Algorithm 7.2 is used to obtain the images data at each point in the route and ensures that each set of sequential points is different. It also provides some resilience to missing links and area where Street View data is not available by skipping over them.

Algorithm 7.3 is called after at least three corners have been obtained. The first thing Algorithm 7.3 does is find the next image's panoId. To do so the algorithm determines the bearing from the current corner to the nextCorner and looks for a link with bearing closest to the calculated bearing within the curLocation's links. If the link is within 25°, it is selected as the link to follow. The panoId of the next link is then compared to the panoId of the image at the next corner. If they are the same, the Algorithm 7.2 is called and nextCorner is advanced if there is another corner. If there is not another corner, the end of the route has been reached. The location of the next image is then obtained by

Algorithm 7.1 Display subsystem initialize route.

Input:
List of waypoints *path*,
Optional Input:
Route Id number *savedRouteId*,
Distance Traveled *totalDist*,
Last Location on Route *lastLocation*
State:
Buffer Frames *frames*,
Inset Map *map*,
Current Location on Route *curLocation*,
List of waypoints *route*,
Frame To Load *frameToLoad*,
Last Frame Loaded *lastFrameLoaded*,
Number of Panoramas In Buffer *numInBuffer*,
Route Id number *routeId*,
Distance Traveled *distTraveled*,
Last Route Index *routeIndex*,
Distance Between Street View Images *distanceBetweenPanos*,
Current Corner Index *nextCorner*,
Longitude Latitude Coordinates of The Corners of the Route *corners*
GStreetviewClient Class *streetviewClient*,
Last Corner that was Passed *curCorner*
Begin
Display message Please Wait While Buffer Loads
If{*routeId* in not input} /*This is a new route*/
Begin
Parse *path* removing duplicate adjacent entries
$route = path$
$routeId = -1$
$distTraveled = 0$
Ask User if they would like to save *route*
$routeIndex = 0$
If{Save *route*}
Begin
Send route to web server using a AJAX request
/* routeId will be given its value when request completes*/
End
End
Else If{*routeId* input **and** *routeIndex* input } /*Resume route*/
Begin
Search segments of *route* for segment that *curLocation* lies near.
algorithm continued below

Algorithm 7.1 cont. Display subsystem initialize route.

```
        If{curLocation within 3 meters of more than one seqment}
        Begin
            routeIndex =index of segement with smallest error in dist. traveled
        End Else
        Begin
            routeIndex =index of the first point of the closest line in route
        End
        route[routeIndex] = lastLocation
        distTraveled = totalDist
        routeId = savedRouteId
    End
    Else /*Restart route */
    Begin
        distTraveled = 0
        routeIndex = 0
        routeId = savedRouteId
    End
    streetviewClient = New GStreetviewClient
    frames = CreateBuffer(10)          /* creates a buffer of 10 frames */
    Create Inset Map as map
    Display route on map
    curLocation = route[routeIndex]
    curCorner = 0
    frameToLoad = 0
    numInBuffer = 0
    lastFrameLoaded = −1               /*marks that no frame has been loaded*/
    distanceBetweenPanos = 0
    While{corners_length < 3 and routeIndex < route_length}
    Begin
        Call Get Corner
    End
    Call Load Buffer
    Remove please wait message
    Set Interval Timer to 500ms
End
```

using the GstreetviewClient to obtain the longitude and latitude of the image referred to in the link. When the location is returned, the GstreetviewPano class connected to the next frame is instructed to load the image at that location and set its point of view in the direction of the determined bearing. The variable lastFrameLoaded is then advanced. It is

Algorithm 7.2 Display subsystem get corner.

Input:
State:
List of waypoints $route$,
Array of Corner Pano Data Objects $corners$,
Last Route Index $routeIndex$,
Output:
Begin
$status = unsuccessful$
While$\{status = unsuccessful$ **and** $routeIndex < route_{length} >\}$
Begin
Request nearest image data for location $route[routeIndex]$
Wait For Request to Complete returning $data$
If{Query Successful **and** $data_{panoId} \neq corners_{last}\}$
Begin
Push $data$ into $corners$
$status$ =successful
End
Increment $routeIndex$
End
End

necessary to first obtain the longitude and latitude of the image by using the panoId because the GstreetviewPano class only has methods to load images by location. The process then repeats until the buffer is full at which point the algorithm exits.

7.7 Following a Route

Following a route is done by determining the distance between the active frame and the next frame in the buffer. The current speed is measured and the distance traveled in half a second accumulated. When the accumulated distance is greater than the distance to the next frame, the active frame is moved to the back of the buffer revealing the next frame. The revealed frame is set as the active frame. If the route has been saved, then the location of the active frame and the total distance traveled are saved to the database. This process is continued until the active frame that is revealed is marked as the last frame. At this point, a message is displayed saying the end of the route has been reached.

Algorithm 7.3 Display subsystem load buffer.

Input:
State:
Current Location on $curLocation$,
Last Frame Loaded $lastFrameLoaded$,
GStreetviewClient Class $streetviewClient$,
Buffer Frames $frames$,
Number of Frames $numFrames$,
Number of Panoramas In Buffer $numInBuffer$,
Last Corner Crossed $curCorner$
Corners Retrieved by Get Corners $corners$,
Output:
Begin
While$\{numInBuffer < frames_{length}$ **and** $curCorner < corners_{length} + 1\}$
Begin
Request image nearest to $curLocation$ using $streetviewClient$
Wait For Request to Complete
pov = Bearing from $corners[curCorner]$ to $corners[curCorner + 1]$
Find closest link in $frames[lastFrameLoaded]$
with nearest bearing to pov as $link$
If$\{abs(pov - link_{bearing}) > 25\}$ /* Skip to next Corner*/
Begin
$nextPanoId = corners[curCorner + 1]_{panoId}$
End
Else
Begin
$nextPanoId = link_{panoId}$
End
If$\{nextPanoId = corners[curCorner + 1]_{panoId}\}$ /*At next Corner*/
Begin
Mark next frame as the last
End
Else
Begin
Increment $curCorner$
Call Get Corner
End
Advance $lastFrameLoaded$
Get location of street view image with panoId $nextPanoId$
$frames[lastFrameLoaded]_{image}$ = image at returned location
$frames[lastFrameLoaded]_{pov} = pov$
$numInBuffer = numInBuffer + 1$
End
End

7.8 Attempted Improvements to Traversing a Route

The immersive environment create by the traversal algorithm has a few areas that could be improved and some attempts were made to try to improve them. One of the short comings is that Street View images are, on average, 10 meters apart. For someone running at a moderate pace of 2.5 meters per second, a new image will be displayed about every 4 seconds. During those four seconds it can feel as if the user is not moving. To improve the sense of motion zooming of the images was attempted. Each Street View image has a range of zoom levels provided by Google. Experimentation with these zoom levels showed that zooming in a level had about the same effect as moving to the next image. This could potentially be used to reduce the bandwidth required by the system, but does little to improve the environment. It actually degrades the environment because the zoomed images contain less peripheral data and can be quite blurry.

A second method was attempted where the size of the image was expanded using the DOM of the web page so it extended off the screen. This causes the browser to stretch the image and makes the viewable portion larger. This method showed some promise, but had draw backs. Matching a zoom level to a distance traveled proved to be very difficult. Often the zoom would go too far and when the next image was displayed it would step back. This caused the author to feel like he was getting whiplashed. This seriously degraded the environment. If it were possible to make the transition between the images smooth, this method would be promising, but this may require image processing to correctly match the distance between the images and the zoom level. Google does not provide this information and accessing it through other means violates the terms of use. The final reason why this approach was abandoned was because it pushed Google's Logo and copyright information off the screen, thus violating Google's terms of use.

7.9 Traversing a Route Summary

The algorithm used to traverse a route allows the user to experience an immersive environment while exercising. It uses a buffer of Street View images to allow the images to load and render in the background. This improves the feel of the immersive environment.

The algorithm also handles traversing sections of the route that do not have Street View data by displaying a map and the marker showing the current location. Thus, the algorithm continues to provide an enhanced environment for exercising even when Street View data is not available. The last component needed to complete the display subsystem is means to communicate with the sensors.

7.10 ActionScript Socket

A key component of the display subsystem is a flash application that provides a socket connection to the micro-server. This was implemented to allow two way asynchronous communications between the display subsystem and the sensors. The flash program exposes several functions that send commands to the sensors. When it receives packets it decodes the packets and calls JavaScript functions. A JavaScript wrapper was written that allows JavaScript scripts to interact with the flash application seamlessly.

