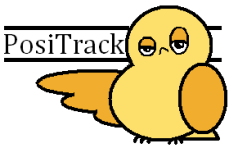


School of Engineering Science • Burnaby • BC • V5A 1S6

March 11th, 2010

Dr. Andrew Rawicz
School of Engineering Science
Simon Fraser University
Burnaby, BC
V5A 1S6

Re: ENSC 440 Design Specifications for a System to Track Athlete Performance

Dear Dr. Rawicz:

The attached document outlines the design specifications for a diagnostic tool that tracks athlete performance. This proposed product is called the PosiTracker and it can be used by large sports teams as both a broadcasting tool and a training tool. Millions of dollars are spent every year on both player salaries and broadcasting rights, our product can protect this investment through enhanced training and an improved viewer experience.

This document outlines the proof-of-concept design specifications for the PosiTracker System and each sub-component. The goal of the document is to provide a design that implments the all of functionality descibed in our functional specifications for a proof-of-concept device. An overview of the whole system is first given before it is broken down into sub-components and discussed in detail. The design specifications for a final product are not included but will be discussed in each section.

PosiTrack Systems consists of four hard-working and motivated team members: Andreea Hrehorciuc, Jaime Valdes, Jeff Anderson, and Ryan Lynne. Should you have any questions or comments about the design specifications please feel free to contact me by phone or email.

Regards,

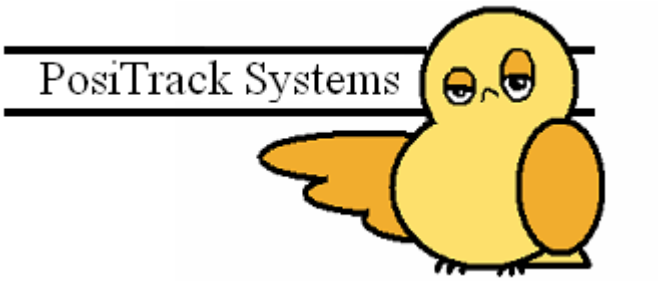
Ryan Lynne

Ryan Lynne
1.778.840.9111
posi.track.systems@gmail.com
PosiTrack Systems

Enclosure: Design Specifications for a System to Track Athlete Performance

Design Specifications for a

System to Track Athlete Performance



A highly precise diagnostic tool to analyze all aspects of an athlete's performance

Project Team: Andreea Hrehorciuc
Jamie Valdes
Jeff Anderson
Ryan Lynne

Contact Person: Ryan Lynne
posi.track.systems@gmail.com

Submitted to: Dr. Andrew Rawicz
School of Engineering Science
Simon Fraser University

Issued date: March 11th, 2010

Revision: 16

Executive Summary:

The attached document outlines the proof-of-concept design specifications for a system to track athlete performance using location, acceleration, heart rate, and speed. This proposed product is called the PosiTracker and it can be used by large sports teams as both a broadcasting tool and a training tool. Millions of dollars are spent every year on both player salaries and broadcasting rights, this product can protect this investment through enhanced training and an improved viewer experience.

The goal of the document is to provide the PosiTrack team with a design that implements the functional specifications described in [1] for a proof-of-concept device. An overview of the whole system is first given before it is broken down into sub-components and discussed in detail. Finally, a test plan is provided to ensure that all of the functionality requirements have been met.

Listed below are the key design decisions for the proof-of-concept device. Justification of each decision can be located in the relevant sections of this document.

1. The localization technique will use a combination of Received Signal Strength from wireless Access Points and inertial measurements from accelerometers/gyroscopes.

2. The Gumstix Overo Fire Computer on Module will be used in the Diagnostic Tool located on each athlete. It will measure the wireless signal strength and inertial measurements from the accelerometers/gyroscopes.

3. The Gumstix Tobi expansion board will connect to the Overo Fire to allow for debugging/development of software and to allow for the Inertial Measurement Unit containing the accelerometers/gyroscopes to be attached.

4. The TCP/IP protocol in combination with the use of Dynamically Linked Libraries will create the connection between the Diagnostic Tool located on the athlete and the Graphical User Interface.

5. The visual basic language will be used to create the Graphical User Interface.

6. Standard wireless Access Points will be used to generate the measurable wireless signals and to also create a wireless link between the Graphical User Interface and the athlete.

PosiTrack Systems is committed to the production of the PosiTracker prototype, using the design specifications outlined in this document. The completion of the proof-of-concept device is expected during the week of April 16th 2010.

Table of Contents:

List of Figures:

List of Tables:

List of Objects:

List of Functions:

Glossary and Acronyms:

A/D	Analog to Digital
Actions	Athlete's movements and biological information such as heart rate
Anchors	A signal source with a known location
AOA	Angle of Arrival
AP	Access Point (see Anchors)
COM	Computer on Module
CPU	Central Processing Unit
Dead Reckoning	Using current direction and speed to determine future location
DLL	Dynamic Linked Library
DOF	Degrees of Freedom
DT	Diagnostic Tool
EM	Electro-Magnetic
GPS	Global Positioning System
GUI	Graphical User Interface
HDMI	High Definition Multimedia Interface
I2C	Inter-Integrated Circuit
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Measurement Unit
IP	Internet Protocol
OS	Operating System
RAM	Random Access Memory
RSS	Received Signal Strength
RSSI	Received Signal Strength Index
SD	Secure Digital
Sensor Fusion	To combine multiple measurements to increase measurement accuracy
SSH	Secure Shell
SPI	Serial Peripheral Interface

TCP	Transmission Control Protocol
TOA	Time of Arrival
UART	Universal Asynchronous Receive Transmitter
USB	Universal Serial Bus
VB	Visual Basic
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network

1. Introduction

The PosiTracker is a system that tracks the relevant statistics of athletes during game play or training. The system is targeted towards large sports teams and can be used for physical training, coaching or broadcasting. This system contains three main sub-components: the Diagnostic Tool, the Access Points, and the Graphical User Interface. The Diagnostic Tool will be worn by the athletes in order to gather information such as location, acceleration, heart rate, and speed. This will be sent wirelessly back to a computer station running the Graphical User Interface. The Diagnostic Tool is able to determine it's location by measuring the signal strength of the various Access Points. Throughout this document the Diagnostic Tool, Access Points, and Graphical User Interface will be referred to as the DT, APs, and GUI.

1.1. Scope

This document provides a design implementation for the functional requirements provided in "*Functional Specifications for a System to Track Athlete Performance*" [1]. This document specifies requirements for both a proof-of-concept device and a final end product. This design will only implement the requirements for the proof of concept device; however, the end product requirements will be considered in order to allow for a smooth transition from the proof-of-concept design to the final product design. Functional requirements marked as [1-X], and [2-X] outlined in [1] will be implemented into this design.

1.2. Intended Audience

This design specification document is intended to be used by the members of the PosiTrack team in order to create a working proof-of-concept device. Its purpose is to provide a unified vision for the team as well as to provide design guidelines for the system as a whole and for each sub-component. It is also hoped that by using this document the PosiTrack team will reduce the overall integration time of the systems sub-components.

2. Full System Design

This section contains a complete overview of the system as a whole and will cover the following: desired functionality, design overview, localization methods, device layout, and software design. The subsequent sections will dive more in-depth looking at each component and its design. All of the sections will discuss the proof-of-concept design and the justifications for the design choice. The sections will also briefly touch on the production version and will be clear as to what will not be included in the prototype version. The final section will outline test plans to ensure a properly functioning device.

2.1. System Functionality

The proposed system has the desired functionality as outlined in the document *Functional Specifications for a System to Track Athlete Performance* created by PosiTrack Systems [1]. This functionality is summarized in the black box diagram of Figure 1. The user will input their physical goals and then try to match or surpass their goals. The system will analyze the athlete's actions and prompt to the user's level of achievement, being that his goal has been met or not. With the time constraints of this projects our proof-of-concept model will only analyzes the athlete in what is called "Game Analysis". The objective training software will not be produced for this version of production.

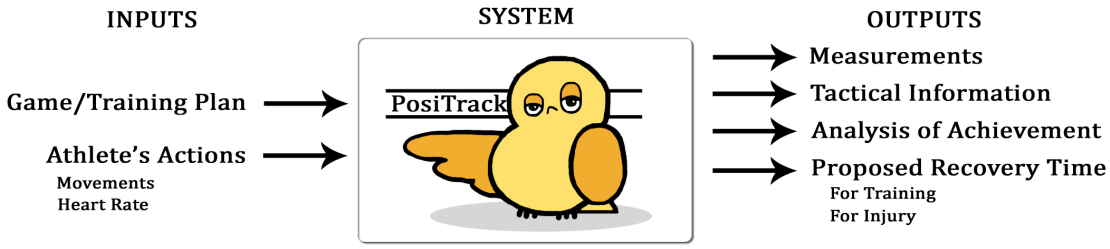


Figure 1: High level functional view of the PosiTrack system.

This system must be able to track the athlete's position, speed, and acceleration. For future production models the system will include the ability to measure biological information such as heart rate. The following sections outline the designed system intended to give these functionalities.

2.2. Design Overview

Our system to track an athlete's performance consists of a Diagnostic Tool (DT), Access Points (APs) and a Graphical User Interface (GUI). The DT is a small electronic device that will be placed on the athlete and will perform the desired measurements. APs are wireless routers that conform to the IEEE 802.11 Wi-Fi protocol. The DT measures the

Wi-Fi signal strength from APs placed around the playing area to determine position at a given time. The DT also measures the linear and rotational acceleration of the athlete. The raw data generated by the DT will be sent via a WLAN to a host computer. This host computer will be running the GUI software and receive the data in real time. The GUI takes the acquired data and generates useful information such as location, speed and acceleration. Figure 2 below outlines the devices discussed. For this proof-of-concept prototype the system will track only a single DT however the design allows for the ability to extend the system to track more than one.

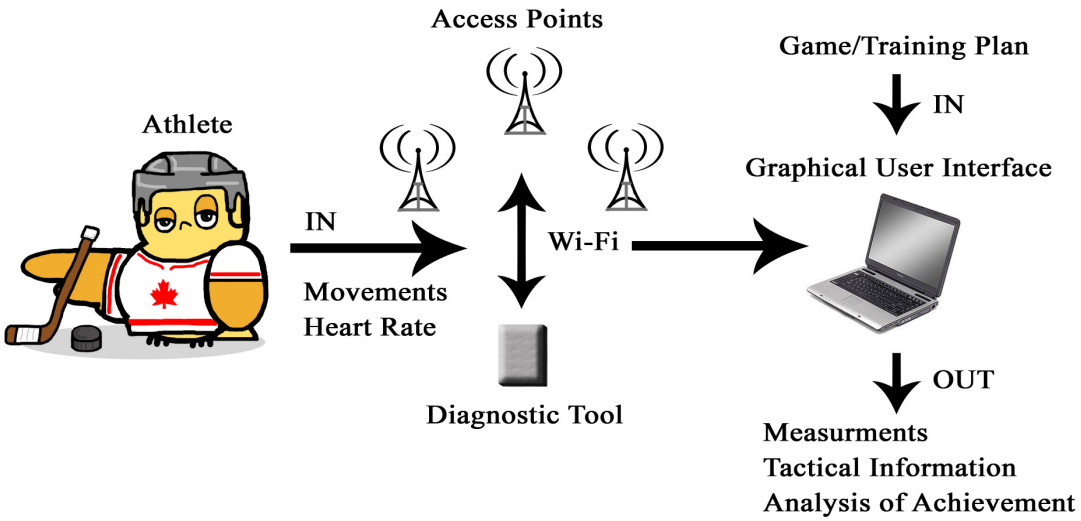


Figure 2: Design overview of the PosiTrack system[2].
The large antennas in this figure represent wireless APs.

The sub-component design section will explain in detail the design of each component including the DT, the APs and the GUI. Section 2.4 discusses the proposed design layout of the APs for optimum localization. Section 2.5 touches on the software of the system as a whole. The communication between devices will be discussed and as well as the program flow. The following section discusses the chosen localization technique, alternatives and justification for the choice.

2.3. Localization Methods

Tracking an object's position can be achieved in a multiple of ways. Fundamentally, the objects position has to be measured relative to something else and this measurement can be made via an array of distance measurement tools. In a sports situation the measurement system cannot obtrude the player's path or play and the number of measurements must be great enough to give the velocity and acceleration of the athlete with reasonable accuracy. Another system constraint is the ability to measure biological information such as heart rate. This section will discuss time of arrival, angle of arrival, video processing, power triangulation, and dead reckoning as possible localization

methods. The final localization choice will be presented and an estimate of localization accuracy will be made.

Time of Arrival (TOA) is a promising localization method for our application. This method involves measuring the wave propagation time from the location of interest to a beacon or an AP. Through three or more of these measurements the location can be resolved using triangulation. Typically sound or electromagnetic waves are used because of their predictable velocity in the medium that they travel. Both of these methods have been researched and limitations were found for both methods.

Using acoustic measurements for TOA is quite appealing because of its high accuracy (<10cm) [3]. The potential problems that arise from this method are range limits, expensive transducers, and directionality constraints. Long range ultrasonic transducers were sourced to be $400 and up with ranges of 11m to 15m [8]. The long range transducers were also found to be highly directional which means large arrays would have to be used. These qualities are not suited for the application of tracking athlete's in an approximate area of 60m by 30m.

Currently athlete tracking systems use GPS for localization so it is natural to discuss the possibility of using EM wave TOA like GPS employs. The limitations of this system are today's computer speeds. Because EM waves travel at such a high speed the either the distance the wave travels has to be large or the clock to measure the time of propagation has to have high resolution. The current top of the line CPUs can process just fast enough to measure the time difference across a 100m distance however the cost does not fit our projects budget. For this reason this method of localization was not chosen.

Angle of Arrival (AOA) was considered to locate the player. This technique uses the directionality control of an array antenna system to find the angle the location of interest is relative to the AP. Using two or more of these type of APs the location can be found. It has been shown that the 50% of location measurements are within 0.5m of real position [4]. This accuracy is good for the desired application and the system meets range requirements however the increased complexity of the APs deliver an unwanted burden to an already tight schedule.

To create a system that uses video processing to generate acceleration, speed, and location is possible with excellent accuracy. Although this type of system can do this well it cannot generate biological data such as heart rate. For this reason the technique was not chosen.

The chosen method for localization is EM radiation signal strength measurements with sensor fusion. EM radiation from a source reduces in power predictably as a function of distance [9]. This predictability can be used to generate a relative distance measurement from the source (Access Point) to the location of power measurement. Using three or more APs the position of the measurement location can be obtained. It has been stated [9]

that using a Received Signal Strength (RSS) for an uncalibrated system in an indoor and outdoor environment gives a location accuracy of 10m and 5m respectively. This information is presented in Table 1. With a calibrated system the accuracy can reach within 3m. This accuracy is not as desirable as other localization methods however with different techniques the accuracy can be improved. Other reasons will also be discussed as to why RSS is best suited for this system.

Table 1: The accuracy of various localization methods

Localization Method:	Outdoor Accuracy	Indoor Accuracy
RSS - uncalibrated [9]	5m	10m
RSS - calibrated [9]	3m	3m
RSS - Wi-Fi with propagation model [6]	-	61% - 1.5m 94.1% - 3m
RSS - Wi-Fi with Sensor Fusion excluding Dead Reckoning [7]	-	50% - 0.5m 90% - 2m
RSS - Wi-Fi with Sensor Fusion including Dead Reckoning [5]	-	0.4m

The main design consideration when building an RSS localization system is the type of antenna along with the frequency of radiation. Three types of antenna were considered: custom, Wi-Fi 802.11, and Bluetooth. The custom antenna was not the best choice because of the time and budget constraints of this project. Bluetooth is a popular wireless connection method however the range is a concern for this application. Wi-Fi, the IEEE 802.11 protocol, was chosen because of its popularity, its ability to cover the appropriate range with an appropriate number of APs and its secondary application as a communication method. This protocol is implemented by a large number of chips on the market and WLAN routers are cheaply available. This makes this protocol ideal to generate a proof-of-concept model with the time and budget constraints. As an added benefit, this protocol can also be used to send the measured data from the athlete to the GUI. There has been a large of amount of work using Wi-Fi for localization, and some of that work is presented in Table 1. The work in [6] [7] [5] shows that Wi-Fi localization accuracy can be improved using an appropriate propagation model [6] and a probability filter, using senor fusion [7], and using dead reckoning [5]. Although [5] uses an

odometer for dead reckoning it has been shown that accelerometers have been used to dead reckon a person's position quite well [10].

To summarize, the localization design choice is RSS using Wi-Fi with a combination of sensor fusion and inertial measurements to dead reckon the position of the athlete. This document will outline hardware, software and other designs that are integral to this project. The plan for testing this product will be outlined at the end of the document. The next section explains the layout of the system, discussing the position of the APs for the best localization accuracy.

2.4. Device Layout

In order to track the DT in the method described in the previous section, the system needs three APs for two-dimensional localization and four APs for three-dimensional localization. This project will be only concerned with 2D and not 3D tracking because of budget and time constraints. The number of access points can be expanded in future iterations if desired.

The access points should be placed in locations that give direct line of sight to the athlete in order to create propagation model simplicity. The APs should also be placed in a manner as to cover the playing field with their signal and the APs should also span a 2D space. These statements are required to allow the system to locate the player in all space of the playing field and fundamentally allow the localization to occur in 2D. Ideally the APs should be placed in the field so that they are all as far apart from each other as possible while still providing coverage to the whole field. An example of AP placement is shown in Figure 3. This figure shows the AP on the edge of the playing field giving direct line of sight while still allowing of full coverage of the playing field.

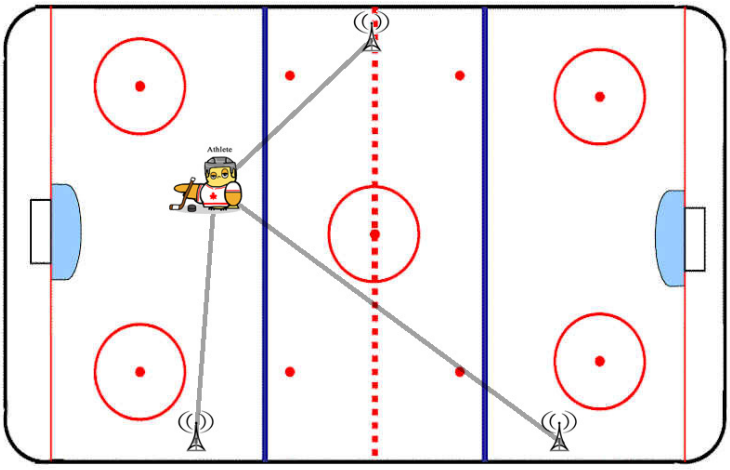


Figure 3: Example layout of PosiTracker in playing area. The large antennas in this figure represent wireless APs.

2.5. Software

The system consists of software designed to operate on the Diagnostic Tool and the Host Computer (the GUI).

The GUI is created in two parts. The first part being the user interface created with Visual Basic, VB. The second module is a Dynamic Linked Library written in C++ and C. The DT consists of an executable written in C++ and C. Messages will be sent both directions from the GUI DLL to the DT program. The message passing will be achieved by using the internet protocol suite specifically, Transmission Control Protocol, TCP. Figure 4 shows the software blocks.

The user interface is written in VB because of its ability to create complex interfaces rapidly. The user interface is connected to a DLL library written in C/C++ because of that language's flexibility. This is the same reason for creating the DT executable in that language. The messages are passed using TCP because our hardware's ability to perform this operation and the current popularity of TCP/IP. Other languages and message passage methods were reviewed however the choice suited the members of the PosiTrack in terms of abilities.

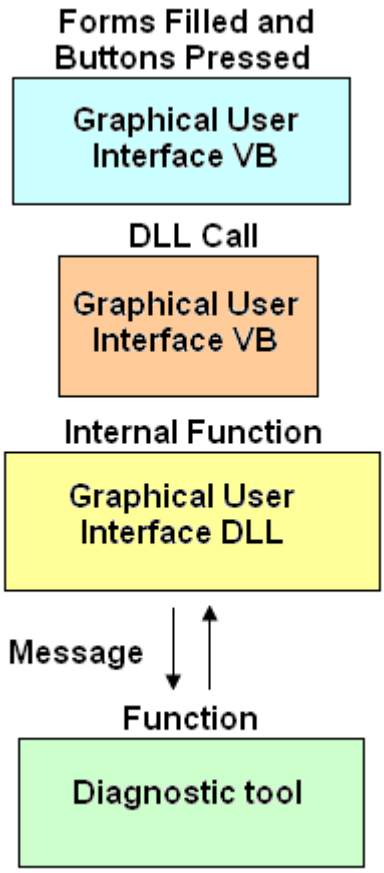


Figure 4: Software Blocks and communication of the PosiTracker

This section explains the high level communication of the software components and the sequence of operation that occurs inside and outside the program. The messages that are passed are discussed. Data structures that are common to all software are presented.

2.5.1. Program Flow

The program flow is presented in tables 2-5 below. Table 2 summarizes the GUI connecting to the DT while creating the object CAthlete. Table 3 outlines operation to allow the GUI to receive the AP identification that the DT senses and let the users input

AP locations and transmitted signal strengths. This information is crucial for localization. Sampling procedures are outline in table 4 and Shut down procedures are stated in table 5. All of the tables relate the user-GUI interactions with the functions that are called within the software modules. All functions are described in the sections below. Refer to the list of functions and objects for quick referencing. The next section presents a detail discussion on the message passed between the GUI DLL and the DT program.

Table 2 outlines the software start up, athlete creation and connection procedures