The wrapper uses three functions for each packet type received: a send function, a receive function, and a register callback function. The send function is used to send the packet type with a specified address and, if applicable, data. The receive function is called by the flash application when it receives the particular packet type. The receive function looks up the callback function that should be called when this packet type is received from the particular device. This requires the third function that registers the callback function for each device. A look-up table of callback functions is kept for each packet type and is indexed by the address the packet arrives from. For example, if it was desired to have function updateSpeed called every time a time packet was received from address 0x53-0x43-0x34-0x98. The updateSpeed would be registered as a callback for device 0x53-0x43-0x34-0x98, by calling the register callback function for the time packet. The register callback function would add updateSpeed to the table of callbacks with the index 0x53-0x43-0x34-0x98. When the flash application received a time packet from device 0x53-0x43-0x34-0x98, it would call the receive function for the time packet, passing the data and address the packet was received from. The receive function would then look up in the time packet table for a callback function using the address 0x53-0x43-0x34-0x98. It would find the updateSpeed

function and call it, passing it the data. All other packets are handled similarly. The wrapper also includes a few other functions that group together common combinations. For example, one of these functions sends an associate command to an address and registers callbacks for the awake and time packet times. The complete code for the flash application and wrapper are given in the Appendix.

By using the wrapper, how each packet is handled can be change based on the state of the program. For example, when a route is being created, the callback functions for all received packets for the active sensor are set to functions that update the status of the sensor. When the system is traversing a route, the callback function for the time packet updates the current speed used in traversing the route.

7.11 Debugging

One of the greatest challenges of implementing the display system was testing and debugging. The display subsystem is the largest system and most diverse systems that the author has ever created. It uses six different languages. In JavaScript alone over 2200 lines of code are implemented. In addition, the system involves scripts running on both client side and server side with the need to keep data current on both ends.

The largest portion of the display subsystem code is written in JavaScript. In order for JavaScript code to execute, it must be triggered by an event. The display system was implemented in small sections so that debugging could be accomplished. This required the ability to run small portions of code and to vary the inputs. The simplest event to generate is the clicking of a button. This required that a button be added to the webpage for every portion of code that needed testing. Adding buttons while trying to maintain and design a user interface for the display system quickly became tedious. The solution was to implement a JavaScript class that created a console in the webpage. The console class has the ability to add buttons the callbacks. This allowed buttons to be added to the webpage with a few lines of JavaScript, and when the button is clicked, the callback function executed. The also console provided a place to display messages. This greatly aided the debugging of the display system. The console class also has a hide button which moves the console so it

is nearly off the screen. This made it so it was accessible but did not interfere with the designing of the user interface. Figure 7.4 show this console in the expanded mode over the main page.

Google's Chrome web browser also has a built in debugger that was very valuable in debugging because it allowed execution of the scripts to be stepped though line at a time. Using the console and debugger provide the tools needed to debug the Display subsystem.

7.12 Testing and Analysis

Testing of the display subsystem was the most involved of all the systems. It focused on several areas. They included creating routes, saving routes, restarting routes, resuming routes, and traversing routes. In all testing, the speed of the exercise equipment was hard-coded to 10 meters per second. A qualitative analysis was performed to determine the maximum speed the system can handle.

7.12.1 Creating Routes

Entering Start and Stop Location

Two test were performed for the testing of entering start and stop locations, entering valid locations, and entering invalid locations. When valid location were entered, the route was created and displayed on the map. When invalid locations were entered, a message indicating it was unable to locate the start and stop points was displayed. Thus, it performed

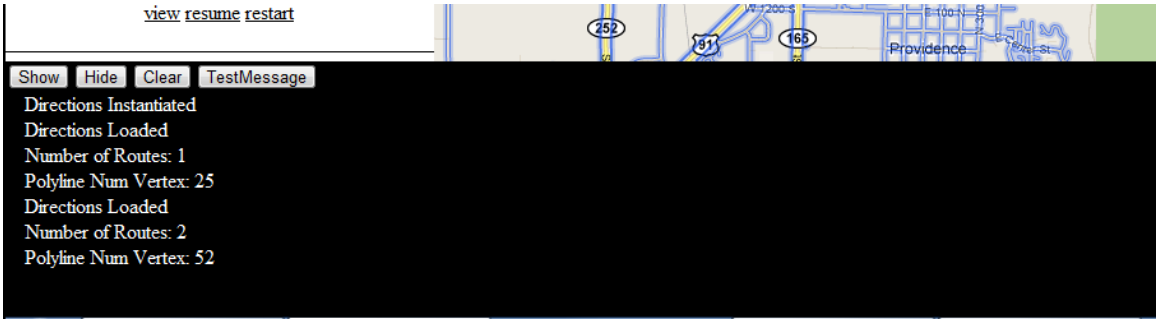

Fig. 7.4: The console created to facilitate debugging.

as expected.

Clicking to Set Waypoints

Clicking to set waypoints has two features that needed testing: setting waypoints and removing waypoints. To test it, several way points where added to the map by clicking on it. With each click the route was created connecting the points. The points were then removed in random order by clicking on the markers. With each click the marker was removed and the route updated until only one marker was left. Clicking on this marker removed it. A new route was then created by clicking on the map several times. A few of these points were then removed. The list of way points was then inspected using Chrome's debugger, and it was determined that the list of way points only had the desire markers remaining and in the correct order. Thus, the method of creating a route was successfully implemented.

7.12.2 Saving and Viewing Routes

After it was verified that the routes could be create using both methods, saving the routes was tested. This was done by creating a route, noting how the route looked on the map, and then starting to traverse the route. At this point the display system asks to save the route. A name was given to the route and save clicked. The display system was then reloaded. When the page was reloaded, the route was shown with the options to view, resume, and restart the route. View was selected and the route appeared on the map. The retrieved route was compared with the view of the route when it was creates. It was determined that it was the same.

A second test was performed, by creating a small route. The route was saved as before and then retrieved. The coordinates of the saved route and the retrieved route were compared to ensure they were the same.

7.12.3 Traversing and Restarting Routes

Saving and retrieving routes proved very useful in debugging the traversal of the route, because it allowed the same route to be repeated multiple times, and thus errors identified

and removed more easily. Traversal of the routes was checked using several tests. These tests where designed to ensure that the traversal algorithm could handle a wide diversity of routes. Four routes where created. The first route, Logan to Garden City, was created by typing "Logan UT" into the start location and "Garden City UT" into the endlocations. The created route is shown in fig. 7.5. It was then saved and started. This route is the longest route that was tested at 63km. About 8.2km into the route, the display started repeating the same images in a loop. This occurred on a bend in the route, and it is suspected, that when querying to get the corner data, that a loop was created involving three images. To address this issue, a new algorithm for traversing the route is proposed in Chapter 9.