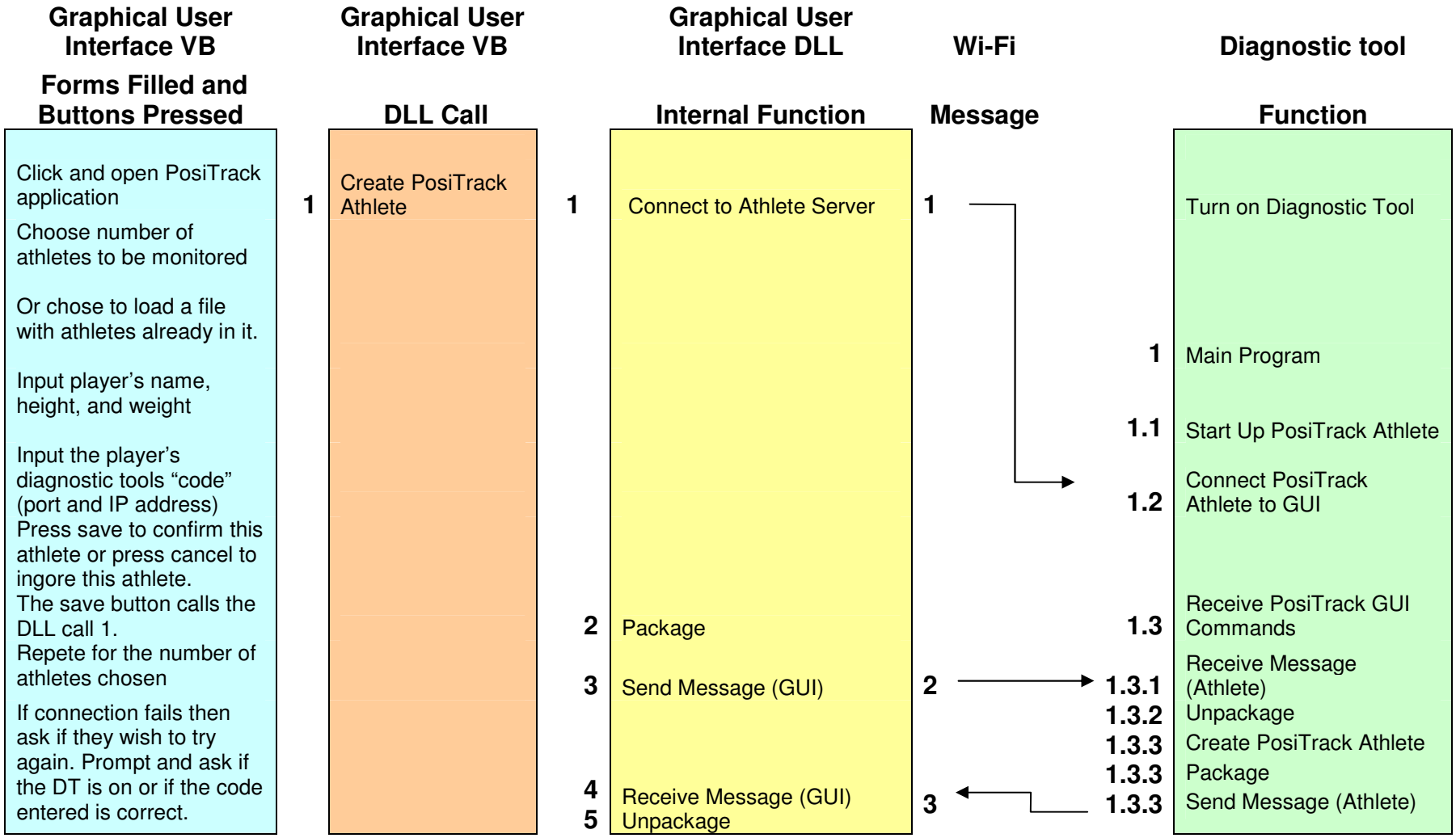


Table 3 outlines operation to receive the AP ID and input AP location and transmitted signal strength

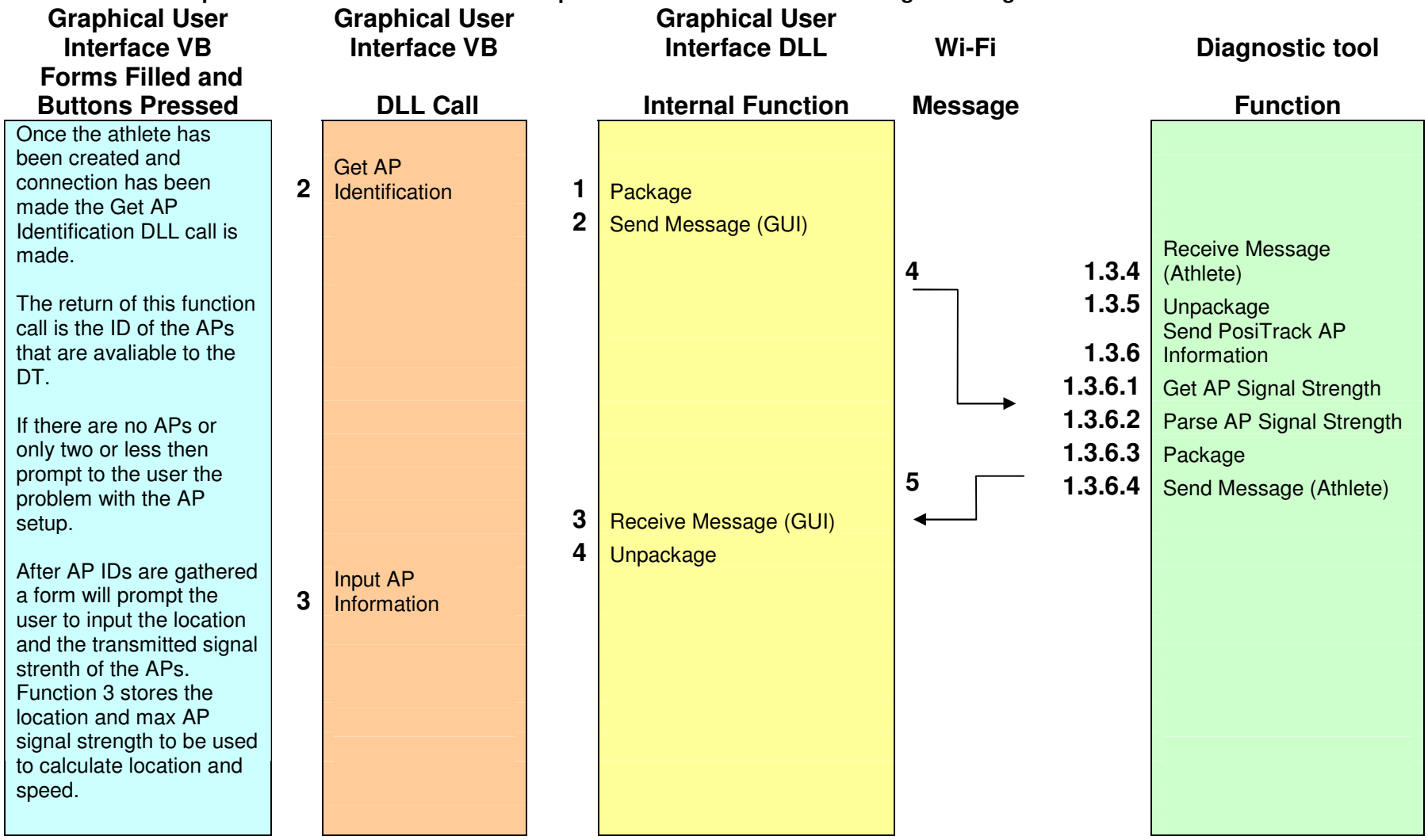


Table 4 outlines the systems sampling procedure

Graphical User Interface VB Forms Filled and Buttons Pressed		Graphical User Interface VB DLL Call		Graphical User Interface DLL Internal Function		Wi-Fi Message		Diagnostic tool Function
If all setup is success full then sampling may begin. To start sampling the user must be on the correct athlete page and must press the start sampling button. (see the GUI section) If the sample takes to long to aquirer then the fuction will time out and a new sample request will be sent. This function will return a success or failure status on the sample that is being taken. If the sample is successful then the location and speed will be calculated and the information will be displayed to the user	**4**	Sample	**1**	Package			**1.3.7**	Receive Message (Athlete)
			2	Send Message (GUI)	**6**	→	**1.3.8**	Unpackage
							1.3.9	Take PosiTrack Sample
							1.3.9.1	Get AP Signal Strength
							1.3.9.2	Parse AP Signal Strength
							1.3.9.3	Get Acceleration
							1.3.9.4	Parse Acceleration
							1.3.9.5	New Sample
							1.3.9.6	Send PosiTrack Samples
							1.3.9.6.1	Package
			3	Receive Message (GUI)	**7**	←	**1.3.9.6.2**	Send Message (Athlete)
			4	Unpackage				*If the number of samples exceeds MAX_SAMPLES then call:*
			5	New Sample				Append PosiTrack Athletes Samples to File
								Or If no file exists
							1.3.9.7	Write PosiTrack Athletes to File

using functions 5, 6, and 7 to gather the data.	5 6 7	Get Acceleration Generate Location Generate Speed	6	*If the number of samples exceeds MAX_SAMPLES then call:* Append PosiTrack Athletes Samples to File *Or If no file exists* Write PosiTrack Athletes to File See Section 3.3.4.2. Algorithms		1.3.9.8	*If the number of samples exceeds MAX_SAMPLES then call:* Delete All Samples

Table 5 outlines the system's shut down procedures

Graphical User Interface VB Forms Filled and Buttons Pressed		Graphical User Interface VB DLL Call		Graphical User Interface DLL Internal Function	Wi-Fi Message		Diagnostic tool Function
When the program is shutdown the GUI sends a shut down message to the DT. The DT response by removing its data and replying and exiting the program. The GUI then deallocates its memory using 9 and 10.	8	Shutdown Diagnostic Tool	1 2	Package Send Message (GUI)	8 →	1.3.10 1.3.11 1.3.12 1.3.13	Receive Message (Athlete) Unpackage Delete PosiTrack Athlete Package
	9	Delete Athlete	3 4	Receive Message (GUI) Unpackage	9 ←	1.3.14 1.3.15	Send Message (Athlete) Disconnect to GUI Client
	10	Delete Samples					

2.5.2. Messages

Defined in the tables 2-5 above are messages that are passed between the GUI and the DT. These messages are sent over a Wi-Fi connection that uses TCP/IP to pass data packets. In this section the packets will be defined.

All packets start with a command byte which tells the program what actions should be taken. These commands are defined in table 6 below. Table 7 describes each message in tables 2-5. The message number in table 7 corresponds to that in the tables above.

Table 6 defines the message commands

Command	Value (Hexadecimal)
END_SAMPLE	0x00
ATHLETE_INFO	0x01
AP_INFO	0x02
SIGNAL_STRENGTH	0x04
ACCELERATION_LIN	0x08
ACCELERATION_LIN_ERROR	0x10
ACCELERATION_ROT	0x20
ACCELERATION_ROT_ERROR	0x40
HEART_RATE	0x80
SHUTDOWN	0xFF
START_SAMPLE	(SIGNAL_STRENGTH | ACCELERATION_LIN | ACCELERATION_ROT | HEART_RATE | ACCELERATION_LIN_ERROR | ACCELERATION_ROT_ERROR)

Table 7 defines the messages sent

Message	Command	Description
1		The first message is not actually a message but a connection that is being made.
2	ATHLETE_INFO	This message sends the athlete information to the DT so the athlete class can be created on the DT. This functionality will be useful is a distributive system is created in the future. A distributive system is one in which the DT will generate the location it self through calculations.

3	ATHLETE_INFO	This is a confirmation message that the Athlete has been created and the DT is connected.
4	AP_INFO	This message from the GUI to the DT is a request for the AP IDs that are sensed by the DT.
5	AP_INFO	The DT sends this message to the GUI with the AP IDs it has sensed. This is the response message to the previous request.
6	START_SAMPLE	The GUI is requesting a sample with this message.
7	START_SAMPLE	The DT responds to the GUI request with the data that is in requested in the START_SAMPLE command. This command can be defined in different ways to return different data such as signal strength, and acceleration.
8	SHUTDOWN	This message requests the DT to shutdown its program and deallocate memory.
9	SHUTDOWN	This is the confirmation message that the DT will shutdown its program and deallocate memory.

Message time outs are handled by the Send Message and Receive Message functions. For all commands other than START_SAMPLE the message will time out after 5 seconds and the message will be resent 3 times before external action takes place. For AP_INFO, and ATHLETE_INFO after three retries the GUI will prompt the user and error message. For the command SHUTDOWN the GUI will assume the DT is off and will shut down. For the START_SAMPLE command the timeout will be 1 second and there will have to be 20 consecutive time outs before the GUI prompts to the user an error message. These time outs and retries are to ensure proper data transfer and program execution. The numbers are not set in stone and will be optimized during testing.

The next section defines two functions used for data packaging.

2.5.3. Packaging

This section defines the functions to create and disassemble the data packets which are sent by the messages defined above. Both are common to the GUI DLL and the DT. This allows the messages to be sent in bytes and not certain data types thus making the messaging system portable. This portability is needed because we are using two OSs: Linux and Windows.

Packaging
File: Packaging.c

Functions:

Package

Function:	Package	
File:	Packaging.c	**Includes:** -
Description:	This function places data of all types into a char buffer to be sent in a message.	
Inputs:	The buffer to fill and a list of data to fill it.	
Outputs:	The size of the buffer	
Called By:	Shutdown Diagnostic Tool, Take PosiTrack Sample, Sample, Get AP Identification, Send PosiTrack AP Information, Create PosiTrack Athlete, Receive PosiTrack GUI Commands	
Definition:	size_t package(unsigned char *buf, char *format, ...);	

Unpackage

Function:	Unpackage	
File:	Packaging.c	**Includes:** -
Description:	When messages are received this function is called	
Inputs:	The buffer filled with data and a list of data to be filled.	
Called By:	Shutdown Diagnostic Tool, Sample, Get AP Identification, Create PosiTrack Athlete, Receive PosiTrack GUI Commands	
Definition:	void unpackage(unsigned char *buf, char *format, ...);	

2.5.4. Athlete Class

The Athlete class contains the data about the athlete. The data is used for making the GUI-DT connection, and storing appropriate data for localization. The class has a linked list of samples. This list allows of rapid access and maneuverability through the sample list. Below are the class and the definitions of the objects and functions.

Class CAthlete
File: Athlete.cpp

```
class CAthlete {
        public:
                //
                int portNumber;
                char * iPAddress;
                int * connectionHandle
```

```
                int playerID;
                string playerName;
                int playerWeight;
                int playerHeight;

                int numberOfSamples;
                CSample * startPointerSample;
                CSample * currentPointerSample;
                CSample * endPointerSample;

                int newSample (CSample newSample);
                void deleteAllSamples ();
};
```

Variables and Data Structures:

Port Number

Object:	Port Number		
File:	Athlete.cpp	**Includes:**	NA
Class:	CAthlete	**Scope:**	Public
Description:	This is the port that the communication will take place on.		
Used By:	Connect to Athlete Server, Connect PosiTrack Athlete to GUI,		
Definition:	int portNumber;		

IP Address

Object:	IP Address		
File:	Athlete.cpp	**Includes:**	NA
Class:	CAthlete	**Scope:**	Public
Description:	The IP address of the DT (the GumStix)		
Used By:	Connect to Athlete Server,		
Definition:	char * iPAddress;		

Connection Handle

Object:	Connection Handle		
File:	Athlete.cpp	**Includes:**	NA
Class:	CAthlete	**Scope:**	Public
Description:	The handle of the socket that the communication is on between the GUI and the DT.		
Used By:	Send Message, Receive Message		
Definition:	int * connectionHandle		

Player ID

Object:	Player ID		
File:	Athlete.cpp	**Includes:**	NA
Class:	CAthlete	**Scope:**	Public
Description:	Also referred to as the Athlete ID		
Definition:	int playerID;		

Player Name

Object:	Player Name		
File:	Athlete.cpp	**Includes:**	NA
Class:	CAthlete	**Scope:**	Public
Description:	The name of the athlete. This is used by the GUI when presenting the data.		
Definition:	string playerName;		

Player Weight

Object:	Player Weight		
File:	Athlete.cpp	**Includes:**	NA
Class:	CAthlete	**Scope:**	Public
Description:	This piece of information can be used along side the IMU measurement to produce force data.		
Definition:	int playerWeight;		

Player Height

Object:	Player Height		
File:	Athlete.cpp	**Includes:**	NA
Class:	CAthlete	**Scope:**	Public
Description:	This piece of information is to be used in junction with the location of the APs (in height) to calculate the location of the DT.		
Definition:	int playerHeight;		

Number of Samples

Object:	Number of Samples		
File:	Athlete.cpp	**Includes:**	NA
Class:	CAthlete	**Scope:**	Public
Description:	Describes the number of samples in the linked list.		

Definition:	int numberOfSamples;

Start Pointer Sample

Object:	Start Pointer Sample		
File:	Athlete.cpp	**Includes:**	NA
Class:	CAthlete	**Scope:**	Public
Description:	The starting pointer of the linked sample list.		
Definition:	CSample * startPointerSample;		

Current Pointer Sample

Object:	Current Pointer Sample		
File:	Athlete.cpp	**Includes:**	NA
Class:	CAthlete	**Scope:**	Public
Description:	The current pointer to the linked sample list.		
Definition:	CSample * currentPointerSample;		

End Pointer Sample

Object:	End Pointer Sample		
File:	Athlete.cpp	**Includes:**	NA
Class:	CAthlete	**Scope:**	Public
Description:	The end pointer of the linked sample list.		
Definition:	CSample * endPointerSample;		

Functions:

New Sample

Function:	New Sample		
File:	Athlete.cpp	**Includes:**	-
Class:	CAthlete	**Scope:**	Public
Description:	This function adds a new sample to the end of the linked list.		
Inputs:	The new sample		
Outputs:	Number of samples		
Definition:	int newSample (CSample newSample);		

Delete All Samples

Function:	Delete All Samples		
File:	Athlete.cpp	**Includes:**	
Class:	CAthlete	**Scope:**	Public

Description:	This function deletes all the samples in the linked list.
Definition:	void deleteAllSamples ();

2.5.5. Sample Class

This class defines the samples that are measured and the data space that is used to store the calculated data such as location and speed. Listed below are the definitions of the components of the Sample class.

Class CSample
File: Sample.cpp

```
class CSample {
        public:
                //pointer to next sample
                CSample *nextSample;
                //pointer to previous sample
                CSample *previousSample;

                // The amount of time taken to sample
                double duration;
                // Start of sample
                time_t startTime;
                // End of sample
                time_t endTime;
                std::string dateStamp;

                // Measured Data
                struct signal *sig;
                int numberOfSignals;
                struct acceleration accel;
                struct heartRate heartRate;

                // Calculated Data
                struct velocity velo;
                struct location loca;
};
```

Variables and Data Structures:

Next Sample

Object:	Next Sample		
File:	Sample.cpp	**Includes:**	NA

Class:	CSample	**Scope:**	Public
Description:	The pointer to the next sample in the linked list.		
Definition:	CSample *nextSample;		

Previous Sample

Object:	Previous Sample		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public
Description:	The pointer to the previous sample in the linked list.		
Definition:	CSample *previousSample;		

Duration

Object:	Duration		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public
Description:	The length of time required to generate a sample.		
Definition:	double duration;		

Start Time

Object:	Start Time		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public
Description:	The time just before a sample is taken.		
Definition:	time_t startTime;		

End Time

Object:	End Time		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public
Description:	The time just after a sample is taken.		
Definition:	time_t endTime;		

Date Stamp

Object:	Date Stamp		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public
Description:	The human time at when the sample was taken.		
Definition:	std::string dateStamp;		

Signal

Object:	Signal		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public
Description:	A structure that holds the AP's ID, Signal Strength and time when the RSS measurement was made.		
Definition:	struct signal *sig;		

```
struct signal {
        int strength;          // signal Strength of the sampled AP
        int ssid;              // ID of the sampled AP
        time_t timeSS;         // Time when the signal was sampled
};
```

Number of Signals

Object:	Number of Signals		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public
Description:	Defines how many access points are sensed to make a sample (3)		
Definition:	int numberOfSignals;		

Acceleration

Object:	Acceleration		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public
Description:	This structure contains the acceleration data made be the IMU.		
Definition:	struct acceleration accel;		

```
struct acceleration {
        // time when sampled linear acceleration
        struct timeDimension linTime;
        // time when sampled rotational acceleration
        struct timeDimension rotTime;
        // Values of rotational acceleration
        struct dimension rot;
        // Values of rotational acceleration error
        struct dimension rotError;
        // Values of linear acceleration
```



```
        struct dimension lin;
        // Values of linear acceleration error
        struct dimension linError;
};
```

Velocity

Object:	Velocity		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public
Description:	Stores the velocity of the DT.		
Definition:	struct velocity velo;		

```
struct velocity
{
        struct dimension rot;           // Values of rotational velocity
        struct dimension rotError;      // Values of rotational velocity
        struct dimension lin;           // Values of linear velocity
        struct dimension linError;      // Values of linear velocity
        int linearSpeed;                // Magnitude of velocity
        int linearSpeedError;           // Magnitude of Error in velocity
        int rotationalSpeed;            // Magnitude of velocity
        int rotationalSpeedError;       // Magnitude of Error in velocity
};
```

Location

Object:	Location		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public
Description:	This structure hold the location data.		
Definition:	struct location loca;		

```
struct location {
        struct dimension location;          // Cartesian coordinate location
        struct dimension locationError;  // Cartesian coordinate error
};
```

Time Dimension

Object:	Time Dimension		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public

Description:	This structure defines time in three coordinates X, Y, Z.
Definition:	See below

```
struct timeDimension {
        time_t X;                   // Sampled time for X component
        time_t Y;                   // Sampled time for Y component
        time_t Z;                   // Sampled time for Z component
};
```

Dimension

Object:	Dimension		
File:	Sample.cpp	**Includes:**	NA
Class:	CSample	**Scope:**	Public
Description:	This structure defines data in Cartesian coordinates.		
Definition:	See below		

```
struct  dimension {
        double X;           // Values of X component
        double Y;           // Values of Y component
        double Z;           // Values of Z component
};
```


3. Sub-Component Design

This section contains the design specifications for each sub-component in the PosiTracker system: The Diagnostic Tool, The Access Points, and The Graphical User Interface. The functionality with respect to the system will be given followed by a detailed description of the implementation for each component.

3.1. Diagnostic Tool

The main functionality of the Diagnostic Tool is to provide the GUI with data such as the acceleration, location, and speed of each athlete. This data will be sent over a wireless link between the player and the GUI. The data will be obtained through monitoring the RSSI of each AP and the data provided by the IMU.

3.1.1. Gumstix Overo Fire COM

At the heart of the Diagnostic tool will be the Gumstix Overo Fire Computer on Module (COM). This is a small module containing an OMAP 3530 applications processor, a Wi2Wi wireless chip, an integrated power management chip, and an SD card holder. The OMAP processor will run the diagnostic tool software which will collect the wireless data and A/D data. Figure 5 below shows the Overo Fire with each component outlined. The dimensions of the COM are 58mm x 17mm x 4.2mm (approximately the size of a gumstick) and it weighs 5g.

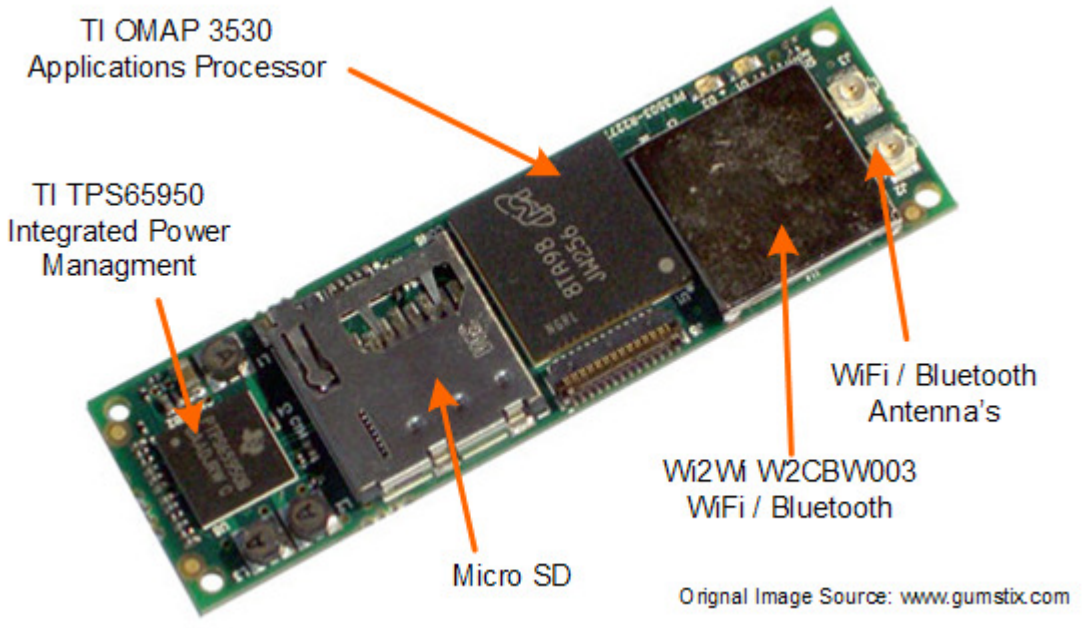


Figure 5: The Gumstix Overo Fire Computer on Module (COM). [11]