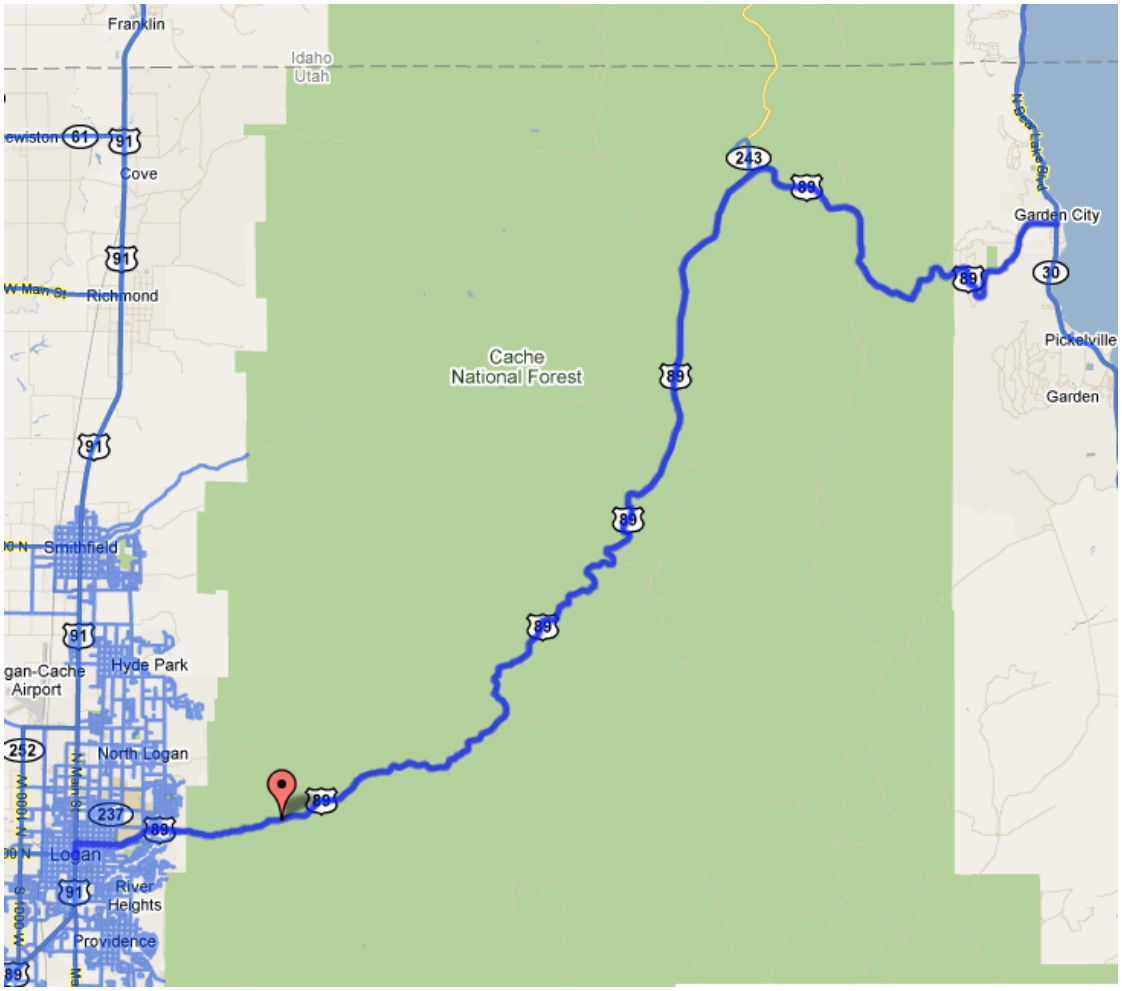

Fig. 7.5: Route going form Logan, Utah to Garden City, Utah that was used for testing.

The second route called River, shown in fig. 7.6, was created to test how the algorithm did when loops where included in the route. The route was also used to test resuming from a location where the route doubles back on itself. The route reached the correct ending point without difficulty. Resuming on the loop also worked. A video of the traversal of this route is included in the Appendix.

The third route, Center to Riverside, was designed to test the algorithms ability to take sharp corners. It is shown in fig. 7.7. It reached the end successfully navigating the sharp corners. A video of this test is included in the Appendix.

The final route was found mostly by chance, but it involves a location where the links between the Street View images are not present. The route called Lamplighter is shown in fig. 7.8. The intersection in the route does not have a link to follow when making the corner. The traversal algorithm was still able to successfully navigate the route and reach the end. The video of this test is available in the Appendix.

7.12.4 Performance Analysis

Performance of the system will be highly dependent on the computer used and the internet connection. In finalizing the report on July 6th, it was discovered that an update to the Chrome browser broke the Indoor Escape system on the Windows 7 operating system. Identical code running in Chrome on Mac OS X functions properly. This combined with the fact that the system performed properly in Chrome on Windows 7 previously makes it appear the problem is caused by the update. Time has not allowed the problem to be investigated further. Chrome also automatically updates and provides no means of rolling back to previous versions thus performance results are only presented for the Indoor Escape running in Chrome on Mac OS X.

For testing, a 21.5 inch iMac with 3.06Ghz Intel Core 2 Dou processor 4GB of Ram and 256MB graphics card was used. It was connected via Wi-Fi to Utah State University's Bluezone network. Different rates of travel were hard-coded in to find the maximum speed at which the route would work. The route Logan to Garden City was used. While this route had problems, the first portion of the route worked fine, but for this testing it was

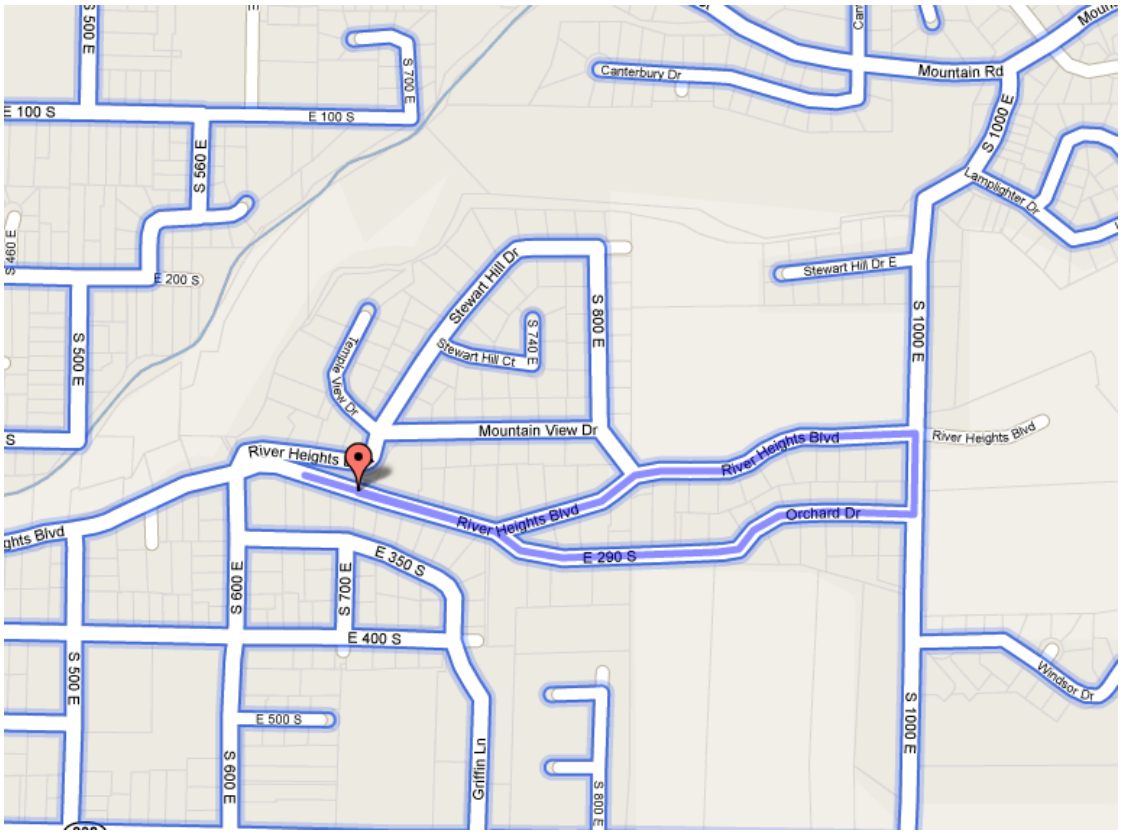

Fig. 7.6: Route, River, used to test performance of traversal algorithm on loops.

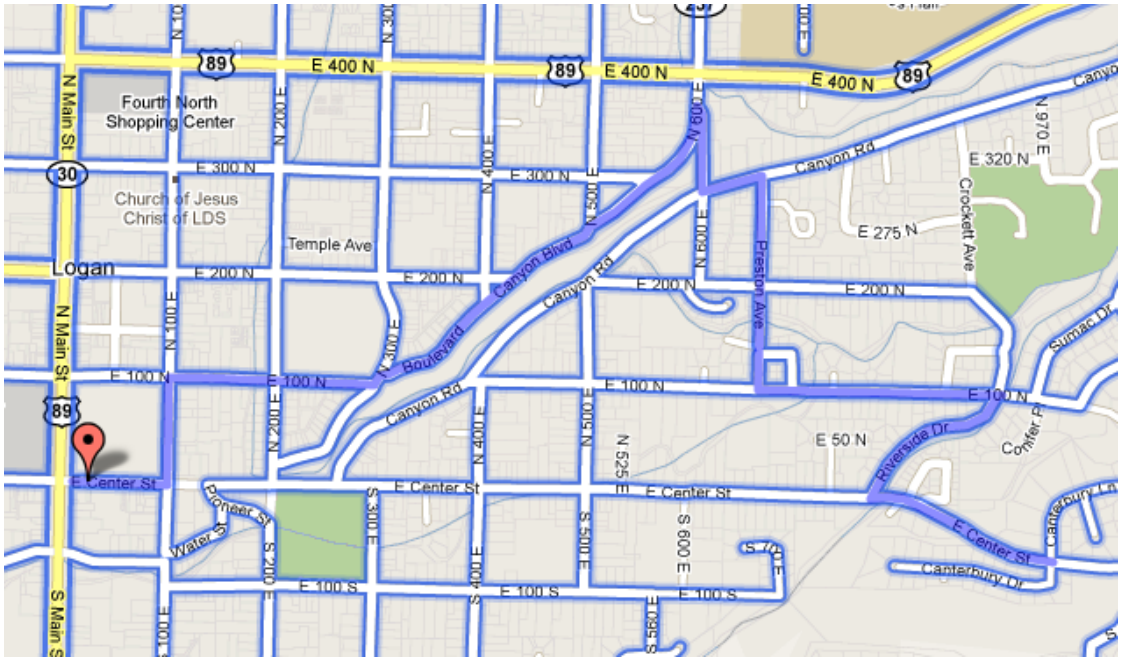

Fig. 7.7: Test route, Center to Riverside, used to test traversals algorithms ability to navigate sharp corners.

satisfactory. It was determined that the system could handle rates up to 15 meters per second. This would be a very difficult pace for the average person to maintain for extended periods on a bicycle. The speed of the internet connection would be the biggest limiting factor to the performance of a computer. This is because downloading the images is a very data intensive process.