The Gumstix Overo Fire COM was chosen for the following reasons:

1. **It is intended for embedded applications as it is low power, light weight, and has extremely small dimensions.** The actually weight of module is 5g and its dimensions are 58mm x 17mm x 4.2mm which easily meets the requirements for both a final product and prototype. The power consumption of the module with WiFi turned on and transmitting continuously is 390mA [12] [13]. Using a standard Iphone 3G battery of 1150mAh the module would run for 2.9hours. This meets the needs of both a final product and a prototype as this is an absolute worst case scenario. A final product could also increase the battery life by using a slightly larger battery.
2. **It is extremely powerful and performance will not limit our possibilities when developing our proof-of-concept device.** The processor used on the module is extremely powerful and is commonly used in cell phones. This means it has more computing power than ever needed for this application. This is very important for two reasons: the system should not be limited by what the hardware can do, and the system should not require the GUI to do all of the calculations. The latter is more of an issue in the final product as multiple athletes will be sending data to the GUI.
3. **It is powerful enough to run Linux.** As a result of the module being so powerful, it is able to run the Linux kernel. This is very important as it allows access to essentially endless open source software and device drivers. This is important as it will reduce development time dramatically through using pre-existing device drivers for WiFi, BlueTooth(final product only), and the A/D converters. This is an excellent advantage as device drivers are time consuming to write. The module is ready to go as it comes preflashed with Linux 2.6.31 or higher.
4. **It has built in WiFi, BlueTooth, and A/D converters built in.** This is important as the hardware for each of these is needed to collect the data from each athlete. The WiFi is responsible for obtaining signal strength and also creating a wireless link to the GUI. The A/D converters are required to obtain the acceleration and rotational data from the IMU. The BlueTooth is not used in the prototype but in the final product will be used to communicate with a heart rate monitor for each athlete.
5. **There is an established Gumstix development community and it is a proven product for embedded applications.** Gumstix is a trusted company and has an entire community developing various embedded applications with the Gumstix product line. The PosiTrack team hopes to leverage the experience of this community when developing both our product and prototype.
6. **PosiTrack team has direct access to an individual with Gumstix development experience.** One of our group members has direct access to someone who has

experience with Gumstix development. The team will take advantage of this relationship to dramatically reduce our development time.

7. **There are various expansion boards that allow for debugging and pin breakout.** These boards will allow us to easily debug through USB, serial, or Ethernet. They will also allow for the IMU and battery power to be easily connected through a 40-pin expansion header. This board is very large compared to the Gumstix and will be used in the prototype only. For the final product a custom breakout board will be created to connect power and the IMU. The expansion board is discussed in more detail in the next section.
8. **It is highly versatile and reusable.** This was important initially in order to secure funding from the Engineering Student Society Endowment Fund. It also allows for easy hardware changes because the module is so versatile. For example in the final product a GPS could easily be attached to the I2C or SPI ports with very little changes.
9. **It could be used in the final product.** Depending on the price point of our final product this module could definitely be used. A competitor, VX Sport, has priced their modules between $1300 and $1500. The final product would be competitive with this retail price [14].

As an alternative to the Over Fire COM, the team considered using various microcontrollers but they are not as powerful and compact (with respect to breakout boards) as the Overo. Microcontrollers do however consume less power, but as shown above, the power consumption of the Overo will meet our requirements. The final drawback is that microcontrollers do cost substantially less than the Overo, but the final product would still be priced competitively to similar products in the market.

Key features of the Over Fire COM:

- OMAP 3530 Applications Processor with ARM Cortex-A8 CPU
- Draws ~250mA @ 4V (1W) with WiFi and BlueTooth disabled [12]
- Draws ~390mA @ 4V (1.6W) with WiFi transmitting continuously [13]
- 600 MHz, 256MB Ram, 256MB Flash
- I2C, SPI, UART
- Six 10bit Analog to Digital lines (located on TPS65950)
- 802.11(g) and Bluetooth®
- Dimensions 58mm x 17mm x 4.2mm
- Weighs 5g including micro SD card
- Runs Linux Kernel, comes with WiFi / BlueTooth drivers
- Built in power management, and micro SD card slot
- Various expansion boards available for debugging and pin breakout
- Preflashed with Linux 2.6.31 or higher

3.1.2. Gumstix Tobi Expansion Board

The Gumstix Tobi expansion board must be attached to the Overo Fire in order for debugging and access to the 40-pin expansion header. This will allow for direct development /debugging of software on the Gumstix using the USB, UART, or Ethernet ports. It will also allow for the battery and the IMU to be easily connected via the 40-pin expansion header. The Tobi expansion board can be seen in Figure 6 below with the relevant parts outlined. The dimensions of the board are 105mm x 40mm x 12mm and it weighs 28g.

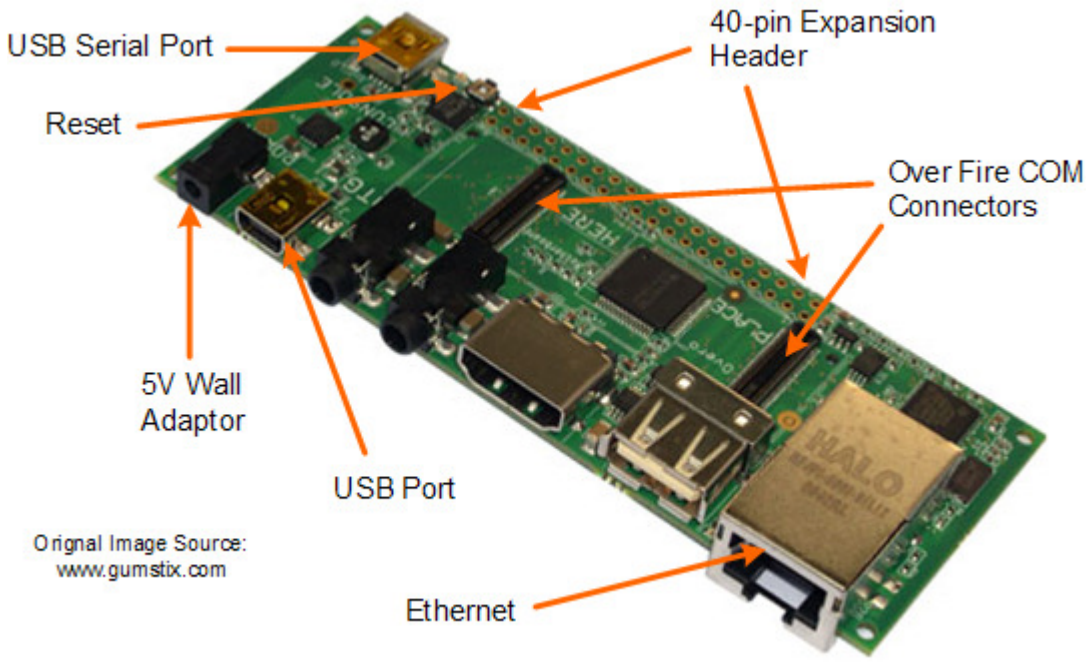


Figure 6: Gumstix Tobi Expansion Board. [15]

This expansion was the chosen board over some of the smaller and more condensed versions because of the extra debugging ability. In particular we wanted Ethernet so that we could mount network drives and use SSH (Secure Shell). The first allows us to easily develop and compile our code on another machine and the second allows us to remotely run shells on the Gumstix. This can be done over the WiFi link but since some work is going to be done possibly modifying the Wireless drivers we also need a physical link.

For the final product none of these debug connections are needed and a custom expansion board could be dramatically smaller. Since the design time is so short a custom expansion board will not be designed. The Tobi board is still a reasonable size to create a working prototype, but would not be used in the final product.

Key features of Gumstix Tobi Expansion board:

- Debug ability through the USB,UART, and Ethernet ports
- 40-pin expansion header that allows battery and IMU to be connected
- TI TPS62111 integrated power management chip
- 40-pin expansion header also allows access to I2C and SPI ports
- Dimensions are 104mm x 40mm x 12mm
- Weighs 28g, and 33g with Overo Fire COM
- Board schematics are available for download from the Gumstix website [16]
- It also has 2 audio jacks and an HMDI jack that will not be used

The table below gives a detailed pin description of the 40-pin expansion header and most of these signals connect directly to the OMAP 3530 processor on the COM. The signal V_BATT connects to the TPS62111 power management chip and will be used to connect the battery. The signals VCC_3.3 and VCC_1.8 are outputs of the power management chip. The signals ADCX and AGND are for the A/D lines, they connect to the TPS65950 power management chip on the Overo Fire COM.

Table 8: 40-pin Expansion Header Pin Descriptions. [17]

Signal	Pin	Pin	Signal
V_BATT	40	39	**ADCIN4**
ADCIN3	38	37	**AGND**
ADCIN5	36	35	**ADCIN6**
ADCIN2	34	33	**ADCIN7**
PWM1	32	31	PWM0
GPIO144_PWM9	30	29	GPIO147_PWM8
GPIO145_PWM10	28	27	GPIO146_PWM11
VCC_1.8	**26**	**25**	**GND**
GPIO185_SDA3	24	23	GPIO184_SCL3
GPIO166_IR_TXD3	22	21	GPIO165_IR_RXD3
GPIO163_IR_CTS3	20	19	GPIO170_HDQ_1WIRE
GPIO127_TS_IRQ	18	17	GPIO128_GPS_PPS
VCC_1.8	**16**	**15**	**GND**
POWERON	14	13	GPIO0_WAKEUP
VBACKUP	12	11	SYS_EN
GPIO148_TXD1	10	9	GPIO151_RXD1
GPIO175_SPI1_CS1	8	7	GPIO173_SPI1_MISO
GPIO174_SPI1_CS0	6	5	GPIO172_SPI1_MOSI
GPIO114_SPI1_NIRQ	4	3	GPIO171_SPI1_CLK
VCC_3.3	**2**	**1**	**GND**

3.1.3. Sparkfun 6-DOF IMU

The Sparkfun 6 Degrees of Freedom (DOF) Inertial Measurement Unit (IMU) provides the Overo Fire COM with analog acceleration and gyroscope data. It has 6-DOF because it provides 3-axis (x,y,z) acceleration and gyroscope data. There are 6 wires that will connect this board to the 40-pin expansion header on the Tobi expansion board. The 40-pin header then connects the analog lines to the COM's integrated power management unit (TPS69650) which contains the 6 A/D converters. The power management unit sends the data back to the OMAP processor through the processors GPIO ports. The Sparkfun board contains a 3-axis accelerometer (ADXL335), a 2-axis gyroscope (LPR530AL) and a 1-axis gyroscope (LY530ALH). The Sparkfun board can be seen in Figure 7 below with the relevant parts outlined. The dimensions of the IMU board are 35mm x 17mm x 3mm and it weighs 2g.

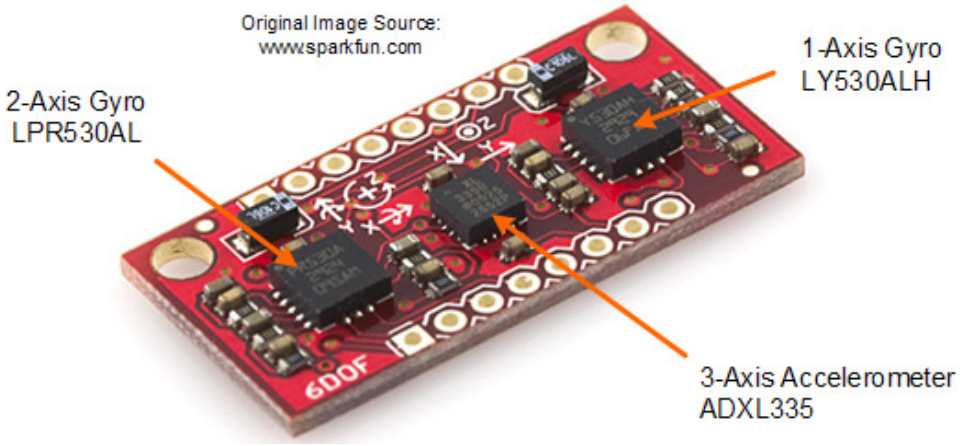


Figure 7: Sparkfun 6-DOF IMU. [18]

The purpose of the IMU is to provide acceleration and gyroscope data in order to dead reckon the athletes location. Dead reckoning is simply using current direction and speed to determine future location. Keeping this in mind, the Sparkfun IMU was chosen based on reasonable sensitivity, working range, and cost. The main specifications are outlined in the table below.

Table 9: Sparkfun 6-DOF IMU Specifications. [19] [20] [21]

Signal	Sensitivity	Supply	Zero Bias	Current	Range	Bandwidth
3-Axis Accel ADXL335	300 mV/g	1.8 - 3.6V	1.5V	0.35mA	±3G	1600Hz (x-y) 550Hz (z)
2-Axis Gyro LY530ALH	0.83 mV/°S 3.33 mV/°S	1.8 - 3.6V	1.23V	6.8mA	±300 °/s	140Hz
1-Axis LPR530AL	0.83 mV/°S 3.33 mV/°S	1.8 - 3.6V	1.23V	5mA	±300 °/s	140Hz

The Sparkfun IMU was chosen mainly to reduce development time as the breakout board is already designed and pre-assembled. The sensitivities are also appropriate for dead reckoning but the range on the accelerometer is too small to be useful in large impacts. This range can be increased but only at the cost of sensitivity at low accelerations. Since dead reckoning is more important to our project, optimizing the accelerometer for it instead of impact detection was chosen. For the final product a custom breakout board with the IMU directly on it can replace the Tobi board. On the final product another accelerometer with a higher working range can also be added to the SPI interface for impact detection.

The voltage provided by Li-Ion batteries is approximately 3.7-3.9V, which is too high to power the IMU. However, the 3.3V supply on the 40-pin expansion header can be used; it is provided by the Tobi onboard power management (TPS62111). The typical power consumption of the IMU is 13mA.

3.1.4. Li-Ion Battery and Charger

To power the Overo Fire COM, Tobi Board, and Sparkfun IMU, the design will use a 3.7V 1000mAh Li-Ion battery. In order to charge the battery the design uses the Maxim1555 Li-Ion Battery Charger. This chip is needed in order to prevent the over charging of the batteries. The battery and the Maxim1555 Charge Board can be seen in Figure 8 below. The dimensions of the battery are 53mm x 33mm x 5.7mm and it weighs 21g. The dimensions of the board are 37mm x 37mm x 6mm and weighs 5g.

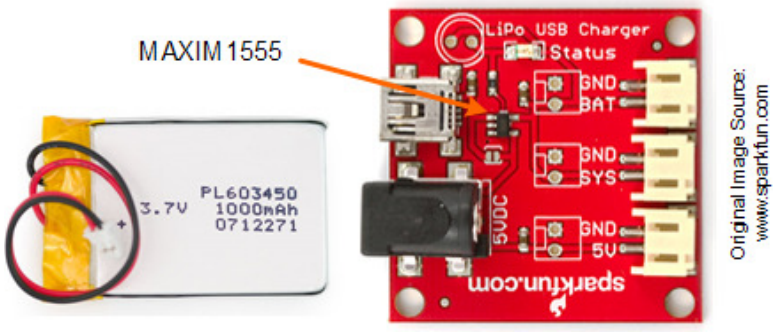


Figure 8: Li-Ion Battery and Maxim1555 Charge Board. [22] [23]

The Maxim1555 has a maximum charge current of 280mA when a DC wall adapter is used. This means the charge board will take 3.6hours to charge the 1000mAh battery. The total current draw of the entire module with the WiFi transmitting continuously and the IMU attached is 403mA. This means that the battery life for the 1000mAh battery will be approximately 2.5hours. The charge time and battery life are not ideal for the final product but are more than satisfactory for the proof-of-concept device. The charge time can be improved by using a higher current charging chip and the battery life can be extended by using a larger battery. Also, in the final product the charging chip will be

included directly onto the custom breakout board (Tobi replacement) which will be attached to the Overo Fire.

3.1.5. Diagnostic Tool Schematic

The schematic showing the connections between the Overo Fire COM, the Tobi expansion board, the IMU, the Li-Ion battery, and the Li-Ion charger are shown in Figure 9 below. More detailed schematics can be found on the Gumstix and Sparkfun website and are listed in the reference section [16] [24] [25].The schematic for the Overo COM is not included as it is proprietary information; however, a detailed description of the signals can be found in the reference section [26].

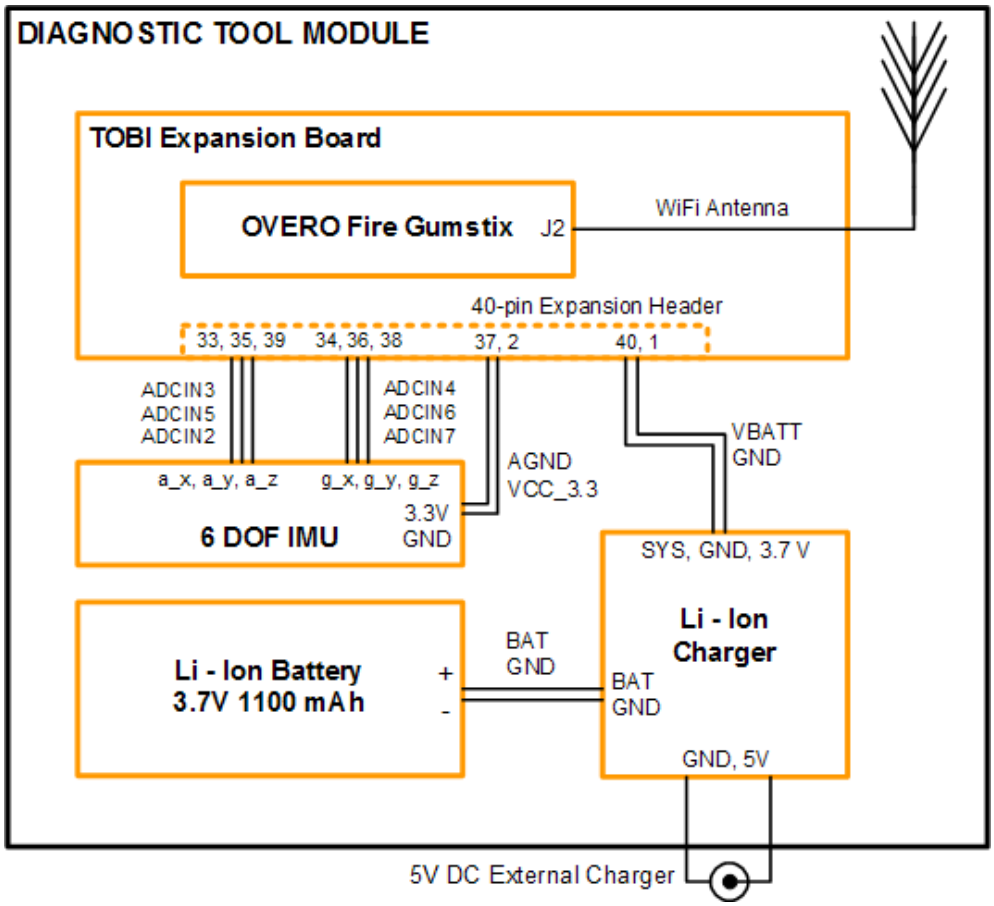


Figure 9: Diagnostic Tool Schematic.

The ADCIN(2,3,4,5,6,7) signals from the IMU are connected to the 40-pin expansion header which then connects them to the Overo Fire COM. On the Overo they are connected to the power management chip (TPS65950) containing the A/D converters. The IMU's 3.3V supply voltage is created by the Tobi expansions boards' power management chip (TPS62111). The 40-pin header is defined in Table 10. The Charger signals VBATT and GND connect to the power management chip on the Tobi board.

3.1.6. Diagnostic Tool-Casing and Strap

This section will discuss the DTs casing for both a proof-of-concept device and a final product. A proposed strap is also introduced that could be used in the final product in order to attach the DT onto the athlete. Table 1 below summarizes the physical characteristics of each sub-component in the DT and also includes a potential casing for the proof-of-concept device. The final entry in the table contains the weight and dimensions of the prototype using this casing. This casing is a small case that Gumstix provided with the Tobi expansion board and it can easily be modified to be used in the proof-of-concept device.

Table 10: Summary of the relevant physical information of the DTs sub-components

Component	Weight	Dimensions
Overo Fire COM	5g	58mm x 17mm x 4.2mm
Tobi Expansion Board	28g	105mm x 40mm x 12mm
Sparkfun 6-DOF IMU	2g	35mm x 17mm x 3mm
Maxim1555 Charge Board	5g	37mm x 37mm x 6mm
3.7V 1000mAh Li-Ion Battery	21g	53mm x 33mm x 5.7mm
WiFi Antenna	6g	Cylinder: R = 4mm , L = 80mm
Gumstix Tobi Casing	38g	110mm x 51mm x 21mm
Diagnostic Tool Prototype	105g	110mm x 51mm x 21mm

Using the Tobi boards casing, the prototype will have a total weight of 105g and dimensions of 110mm x 51mm x 21mm. This includes the Overo Fire, Tobi board, Wi-Fi antenna, battery, charging module, and IMU contained inside. This fits the functional requirements for it being both small and lightweight so it will be unobtrusive the athlete wearing it. Figure 10 below shows a Solidworks CAD drawing of the all of the components placed inside of the prototypes casing. This figure allows for a visualization of how each component will be placed inside of the casing. Figure 11 below shows how the final product or prototype could be potentially attached to the athlete. In this figure the DT is clipped into a holster located on the athletes protective gear or clothing. Alternatively, the DT could be attached to the player with straps similar to a backpack.

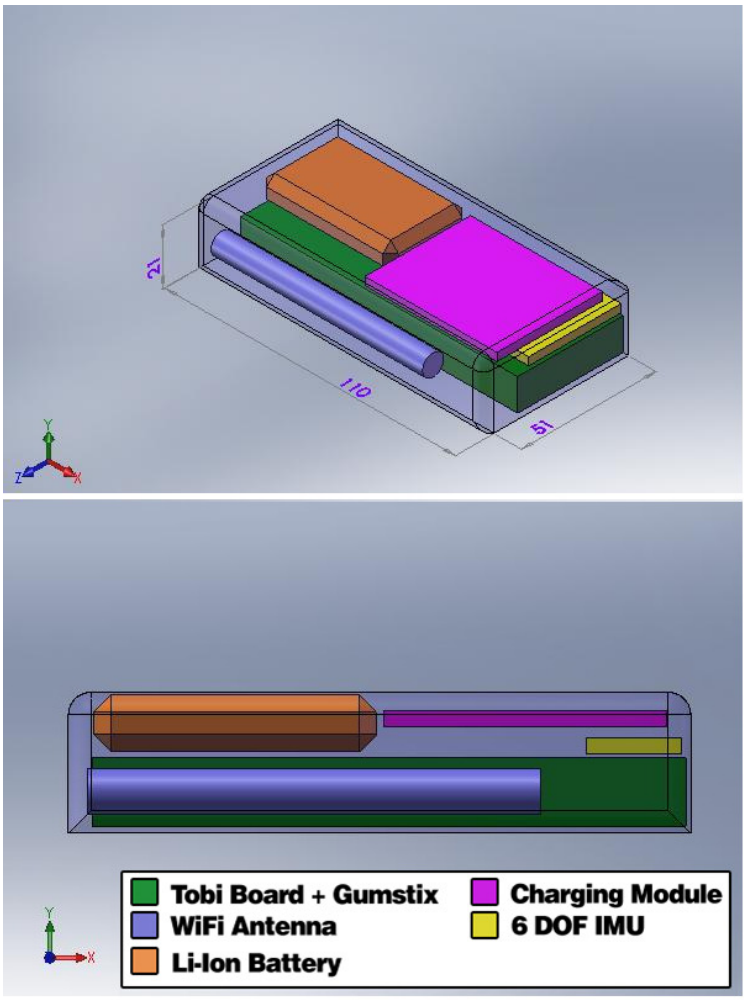


Figure 10: Solid works drawing of the entire Diagnostic Tool Module.