7.13 Summary

The display system is the largest and most complicated subsystem of the Indoor Escape system. It allows the creation of routes, using two different methods. It can also save the routes so they can be reused later. These saved routes can either be restarted or resumed from the last location. Testing of the system showed that the display subsystem could navigate many routes, but that it may run into problems. An improved traversal algorithm is proposed in Chapter 9 to address the issues discovered. While the system is not capable of successfully navigating all routes it was able to complete several routes and does provide an immersive exercise environment. Chapter 8 will discuss testing of the complete system

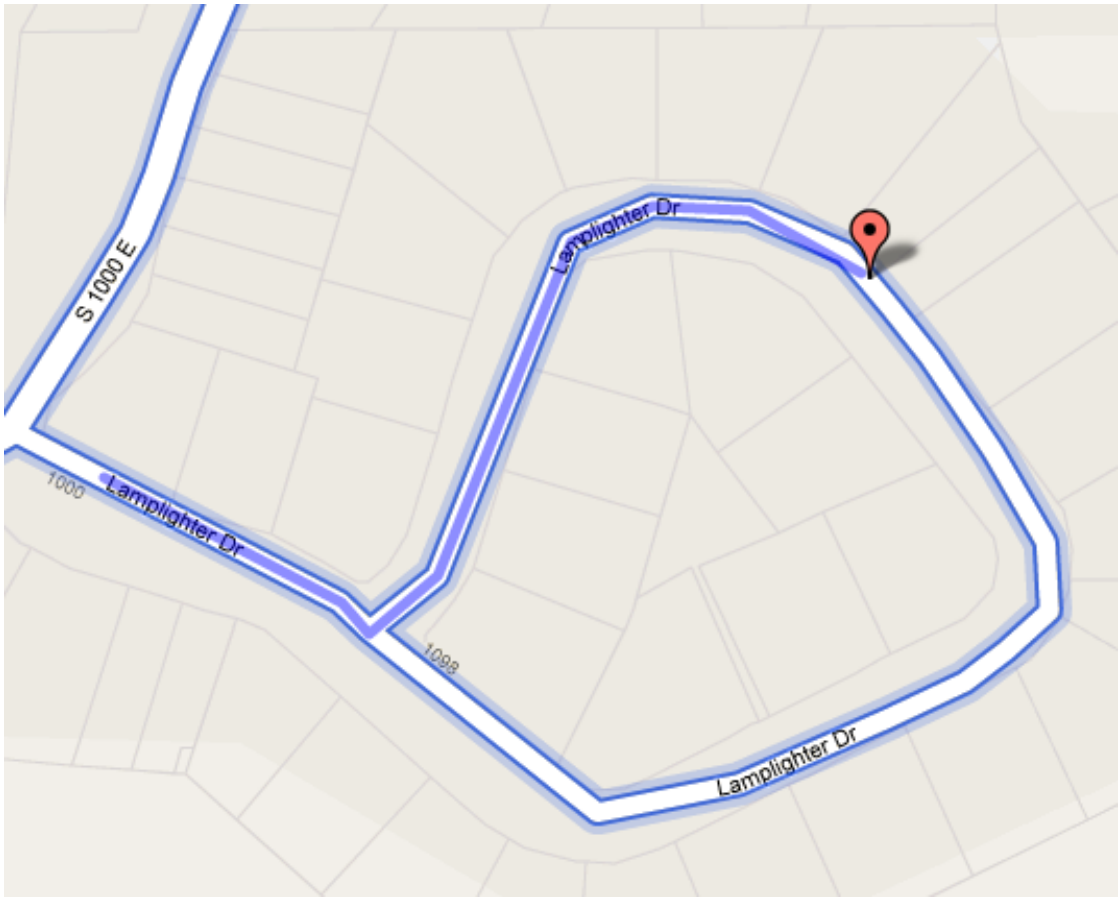

Fig. 7.8: Lamplighter route used to test ability to handle missing links in Street View data.

which will show that the display system and complete Indoor Escape system met the design goals.

Chapter 8

Complete System Testing

Most of the features of the Indoor Escape could be tested in the subsystems. The features that required interaction between the subsystems could not be tested in isolation. The testing of these features is described in this chapter. The features that needed testing were the communication between the display system and micro-server, and communication between the micro-server and sensors. The display subsystem also implements features that required input/output responses from the sensors. These features include adding of a sensor to the system, and changing the active sensor. The final testing was a complete traversing of a course for each type of exercise equipment.

8.1 Communication Between Micro-Server and Display Subsystem

Communication between the micro-server and display system was tested by sending each packet type from the display system and then monitoring the output of the micro-server. The micro-server prints each received packet to the screen, so the reception of these packets could be verified. To test communication the other way, the serial port of the micro-server was connected to another computer via a null modem cable. The other computer ran a terminal and packets were entered into the terminal. In the display system a function that printed the packet to the screen was registered for each of the packet types received. In all cases the packet was printed to the screen in the display system when the address used for the testing was associated. When it was not associated the packets were not received. Through this testing, it was ensured that the communication between the display subsystem and micro-server performed as expected.

8.2 Communication Between Micro-Server and Wireless Receiver

The communication between the micro-server and wireless receiver was verified after testing the communication between the display subsystem and micro-server. This enabled the testing to be run from the display system. The display system was used to send the various packet types. A sensor was connected to the wireless receiver and its operation monitored. The display was then used to send an associate command. After which the display began receiving awake packets, indicating that the sensor was in sleep mode and that communication link between the sensor and display was operational. The display then sent a wake up command. The lights on the sensor changed, indicating it was in active mode and the display began receiving time packets. After testing these commands, it was essential that testing of adding and switching sensors be performed.

8.3 Adding Sensor

To test the adding of a sensor, the configure sensor page was activated. The add elliptical sensor was selected. The join button was then pressed on the elliptical sensor. The display system removed the message instructing me to press the join button and displayed the new elliptical sensor with the default wheel radius. The wheel radius was updated to 2.1 and save clicked. A short time later the system displayed a successful update. This chain of events meant that the system successfully added the sensor. A video of the test can be found in the Appendix.

8.4 Changing Sensors

Changing the sensor was initiated by starting the Indoor Escape system with the active sensor being the elliptical sensor. The configure sensor page was then displayed and the treadmill sensor selected as the active sensor. When the configure sensor page closed, it could be seen that the active sensor had changed to the treadmill sensor and that it status was sleeping. The route Lamplighter was then restarted and the speed of the treadmill varied. As the speed of the treadmill was varied the speed displayed also changed and match that of the treadmill. Thus, it was determined that switching sensors was successful.

A video of the test can be found the Appendix.

8.5 Testing for each Type of Exercise Equipment

The final testing performed was to use the entire system for each type of exercise device. To do this a route was selected and then traversed using a bicycle, treadmill, and elliptical. Videos of each test were taken and are contained in the Appendix. The results show that the system was capable of providing an immersive environment for each of the types of exercise equipment.

8.6 Conclusions

From the individual component testing and the combined system testing it was shown that the objective to create a low-cost, easy-to-use, and retrofitable system that could provide an immersive environment for home exercise equipment was accomplished.

Chapter 9

Conclusion and Future Work

Testing of the system showed that the design goals to provide a low-cost, easy-to-use immersive exercise environment had been met. While the current design of the Indoor Escape meets the design goals there is room for improvement to the system. This chapter contains a review of the goals of the project and their realization. It then proceeds to thoughts on lessons learned during the creation of the Indoor Escape system. It concludes with suggestions of directions future work could take.

9.1 Goals

Cost: The prototype sensors developed cost under $100. The cost for commercially viable sensors would be significantly less, because fewer components would be needed, the sensors could be smaller, and volume purchasing would result in discounted pricing. The sensor and wireless receiver would be the only components the end user would need to purchase. The other two subsystems, the micro-server and display subsystem, are entirely software-based and run on a personal computer. Thus, the goal of low-cost was achieved.

Ease-of-Use: The system was designed to be easy-to-use. This goal was achieved by incorporating many different features. These features include the methods to create a route, to save a route, and retrieve a route. It also includes the method of adding sensors to the system. With a few clicks and the push of a button or two, the user can be up and running with the Indoor Escape system. Thus, the goal of ease-of-use was achieved.

Immersive Exercise Environment: Google Street View was successfully implemented to provide an immersive exercise environment. The system allows the navigation of routes through the images of Google Street View, and thus provides an immersive environment for indoor exercise equipment. However, some routes were found to cause the system to

get stuck in a loop or to lose the route all together. Most routes though where able to be traversed without problem. Thus, there is room for improvement and an improved method of traversing the route is discussed later in this chapter.

Schedule: The system was scheduled to be completed by the first of May 2010; however, implementation was not completed until mid June 2010. The delays were mostly in two areas: implementing the sensors and implementing the display subsystem. The noise induced by the step up converter made detecting the speed of the exercise equipment more difficult than expected. The learning curve for the nRF24LE1 component was also steeper than anticipated, especially managing the different low power states. The final complication was designing and implementing the RF protocol. This proved much more difficult than expected with several designs considered and implemented before arriving at the current solution.

The very large size and number of different technologies used in the display subsystem made its implementation take longer than expected. In addition, methods to debug these technologies had to be learned, further increasing the time it took to implement the display subsystem. Combined, these difficulties added about a month and a half to the total schedule.

9.2 Lessons Learned

While creating the Indoor Escape system many valuable lessons were learned. The most significant lessons include adding testing and verification to design considerations and seeking input from others. These lessons are discussed in the following subsections.

9.2.1 Design for Testing and Verification

The large number and size of the Indoor Escape system software components made it very difficult to implement. The nature of the key components that were used necessitated the use of many different languages and operating environments. The largest challenge of this project was in interfacing the varied components together. This was challenging because it often required that large sections of the system be implemented before any

testing could be done. For example, before adding a sensor to the display subsystem could be tested, communication to the micro-server had to be implemented. This required the micro-server to be implemented, the ActionScript socket to be implemented, and then a terminal program to be used to simulate the sensors. Debugging these large sections of the system was extremely time consuming and difficult. Had testing and verification of the system been considered when designing the system, problems like this could have been minimized and perhaps avoided.

In the future the methods for verification of the system components should be considered when designing the system. This would include research into the best practices for implementing and testing the system during the design phase. It would also require that the system be divided into small enough components that each can easily be implemented and the component tested. In the end that was what happened. A little bit was implemented and then a test devised to ensure that it worked correctly. Then another little bit was implemented and tested. Had the system been designed with testing and verification in mind, the system would have been systematically broken into smaller components and testing methods devised for each small piece.

9.2.2 Seek Input from Others

While many suggestions where received and their impact can be seen in the Indoor Escape system, the Indoor Escape was implemented almost entirely by one individual with very limited interaction with others. Further interaction with others would have improved the design of the system. One example, a custom wireless protocol was implemented because it looked to be the best solution to achieve cost and power usage goals. This involved a significant amount of effort with little gain. In discussing a different project with another group, the ANT protocol was mentioned. Further research revealed that ANT is a low-cost, ultra low-power, wireless communication protocol designed primarily for sports and fitness equipment [42]. The only reason it was not considered was because it was not known to the author of the Indoor Escape system until after significant investment had been made into the custom protocol. Many other improvements to the system may have been made to the

system had the experience and knowledge of others been more extensively sought after in the design of the Indoor Escape.

9.3 Future Work

The Indoor Escape system is a complete system, but there are many additions that could be made that would improve the experience that the system provides. There are other things that could be done that would improve the reliability of the system. Some of the suggestions for future work are mentioned here.

9.3.1 Alternate Source for Immersive Environment Data

Depending solely on Google for the immersive environment restricts the use of the Indoor Escape system. For commercialization of the system other sources of data may be considered which would have less restrictive licensing terms. One such source could be data from Untraveled Road [43]. Untraveled Road has images of many trails in locations of interest such as National Parks and National Monuments. These images follow trails through the parks and are taken at regular intervals. The data set provided by Untraveled Road is not as large as that provided by Google, but its quality exceeds the quality of the Street View images.

9.3.2 Add Multiplayer Features

The Indoor Escape system was designed and implemented in a manner to allow multiplayer features to be added later. Such features could include the ability to share routes. Ideally, this would be incorporated with a way to rank the routes. This could be done by either displaying another list of routes that are created by others or through a social networking site such as Facebook. The recommended model would allow users to post routes to their account and share them with their friends.

Another multiplayer feature that would be useful would be to allow head-to-head competitions. The micro-server was designed so that it could be extended to support display to display communication. To do this, one user would act as a host and invite the other users

to connect to its micro-server. This micro-server would share the route of the host with the other users. As they traversed the route, the other users would sent their locations to the host micro-server which would then distribute them to all of those connected to it. Each display would then post the positions of all the users on the map. The advantage of using the micro-server is that it would not require the use of a central server. Instead each group would handle its own communication. The disadvantage is that it would require each user to open the appropriate port on their firewalls to enable the communication. Opening a port on a firewall can be a very intimidating task to those unfamiliar with them. This may be beyond the technical ability of many users of the Indoor Escape system.

This could be overcome by using a central server and each user sending updates of their position at regular intervals. With each update the positions of the others in the competition could be sent back from the central server.

9.3.3 Adaptations for Commercial Gyms

Commercial gyms have different needs than home users. A gym usually has several machines of many different types. Thus, if the Indoor Escape system was to be deployed as it is currently implemented it would require it be done in one of two ways. The first would be a computer would have to be provided for each machine that uses the system. The second would require the gym to install a micro-server and the members of the gym to bring in their own computers that would then connect to the micro-server. A highly desirable feature for a gym would likely be the ability to run several instances of the Indoor Escape system off one computer. This could be done with the current system except the mouse and keyboard are used to input information. Thus, an input device that would allow several users to access the system simultaneously would be needed.

One potential method would be to create a device similar to the Wii remote controller [44]. This device could then be used to control a cursor that is implemented inside the display subsystem. For text input the input box would be clicked on and a wheel with the characters on a keyboard would appear. Tilting the remote up or down would spin the wheel changing the character selected. Rotating the remote right would select the current

letter and move one character right allowing another character to be input. Rotating left would return to the previous character input. By enabling one system to support multiple instances of the Indoor Escape system, the cost for gyms to deploy the system could be significantly reduced.

9.3.4 Sensor Improvements

For the system to be commercially viable the sensors would need to be improved. While the sensors work well at detecting the rate of travel of the exercise equipment the power consumption could be reduced. In addition, the sensors would have to get regulatory approval of the wireless protocol to be sold. The ANT+ protocol was recently learned about and there are modules available that have already been approved by the regulatory agencies [45].

Reduce Power

Several suggestions can be made to create a lower power sensor in the future. An analysis of the sensors shows that in sleep mode 89 percent of the elliptical sensor's power is used, sleep mode on the treadmill uses 45 percent of the total power, and for the bicycle sensor 73 percent of the power is used in sleep mode. Thus, a reduction in the amount of power required in sleep mode would greatly benefit all of the sensors. The first recommendation toward reducing this power would be to reduce the operating voltage. The operating voltage of 3.3 Volts is a convenient voltage, but many components are available that can operate much lower. The nRF24LE1 can operate at voltages as low as 1.9 V. The power the board consumes with everything switched off approaches 1 mA. The step converter is the primary consumer of this power.

A second suggestion would be to turn the SS411P and QRD1114 on only when sampling them. Currently, the QRD1114 and SS411P are always on when the sensors are in active mode. They could be switched on, sampled, and then turned off. The QRD1114 is estimated to consume about 19mA of current when it is active. This high current consumption enables it to sense the reflector from greater distance. The QRD1114 has a rise time of 10μs and

fall time of 50μs, and thus could be on for 60μs and obtain good measurements. If it was on for 60μs with sampling period of 250μs, it would give a duty cycle of about 24 percent, thus the average current could be reduced by about 14mA. A similar method could be used for the SS411P.