Figure 11: Hockey armor with Diagnostic Tool attached to the back.

3.1.7. Diagnostic Tool Software

The figure below outlines the start up software algorithm to be implemented on each athlete's Diagnostic Tool.

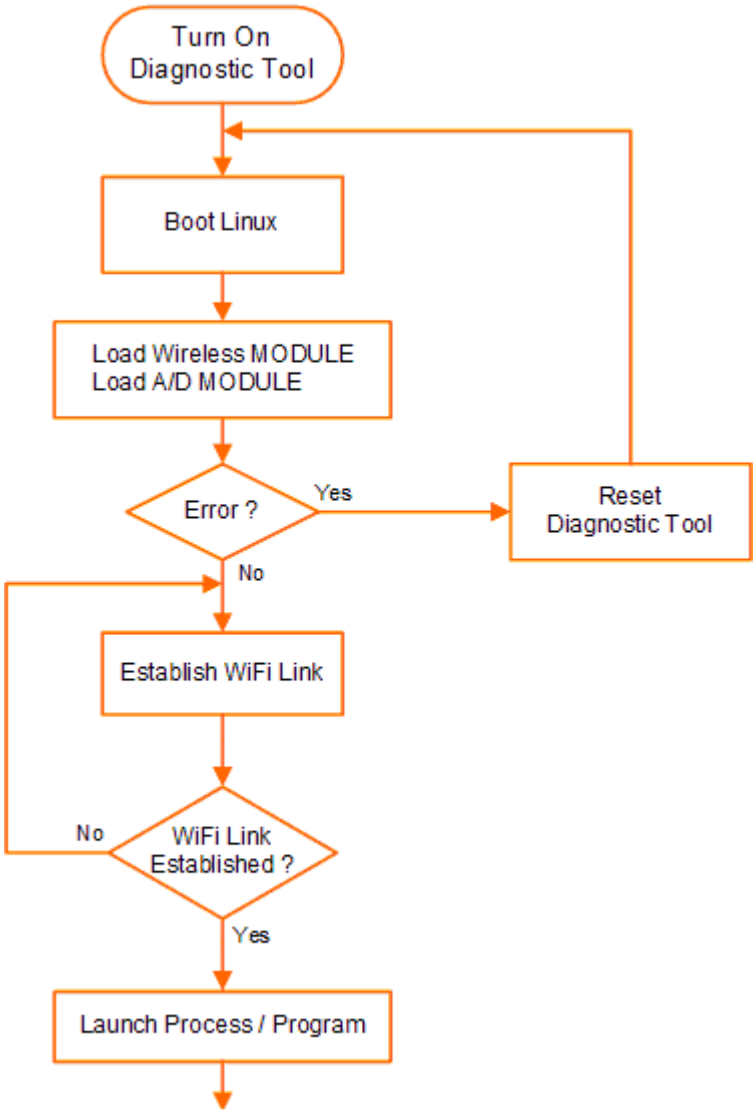


Figure 12: Diagnostic Tool Software Start Up Algorithm.

When the diagnostic tool is turned on, or given power, it will boot Linux from its flash memory. The boot script will be modified to allow for the Wireless module and A/D module to be loaded automatically into the kernel. These modules are device drivers that will enable access to the Wireless chipset and the IMU. Once Linux has loaded, a simple script will execute through the boot log to ensure that the modules have loaded properly, if not the tool will reboot. If there are no errors, the script will create a Wi-Fi connection

to the created WLAN that consists of one of the AP. After the connection is established the main program is loaded.

3.1.7.1 PosiTrack Athlete Executable

The executable that is run after startup in figure 12 is defined here. The functions that this executable calls are listed below.

PosiTrack Athlete
File: PosiTrackAthlete.cpp

Functions:

Main Program

Function:	Main		
File:	PosiTrackAthlete.cpp	**Includes:**	-
Description:	This is the application entry point for the DT PosiTrack Athlete program.		
Calls:	Start Up PosiTrack Athlete, Connect PosiTrack Athlete to GUI, Receive PosiTrack GUI Commands		
Definition:	int main();		

Start Up PosiTrack Athlete

Function:	Start Up PosiTrack Athlete		
File:	PosiTrackAthlete.cpp	**Includes:**	-
Description:	This function is run at start up to initialize variables and to facilitate start up.		
Called By:	Main		
Definition:	void startUpPosiTrackAthlete( );		

Connect PosiTrack Athlete to GUI

Function:	Connect PosiTrack Athlete to GUI		
File:	PosiTrackAthlete.cpp	**Includes:**	-
Description:	This function facilitates the connection procedure when connecting to the GUI.		
Outputs:	Connection handle to the socket		
Calls:	Connect To GUI Client		
Called By:	Main		
Definition:	int * connectPosiTrackAthleteToGUI ( );		

Receive PosiTrack GUI Commands

Function:	Receive PosiTrack GUI Commands		
File:	PosiTrackAthlete.cpp	**Includes:**	-
Description:	The purpose of this function is to receive the messages and act accordingly to the command.		
Calls:	Receive Message (Athlete), Unpackage, Create PosiTrack Athlete, Package, Send Message (Athlete), Send PosiTrack AP Information, Take PosiTrack Sample, Delete PosiTrack Athlete		
Called By:	Main		
Definition:	void receivePosiTrackGUICommands ( );		

Create PosiTrack Athlete

Function:	Create PosiTrack Athlete		
File:	PosiTrackAthlete.cpp	**Includes:**	-
Description:	The purpose of this function is to initialize and fill the Athlete class.		
Outputs:	A pointer to the Athlete class		
Called By:	Receive PosiTrack GUI Commands		
Definition:	CAthlete *createPosiTrackAthlete (int playerID, string playerName, int playerWeight, int playerHeight , int * connectionHandle);		

Delete PosiTrack Athlete

Function:	Delete PosiTrack Athlete		
File:	PosiTrackAthlete.cpp	**Includes:**	-
Description:	This function deallocates memory that is taken by the Athlete class.		
Inputs:	A pointer to the Athlete class		
Called By:	Receive PosiTrack GUI Commands		
Definition:	void deletePosiTrackAthlete (CAthlete * athlete);		

Send PosiTrack AP Information

Function:	Send PosiTrack AP Info		
File:	PosiTrackAthlete.cpp	**Includes:**	-
Description:	This function facilitates the sending of the AP IDs that are sensed by the DT.		
Calls:	Get AP Signal Strength, Parse AP Signal Strength, Package, Send Message (Athlete),		
Called By:	Receive PosiTrack GUI Commands		
Definition:	void sendPosiTrackAPInfo (CAthlete * athlete);		

Send PosiTrack Samples

Function:	Send PosiTrack Samples		
File:	PosiTrackAthlete.cpp	**Includes:**	-

Description:	This function performs the task of setting up the message to send and sending the message.
Inputs:	A pointer to the Athlete class, the Sample class and the number of sample to send in the packet.
Outputs:	A pointer to the last sample sent on the link list.
Calls:	Package, Send Message (Athlete),
Called By:	Take PosiTrack Sample
Definition:	CSample * sendPosiTrackSample (CAthlete * athlete, CSample * currentSample, int numberOfSamplesToSend);

Take PosiTrack Sample

Function:	Take PosiTrack Sample		
File:	PosiTrackAthlete.cpp	**Includes:**	-
Description:	This function facilitates the sampling of desired measurements and sending the samples to the GUI. This function is also responsible for memory management.		
Inputs:	A pointer to the Athlete class and the Sample class.		
Calls:	Get AP Signal Strength, Parse AP Signal Strength, Get Acceleration, Parse Acceleration, New Sample, Send PosiTrack Samples, Write PosiTrack Athletes to File, Delete All Samples		
Called By:	Receive PosiTrack GUI Commands		
Definition:	void takePosiTrackAthlete (CAthlete * athlete, CSample * currentSample);		

Parse AP Signal Strength

Function:	Parse AP Signal Strengths		
File:	PosiTrackAthlete.cpp	**Includes:**	-
Description:	This function takes the measure AP signal strength and converts the format and stores data into the appropriate location.		
Inputs:	A buffer that contains the APs signal strength IDs and values. A pointer to the sample that needs to be filled with the buffer data.		
Called By:	Take PosiTrack Sample, Send PosiTrack AP Info		
Definition:	void parseAPSignalStrengths (char * buffer, CSample * currentSample);		

Parse Acceleration

Function:	Parse AP Signal Strengths		
File:	PosiTrackAthlete.cpp	**Includes:**	-
Description:	This function takes the measure accelerometer data and converts the format and stores data into the appropriate location.		
Inputs:	A buffer that contains the A pointer to the sample that needs to be filled with the buffer data.		
Called By:	Take PosiTrack Sample,		

Definition:	void parseAcceleration (char * buffer, CSample * currentSample);

Write PosiTrack Athletes to File

Function:	Write PosiTrack Athletes To File		
File:	PosiTrackAthlete.cpp	**Includes:**	<fstream>
Description:	This function is used if the number of samples reach a maximum value the samples can be stored in a certain file.		
Inputs:	A pointer to the athlete and the file name to be written to.		
Outputs:	Returns nothing		
Calls:	Nothing		
Called By:	Take PosiTrack Sample,		
Definition:	void writePosiTrackAthletesToFile (CAthlete * athlete, char *filename);		

Display PosiTrack Samples

Function:	Display PosiTrack Samples		
File:	PosiTrackAthlete.cpp	**Includes:**	<iostream>
Description:	This function takes the class CAthlete and outputs the information that has been put into the class to the terminal. The Athletes information, points to its sample link list, and the list of samples to the terminal. Uses to tests the CAthlete class and give confirmation of storage.		
Inputs:	A pointer to the athlete.		
Called By:	displayPosiTrackAthletes,		
Definition:	void displayPosiTrackSamples (CAthlete * athlete)		

Display PosiTrack Athletes

Function:	Display PosiTrack Athletes		
File:	PosiTrackAthlete.cpp	**Includes:**	<iostream>
Description:	Displays the athlete information along with the samples.		
Inputs:	A pointer to the athlete.		
Definition:	void displayPosiTrackAthletes (CAthlete * athlete)		

3.1.7.2 Measurement Tools

The measurement tools are functions that obtain data that is generated from the Linux kernel modules. The modules include the Wireless module which measures the AP signal strength and A/D module which generated the acceleration measurements.

Signal Strength
File: SignalStrength.c

Functions:

Get AP Signal Strength

Function:	Get AP Signal Strengths		
File:	SignalStrength.c	**Includes:**	-
Description:	This function was used to measure the signal strength from the APs.		
Outputs:	A character buffer that contains the AP IDs and corresponding signal strengths.		
Called By:	Take PosiTrack Sample, Send PosiTrack AP Information		
Definition:	char * getAPSignalStrengths ( );		

Acceleration
File: Acceleration.c

Functions:
Get Acceleration

Function:	Get Acceleration		
File:	Acceleration.c	**Includes:**	-
Description:	This function was used to measure the linear and rotational acceleration of the DT.		
Outputs:	A character buffer that contains the acceleration from all inertial measurement units.		
Called By:	Take PosiTrack Sample		
Definition:	char * getAcceleration( );		

3.1.7.3 Message Passing

This software block is used to generate a connection from the DT to the GUI with TCP socket. The following functions also send and receive messages and facilitate the disconnection.