Use a Single Sensor

For production, support, and development costs it would be very desirable to have one sensor that worked for all types of exercise equipment. A single sensor solution could be obtained by using an accelerometer and detecting zero crossings for all sensors. The elliptical already uses this method; by increasing the sample rate the sensor could be adapted to a bicycle. It would then just need to be attached to the spokes of the wheel. For the treadmill it would need to measure the acceleration in the vertical direction near the center of the track. Treadmills are designed to absorb the impact of the runners foot strikes. This is usually done by allowing the track to flex, thus the center of the track oscillates as a person runs on it. This oscillation will create a sinusoidal like signal in the acceleration of the track in the vertical direction. The frequency of the sinusoid will be proportional to the runners speed. By attaching the sensor to the side of the track, the sensor could measure the needed signal without creating a safety hazard for the user.

Thus, all three types of exercise equipment could be measured with the same sensor. For both the bicycle and elliptical, the current elliptical sensor could be used with minor modification, such as increasing the sample rate of the sensor. By rotating the sensor 90° when installing it on the treadmill, the speed of the runner on the treadmill could be measured. This same sensor could also be used with other types of exercise equipment, such as rowing machines, skiing machines, stair climbers, and any other machine that creates a repetitive motion.

Wireless protocol

Testing of the RF protocol showed that it performed well, but testing was very limited and under fairly ideal conditions. Using a developed and verified protocol would be very

beneficial. The ANT+ protocol was designed for sports fitness equipment, uses hardware very similar to the nRF24LE1, and has similar power requirements [45]. It appears to be a very good fit for the Indoor Escape system. The only reason it was not used was it was discovered too late in the design cycle.

Wireless Receiver

For the Indoor Escape system to be commercially viable, the wireless receiver would need to connect to the computer through means other than the serial port. Chips such as FTDI's serial to USB converters could be used and then the current wireless receiver could be used. Another option would be to use the nRF24LU1+ to create a USB receiver [31]. Two other options to be considered are using Wi-Fi or Bluetooth. Personal computers commonly have both of these available. The disadvantage of these protocols is they are more power intensive than the current solution. If Wi-Fi was used the micro-server would not be needed because the sensor could connect directly over TCP/IP. Both Bluetooth and Wi-Fi provide an additional benefit that the current solution cannot. They would have the ability to connect to smart phones and many emerging platforms.

9.3.5 Porting to Other Devices

Porting to iPad

Tablet computers, such as the iPad, provide a very attractive option for running the Indoor Escape system. Their small size and portability would make it easy to arrange so they can be viewed while using exercise equipment. For the Indoor Escape system to run on the iPad two things would need to be changed: the sensors would need a way to communicate with the iPad and it would need a means to display Street View images on it. Street View images require Flash which is not available on the iPad. Google would have to change the way that Street View is delivered and then make an API available to access the data. The iPad lacks a USB or expansion port, but does have Bluetooth and Wi-Fi connectivity. Therefore, the easiest way to get speed data into the iPad would be to convert

the sensors to use Bluetooth or Wi-Fi.

Porting to Android Devices

The current system will not work on the iPhone or iPad because they do not support flash. The Android operating system does support flash and the sensors could be adapted to work with Android devices. To do this, the current display system could be used and run in the web browser. Then, one of two options would need to be selected. The first would be to adapt the sensors to use WiFi and run the micro-server directly on the sensors. The second option would be to use Bluetooth to communicate with the sensors and run the micro-server concurrently with the display subsystem on the Android device. By using an Android device the system becomes portable and would allow a user to take the sensor with them to a gym and use the Indoor Escape system there.

9.3.6 Improved Algorithm for Traversing a Route

The current traversal algorithm can lose the route while traversing it. This is because it obtains the panoId of a corner and heads in the direction of the next corner. It detects that it reached that corner by checking to see it the next image it loads has the same panoId as the next corner. The problem is that when starting at one corner there is no guarantee that there are links to the next corner. Places have been found where links should have been available between the images, but were not. Also some roads have multiple sets of images, and so a location on that road could return different images with each query. If one is not in the chain of links that the next corner is in, the algorithm will never reach the corner and become lost. These challenges presented by the varying and imperfect data from Google were discovered late in the design cycle. Therefore, an improved algorithm is presented, but was not implemented. This algorithm has two advantages. It does not use the links between the images, but instead follows the route and queries at regular intervals for new Street View images. This leads to the second advantage, the algorithm can be used where Street View data is not available. The improved algorithm follows the same steps as the current traversal algorithm, by running an initialization sequence, then loading the

buffer, and then following the route.

Initialization of a Traversal Algorithm

Algorithm 9.1 shows the procedure used to initialize the new traversal algorithm. The initialization algorithm has one required input, a path. It also has several optional inputs. These are input if the route to follow was previously saved. If the route is being restarted, only the route and routeId are given. If the route is being resumed, then additionally the distTraveled, routeIndex, and curLocation from the previous run are input. After initializing the variables, it calls the algorithm to load the buffer. After the buffer is loaded, an interval timer is set to 500ms. In this program the routeIndex that was last crossed is saved.

Loading the Buffer

Loading the buffer is accomplished by using Algorithm 9.2. This algorithm essentially follows the route by determining the bearing between the first location of the first turn of the route and the location of the next turn. It then advances along that bearing in 5 meter increments, querying at each increment to obtain information about the Street View image closest to that location. If Street View information is available, it checks to ensure that the image is different from the previous image received by comparing the pandIds of the images. If the image is new, it saves it to the current frame and then begins looking for the next image by advancing along the path 5 more meters. If the image is not new or not available, it advances again. The algorithm keeps track of how far along the path it moves before finding a new image and saves that distance with the image in the frame. It was decided to advance in 5 meter increments, because through testing, it was determined that the average distance between Street View images is 10 meters. Thus querying at 5 meter increments makes the likely hood of missing one very small.

If the advance moves past a point on the route where the bearing changes, then it queries for an image at the point where the route changes bearings. The algorithm then determines the bearing between the point where it changed bearings and the next point that the direction changes and begins advancing along that bearing. The end of the route

Algorithm 9.1 New initialize route.

Input:
List of waypoints *path*,
Optional Inputs:
Route Id number *savedRouteId*,
Distance Traveled *totalDist*,
Last Location on Route *lastLocation*,
Last Route Index *savedRouteIndex*
State:
Buffer Frames *frames*,
Insert Map *map*,
Current Location on Route *curLocation*,
List of waypoints *route*,
Frame to Load *frameToLoad*,
Last Frame Loaded *lastFrameLoaded*,
Number of Panoramas In Buffer *numInBuffer*,
Route Id number *routeId*,
Distance Traveled *distTraveled*,
Last Route Index *routeIndex*,
Distance Between Street View Images *distanceBetweenPanos*
Output:
Begin
Display message Please Wait While Buffer Loads
If { *routeId* is not input} /* This is a new route */
Begin
Parse *path* removing duplicate adjacent entries
route = *path*
routeId = −1
distTraveled=0
Ask User if they would like to save *route*
routeIndex=0
If{ Save *route*}
Begin
Send route to web server using a AJAX request
/*routeId will be given its value when request completes*/
End
End
algorithm continued below

Algorithm 9.1 cont. New initialize route.

Else if{*routeId* input **and** *routeIndex* input} /* Resuming route */
Begin
routeIndex=*savedRouteIndex*
route[*routeIndex*]=*lastLocation*
distTraveled=*totalDist*
routeId=*savedRouteId*
End
Else /* Restarting route */
Begin
distTraveled=0
routeIndex=0
routeId=*savedRouteId*
End
streetviewClient=New GStreetviewClient
$frames$ = CreateBuffer(10) /* creates a buffer of 10 frames */
Create Inset Map as *map*
Display*route* on*map*
curLocation=*route*[*routeIndex*]
$frameToLoad$=0
$numInBuffer$=0
lastFrameLoaded=-1 //marks that no frame has been loaded
distanceBetweenPanos=0
Call Load Buffer
Remove please wait message
Set Interval Timer to 500ms executing Follow Route every time it expires
End

is reached when it tries to advance past the last point on the route. When this occurs, the frame it is trying to load is marked as the last frame. A critical component of this method is the ability to find the longitude and latitude of a point 5 meters from a longitude/latitude coordinate at a specified bearing. The calculation to perform is given at the movable type scripts webpage [46].

Following a Route

Following a route is achieved by using the Algorithm 9.2 and Algorithm 9.3. When the buffer is loaded, each Street View image is stored with its distance from the previous image. Thus if the frame that is currently being displayed is known, the distance to the

Algorithm 9.2 Improved load buffer.

Input:
State:
Buffer Frames $frames$,
Current Location on Route $curLocation$,
List of waypoints $route$,
Frame To Load $frameToLoad$,
Last Frame Loaded $lastFrameLoaded$,
Number of Panoramas In Buffer $numInBuffer$,
Route Id number $routeId$,
Distance Traveled $distTraveled$,
Last Route Index $routeIndex$,
Distance Between Street View Images $distanceBetweenPanos$,
Current Corner Index $nextCorner$,
Longitude Latitude Coordinates of The Corners of the Route $corners$
Begin
Display message Please Wait While Buffer Loads
While$\{numInBuffer < frames_{length}$ **and** $routeIndex < route_{length}\}$
Begin
Request nearest image to curLocation using streetviewClient
Wait For Request to Complete
If{ Request returns image information }
Begin
If{ Returned panoId $\neq frame[lastFrameLoaded]_{PanoId}\}$
Begin
Load image into $frame[frameToLoad]$
$lastFrameLoaded = frameToLoad$
$frameToLoad = mod(frameToLoad + 1, frames_{length})$
$frame[frameToLoad]_{routeIndex} = routeIndex$
$frame[frameToLoad]_{DistanceTo} = distanceBetweenPanos$
$frame[frameToLoad]PanoId =$ returned Pano Id
$numInBuffer = numInBuffer + 1$
End
End
If$\{routeIndex + 1 < route_{length}\}$
Begin
/*Advance current posistion on route */
$curPov =$ Bearing from $route[routeIndex]$ to
$route[routeIndex + 1]$
$nextLocation =$ Location 5 meters from $curLocation$ in
direction of $curPov$
$segmentDist =$ Distance from $route[routeIndex]$ to
$route[routeIndex + 1]$

algorithm continued below

Algorithm 9.2 cont. Improved load buffer.

```
                nextDistance = Distance from route[routeIndex] to
                  nextLocation
                If{nextDistance ≥ segmentDist} */
                Begin/*next location is past the next turn
                        routeIndex = routeIndex + 1
                        /*set next location point of turn*/
                        nextLocation = route[routeIndex]
                        If{routeIndex = route_length − 1}
                        Begin
                                frame[frameToLoad]_last = true
                        End
                End
                distanceBetweenPanos+ = Distance curLocation to nextLocation
                curLocation = nextLocation
                Request Pano at curLocation
        End
    End
End
```

next frame can be determined. Two distance measurements are kept: the sum of distance traveled, distTraveled, and the sum of the distances between frames including the distance to the next frame, distToNextFrame. When distTraveled is greater than distToNextFrame, the current frame is moved to the back of the buffer. This reveals the next frame on the screen. Algorithm 9.2 is then called to load new content into the frame that was just moved to the back of the buffer.

This procedure would work very well by itself if Street View images were available at all location. Two options were considered to handle the routes that do not have Street View available at all locations. The first was to check the route for missing locations before traversing the route. Time constraints make this approach infeasible. For example, if it a route was created that crossed the United States, and then before it could be used the entire route had to be checked to see if Street View images were available, it would take a very long time and the user would most likely spend all of the time they have available to exercise waiting for the route to be checked. In reality, only a small portion of the route across the

Algorithm 9.3 New advance route.

Input:
List of Longitude and Latitudes of Waypoints Describing a Path *route*,
Longitude and Latitude of a Location on the Route *curLocation*,
Index of Last Corner Taken *index*,
Distance to Advance *dist*,

Output:
Longitude and Latidute of Next Location *nextLocation*
Index of Last CornerTaken *index*,

Begin:
If$\{index + 1 < route_{numInList}\}$// end of route not reached
Begin
/*advance current position on route*/
curPov=Bearing from *route* [*index*] to *route*[*index*+1]
nextLocation= Location *dist* from *curLocation* along bearing *curPov*
segmentDist=Distance from *route*[*index*] to *route*[*index* + 1]
nextDistance =Distance from *route*[*index*] to *nextLocation*
If{*nextDistance* > *segmentDist*} /* next location is past the next turn */
Begin
index + +
nextLocation = *route*[*index*] //sets next location point to turn
End
End
Else
Begin
Return error
End
End

United States could be crossed in one exercise session, thus only that small section would need to be checked before using it. But it is unknown how much needs to be checked, so a method of handling the missing data that works while traversing the route is needed.

The second approach was taken. The loading algorithm measures the distance along the route between Street View images. If there is a section that does not have Street View data, the distance between the image just before and just after that section will be very large. It would be very boring to stay looking at the same image for a long time. It was decided to display the map with a marker showing the progress over the top of the Street View images when the distance between images is larger than 30 meters. The marker in a

manner similar to the advancement of the location used to load the Street View data.

9.4 Conclusion

The Indoor Escape system is a low-cost, easy-to-use, retrofitable system that provides an immersive environment for treadmills, bicycles, and ellipticals. The system was created by implementing three subsystems: the sensor subsystem, the micro-server, and display subsystem.

The sensor subsystem consists of three sensors, one for each of the different types of exercise equipment supported, and a wireless receiver. The sensors are estimated to have at least 19 days of operations on a pair of AAA batteries. Suggestions are made on methods to extend the battery life. Each of the sensors was built for under $100. The cost of the sensors would be the only item that would need to be purchased by the end user.

The micro-server was implemented to enable communication between the sensors and the display subsystem. It was necessary because of the security and abstraction models imposed on the system using web browsers. It connects to the wireless receiver via a serial port and accepts TCP/IP socket connection from display subsystems. The micro-server can then route the data from the sensors to the correct display and vice versa.

The display subsystem provides an easy-to-use interface for the user. Nearly all interaction between the Indoor Escape system and the user are performed through the display subsystem. It provides means to add, select, and configure sensors. It also provides means to create, save, restart, resume, view, and traverse routes. An improved algorithm for traversing routes is suggested for future work. This algorithm will track routes better and handle areas without Street View images in a more elegant way.

In summary, the Indoor Escape system meets the design goals of providing a low-cost, easy-to-use system that provides an immersive environment for indoor exercise equipment. It was designed and implemented from September 2009 to June 2010, taking a about month and a half longer than originally planned. In consultation with the technology and commercialization office, they have expressed optimism at being able to patent at least portions of the system. The original intent was to design and implement a system that enhanced the

experience of using indoor exercise equipment. The Indoor Escape system is successful to that end.

References

[1] S. R. Watterson, W. T. Dalebout, and D. C. Ashby, "Computer systems and methods for interaction with exercise device," U.S. Patent No. 7,060,006, June 2006.

[2] S. R. Watterson, W. T. Dalebout, and D. C. Ashby, "Methods and systems for controlling an exercise apparatus using a portable remote device," U.S. Patent No. 6,997,852, Feb. 2006.

[3] B. D. Andrus, M. Sikes, C. D. G. Robertson, R. Armes, M. J. Slemko, A. G. Maduza, and A. Nieto, "Physical exercise video system," U.S. Patent No. 5,591,104, Jan. 1997.

[4] R. M. Waters, "User interface and methods of using in exercise equipment," U.S. Patent Appl. No. 12/050,883, Mar. 2008.

[5] S. R. Watterson, W. T. Dalebout, and D. C. Ashby, "Systems and methods for interaction with exercise devices," U.S. Patent No. 6,458,060, Oct. 2002.

[6] D. R. McClure, "Interactive fitness equipment," U.S. Patent No. 6,902,513, June 2005.

[7] "ifit live welcome," `http://www.ifit.com/`, Feb. 2010.

[8] "Tacx trainer system software ii," `http://www.tacxvr.com/en/products/tacx-trainer-software-ii`, Feb. 2010.

[9] B. Ewert, "Dynamic real time exercise video apparatus and method," U.S. Patent No. 6,004,243, Oct. 1996.

[10] A. Mimoto, "Biking in place through google streetview," `http://bako.ca/streetview-riding/index.html`, Feb. 2010.

[11] J. D. Neff, M. T. Verona, and J. M. Roane, "Interactive computer simulation enhanced exercise machine," U.S. Patent No. 7,497,807, Mar. 2009.

[12] G. F. Studor, R. W. Womack, M. F. Hilferty, W. B. Isbell, J. A. Taylor, and B. R. Bacon, "Real time simulation using position sensing," U.S. Patent No. 6,152,856, Nov. 2000.

[13] J. Fernandez and J. Fernandez, "Video fitness machine," U.S. Patent No.7,022,048, Apr. 2006.

[14] A. F. Bobick, T. Ulrich, J. Lewis, E. Shepard, and P. Lehman, "Interactive exercise apparatus," U.S. Patent No. 5,890,995, Apr. 1999.

[15] T. P. Arick, "Exercise device independent, variable display rate visual exercise system," U.S. Patent Appl. No. 10/928,554, Apr. 2005.

[16] D. C. Ashby, S. R. Watterson, K. Lorrigan, and W. T. Dalebout, "Systems, methods, and devices for simulating real world terrain on an exercise device," U.S. Patent Application No. 12/413,362, Oct. 2009.

[17] "Calculate distance, bearing and more between latitude/longitude points," www.movable-type.co.uk/scripts/latlong.html, July 2010.

[18] "Google maps/google earth apis terms of service," http://code.google.com/apis/maps/terms.html, Mar. 2010.

[19] "PeaceKeeper, Acid3, and DOM," http://www.tomshardware.com/reviews/firefox-chrome-opera,2558-8.html, July 2010.

[20] "WebKit SunSpider JavaScript benchmark results," http://ie.microsoft.com/testdrive/benchmarks/SunSpider/Default.html, July 2010.

[21] "Bing maps," http://msdn.microsoft.com/en-us/library/dd877180.aspx, July 2010.

[22] "Keeping up facebook," http://blog.facebook.com/blog.php?post=7899307130, July 2010.

[23] "Wikimedia servers - meta," http://meta.wikimedia.org/wiki/Wikimedia_servers, July 2010.

[24] "Socket - actionscript 3.0 language and components reference," http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/net/Socket.html, July 2010.

[25] "Google maps javascript api v2 reference," http://code.google.com/apis/maps/documentation/javascript/v2/reference.html, July 2010.

[26] Fairchild Semiconductor, "QRD1113, QRD1114 reflective object sensor," http://www.nordicsemi.com/files/Product/data_sheet/nRF24LE1_Product_Spec_v1_4.pdf, Jan. 2008.

[27] Honeywell, "SS311PT/SS411P," http://mouser.com/catalog/specsheets/ProductSheet_SS311_411.pdf, Oct. 2009.

[28] Freescale Semiconductor, "MMA7361L ±1.5g, ±6g Three Axis Low-g Micromachined Accelerometer," http://www.freescale.com/files/sensors/doc/data_sheet/MMA7361L.pdf, May 2006.

[29] "Wireless buying guide," http://www.sparkfun.com/commerce/tutorial_info.php?tutorials_id=128, July 2010.

[30] "Product overview," http://www.nordicsemi.com/index.cfm?obj=menu&act=displayMenu&men=21, July 2010.

[31] Future Technology Devices International Ltd, "FT232R USB UART IC," http://www.ftdichip.com/Documents/DataSheets/DS_FT232R.pdf, July 2010.

[32] Nordic Semiconductor, "nRF24LE1 ultra-low power wireless system on-chip solution," http://www.nordicsemi.com/files/Product/data_sheet/nRF24LE1_Product_Spec_v1_4.pdf, May 2010.

[33] ST Microelectronics, "L6920DB Synchronous rectifier step up converter," http://www.st.com/stonline/books/pdf/docs/11378.pdf, May 2008.

[34] Fairchild Semiconductor, "NC7WB66 low voltage dual spst normally open analog switch or 2-bit bus switch," http://www.fairchildsemi.com/ds/NC/NC7WB66.pdf, Dec. 2005.

[35] Noric Semiconductor, "Bill of materials for nRF24LE1-Q48-DK," NRF24LE1-Q48-DK CD-ROM, Nov. 2009.

[36] Noric Semiconductor, "nRF24LE1-Q48-DK module schematic," NRF24LE1-Q48-DK CD-ROM, Nov. 2009.

[37] SparkFun Electronics, "SFE footprint library eagle," http://www.opencircuits.com/SFE_Footprint_Library_Eagle, Mar. 2010.

[38] "33 Each-2-Layer," http://www.advancedcircuits.com/index.php?load=content&page_id=130, Apr. 2010.

[39] "Product comparison - NordicTrack," http://www.nordictrack.com/webapp/wcs/stores/servlet/ProductCompareView?langId=-1&storeId=10301&catalogId=12401&categoryId=59002&compare=79646&compare=134503&compare=149703, July 2010.

[40] Mouser Electronics, "Consumer and sealed lead-acid battery finder," http://www.mouser.com/catalog/catalogUSD/641/2066.pdf, July 2010.

[41] Adobe Systems, Inc., "Adobe cross domain policy file specification," http://learn.adobe.com/wiki/download/attachments/64389123/CrossDomain_PolicyFile_Specification.pdf?version=1, Jan. 2010.

[42] "Ant – your wireless sensor network solution," http://www.nintendo.com/wii/console/controllers, July 2010.

[43] "Untraveledroad, virtual travel for the hiker, traveler and outdoorsman," http://www.untraveledroad.com/, July 2010.

[44] "Controllers at nintendo," http://www.thisisant.com/why-ant/market-applications, July 2010.

[45] Dynastream Innovations Inc, "Ap2 rf transceiver module," http://www.thisisant.com/images/Resources/PDF/ap2_rf_transceiver_module_datasheet.pdf, July 2010.

[46] "Where is street view? — google maps with street view," http://maps.google.com/help/maps/streetview/where-is-street-view.html, July 2010.

Appendix

Sensor Software

The sensor software can be found under the *Sensor Software* folder on the enclosed data CD. This folder contains several sub directories that each contain the software for the components of the sensor subsystem.

Wireless Reciever

The Keil μVision project for the Wireless Receiver can be found in the *Wireless Receiver* folder.

Common Software

The source code of the software that is common to all sensors can be found in the sub folder *common*.

Treadmill Sensor

The Keil μVision project for the treadmill sensor can be found in the *Treadmill Sensor Rev 1* folder.

Elliptical Sensor

The Keil μVision project for the elliptical sensor can be found in the *Elliptical Sensor Rev 1* folder.

Bicycle Sensor

The Keil μVision project for the bicycle sensor can be found in the *Bicycle Sensor Rev 1* folder.

Sensor PCB Design

The Eagle files used to create the sensor PCB are contained in the folder *Sensor PCB* on the enclosed CD. It contains three subfolders.

EAGLE Library

The subfolder *Eagle Library* contains the library of parts created to while designing the sensor PCB.

Sensor PCB Eagle project

The subfolder *Sensor PCB* contains the Eagle project that was created to implement the sensor PCB.

Sensor Bill of Materials

The subfolder *Bill of Materials* contains the bill of materials used to create the sensors for the Indoor Escape system.

Micro-Server Source Code

The source code used to implement the micro-server can be found in the *Micro-Server* folder of the enclosed data CD.

Display Subsystem Software

The source code used to implement the diplay subsystem can be found in the *Display* folder of the enclosed data CD. This folder contains serveral subfolders. These subfolders contain different portions of the display subsystem.

Web Application

The web application part of the display subsystem can be found in the subfolder *www*. These are the files that are served from the web server.

ActionScript Socket Application

The Action Script Socket Flash Builder 4 project can be found in the subfolder *Socket Connect*.

Database Configuration

The files used to set up the database used by the display subsystem are contained in the subfolder *Database*.

Testing Videos

Much of the testing of the system was documented by creating videos. These videos are found in the folder *Testing Videos*. The folder contains two folders *Display Subsystem* and *Complete System*. The first contains the videos from testing the display subsystem

in isolations. The folder *Complete System* contains the videos from testing the entire system.

Logan to Garden City

The video taken during traversal of the route Logan to Garden City is contained in the subfolder *Display Subsystem* and is named *LoganToGardenCity*.

River

The video taken during traversal of the route River is contained in the subfolder *Display Subsystem* and is named *River*.

Center to Riverside

The video taken during traversal of the route Center to Riverside is contained in the subfolder *Display Subsystem* and is named *CenterToRiverside*.

Lamplighter

The video taken during traversal of the route Lamplighter is contained in the subfolder *Display Subsystem* and is named *Lamplighter*.

Adding Sensor

The video taken during demonstrating a added a sensor to the Indoor Escape system is contained in the subfolder *Complete System* and is named *AddingSensor*.

Changing Sensor

The video taken during demonstrating a changing a sensor in the Indoor Escape system is contained in the subfolder *Complete System* and is named *ChangeSensor*.

Treadmill Test

The video taken during demonstrating a the complete system using a treadmill is contained in the subfolder *Complete System* and is named *TreadmillTest*.

Elliptical Test

The video taken during demonstrating a the complete system using an elliptical is contained in the subfolder *Complete System* and is named *EllipticalTest*.

Bicycle Test

The video taken during demonstrating a the complete system using a bicycle is contained in the subfolder *Complete System* and is named *BicycleTest*.