Server Athlete
File: ServerAthlete.c

Functions:

Connect to GUI Client

Function:	Connect To GUI Client

File:	ServerAthlete.c	**Includes:**	-
Description:	This function initializes a socket and binds the process to a port for the GUI client to connect to. Socket needs to be initialized and bound to enable message passing over a network.		
Inputs:	Port number		
Outputs:	Connection handle		
Called By:	Connect PosiTrack Athlete to GUI		
Definition:	int connectToGUIClient(int argc);		

Receive Message (Athlete)

Function:	Receive Message		
File:	ServerAthlete.c	**Includes:**	-
Description:	Receives a message on the socket which is sent by the GUI.		
Inputs:	The socket handle, a pointer to a buffer and the size of the buffer.		
Called By:	Receive PosiTrack GUI Commands		
Definition:	void recieveMessage (int sockethandle, char *buffer, int sizeOfBuffer);		

Send Message (Athlete)

Function:	Send Message		
File:	ServerAthlete.c	**Includes:**	-
Description:	Writes a message to the socket which will be received by the GUI.		
Inputs:	The socket handle, a pointer to a buffer and the size of the buffer.		
Called By:	Send PosiTrack Samples, Receive PosiTrack GUI Commands, Send PosiTrack AP Information		
Definition:	void sendMessage (int newsocketfd,char *buffer, int sizeOfBuffer);		

Disconnect from GUI Client

Function:	Disconnect to GUI Client		
File:	ServerAthlete.c	**Includes:**	-
Description:	Terminates the connection and closes socket. Cleans socket connection so it can be used upon next execution.		
Inputs:	The socket handle		
Called By:	Receive PosiTrack GUI Commands		
Definition:	void disconnectToGUIClient(int newsockfd);		

3.2. Access Points

The AP's main function is to provide an appropriate number of anchors for localization. The access points will send detectable signals to the diagnostic tool to allow distance approximation. The secondary function of the access points is to supply a wireless communication link between the DT and the GUI.

In the proof-of-concept design, third party wireless routers will be used. Application specific Wi-Fi access points could give better performance in a more compact packaging however, this would have been too time consuming for our development schedule and would have placed the project at a much higher budget point.

3.3. Graphical User Interface

This section describes the Graphical User Interface (GUI) design for the PosiTracker, with design considerations being outlined in the respective subsections. The main functionality of the GUI is to display the athlete information in a graphical, visual-appealing format.

3.3.1. Overview of the Graphical User Interface

The OVERO Air Gumstix runs a Linux environment, and development for the Diagnostic Tool is in Linux as well; however, due to the fact that most PC's currently run Windows, it was determined the GUI should be developed for Windows.

This does complicate development, since in developing for two different operating systems, ActiveX controls need to be written in order to be able to access the functions from the DT in the GUI. However, having the GUI running on Windows will ultimately increase the marketability of the product. Coaches could simply install the application that comes with the PosiTracker package on their existing machines, thus minimizing additional costs associated with the acquisition of a machine running Linux, as well as taking the time to get used to a new Operating System.

From past experience developing programs for Windows, Microsoft Visual Studio 2008 suite, Professional Edition, was the platform which was chosen; it is focused on the development of Windows Vista programs. In particular, Microsoft Visual Basic will be used for developing the GUI.

There are many reasons to support the choice for Visual Studio 2008 - among these are the Code Editor, Debugger and Designer features that greatly aid in the development of applications.

3.3.2. Graphical User Interface-Layout

The GUI consists of several Windows Forms that are linked to each other. The program itself consists of three modules, developed independently: the Game Analysis, Physical Training and Tactical Training modules. The user is supposed to turn on the hardware and start the program, and then a series of prompts will take the user to the module of choice. The data flow is designed to be intuitive and the program, easy to use, as shown by the flowchart in Figure 13.

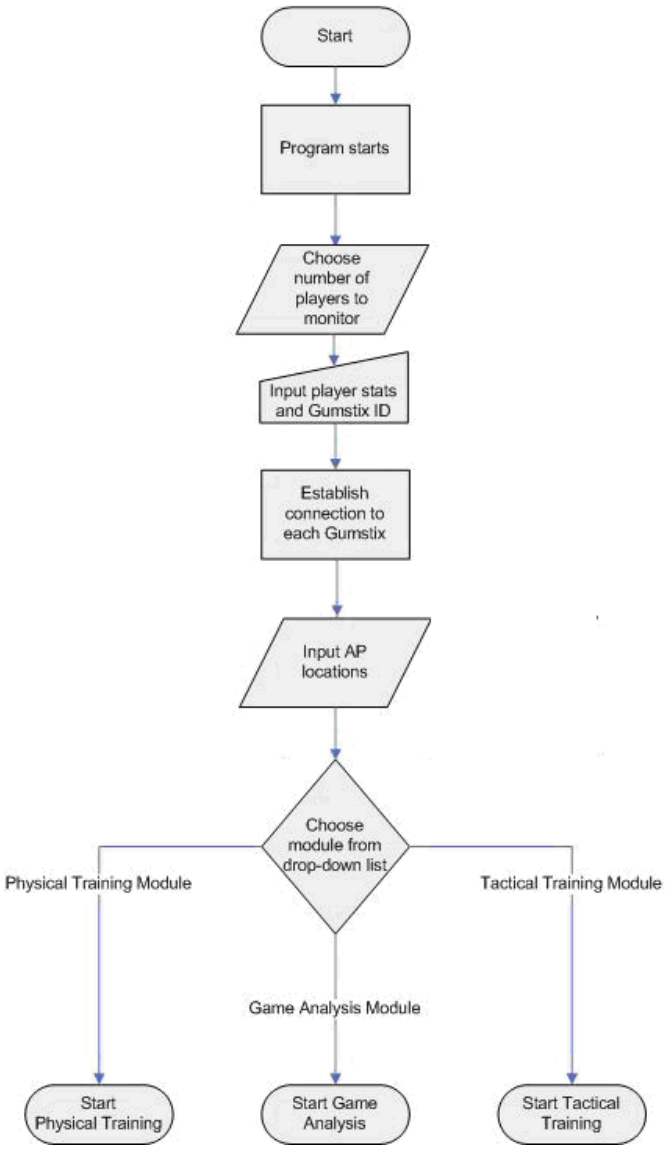
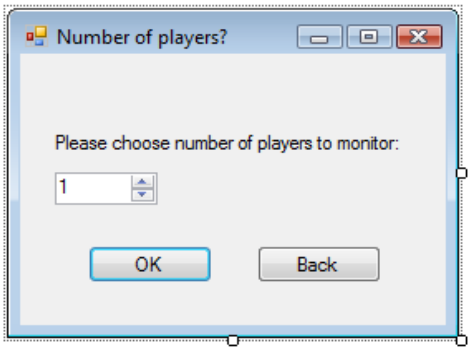


Figure 13: GUI program flow.

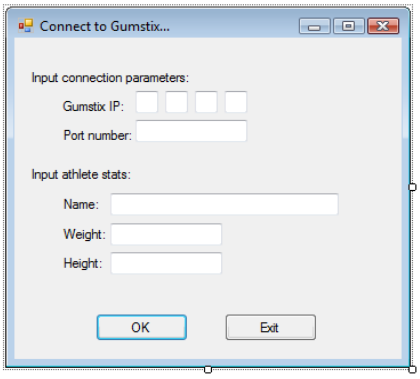
3.3.3. Graphical User Interface-Software Layout Interface

The connection between the Diagnostic Tool and the GUI is done via DLL's. The DLL exported function calls are defined in section 3.3.4. The DLL functions used to allow the Diagnostic Tool to get all required user input, and the GUI to get all the positioning data needed from the DT. The user input will be read through various mechanisms: combo boxes, text boxes or numeric up-down displays. The screenshots below in figure 14 show the intended layout design.

A.



B.



C.

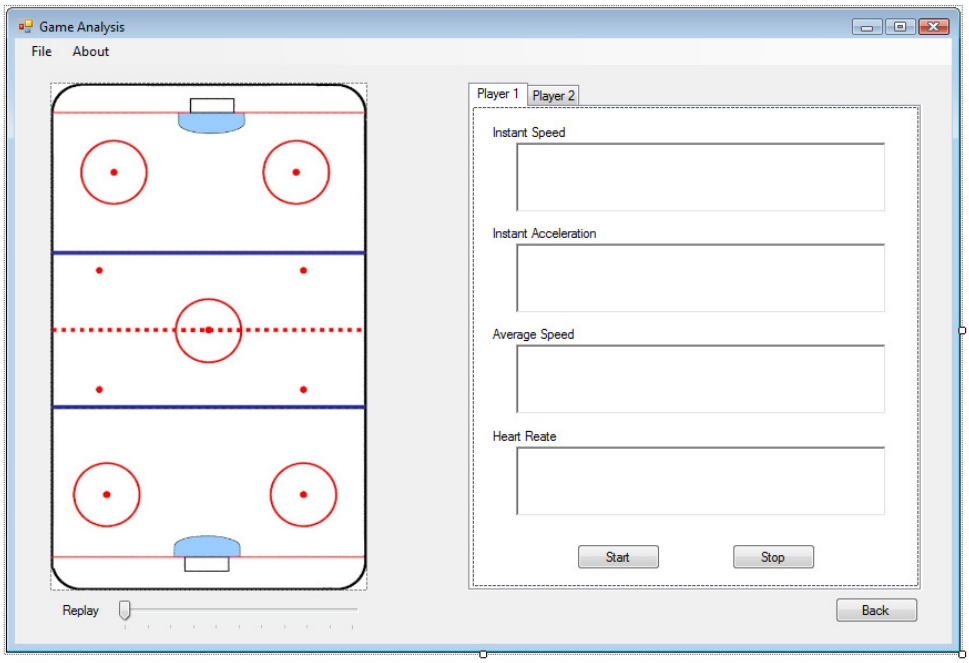


Figure 14: The Draft look of the GUI windows A, B, and C.

3.3.4. Graphical User Interface-Software

The Graphical User Interface C++/C software generates a DLL that the GUI written in VB links too. The exported DLL function calls used by the GUI are described below. The algorithms are discussed and the message passing functions are defined.

3.3.4.1 PosiTrack GUI DLL

This section presents the functions defined in GUI DLL. These functions are called from the VB GUI program

PosiTrack GUI
File: PosiTrackGUI.cpp

Functions:

Create PosiTrack Athlete

Function:	Create PosiTrack Athlete		
File:	PosiTrackGUI.cpp	**Includes:**	-
Description:	This function creates the athlete class in both the GUI DLL and the DT. It does this by first filling the classing the GUI DLL. Then the function connects to the DT with given port and IP address. Once connected the athlete information is sent to the DT. It the function fails it returns a null pointer.		
Inputs:	name, height, weight, port, IP address		
Outputs:	A pointer to the Athlete class		
Calls:	Connect to Athlete Server, Package, Send Message (GUI)		
Called By:	GUI VB program		
Definition:	CAthlete *createPosiTrackAthlete (int playerID, string playerName, int playerWeight, int playerHeight);		

Get AP Identification

Function:	Get AP Identification		
File:	PosiTrackGUI.cpp	**Includes:**	-
Description:	This function is used to receive the available APs measured from the DT.		
Inputs:	A pointer to the Athlete class		
Outputs:	Success(1) or failure(-1)		
Calls:	Package, Send Message (GUI), Receive Message (GUI), Unpackage		
Called By:	GUI VB program		
Definition:	int getAPID(CAthlete * athlete)		

Input AP Information

Function:	In AP Information		
File:	PosiTrackGUI.cpp	**Includes:**	-
Description:	The user inputs the location and transmitted signal strength of the APs.		
Inputs:	The location of each AP and the corresponding access point ID.		
Called By:	GUI VB program		
Definition:	void inputAPInfromation(APData *APInfo, int numberOfAPs)		

Sample

Function:	Sample		
File:	PosiTrackGUI.cpp	**Includes:**	-
Description:	This function calls the DT to take a sample and send it to the GUI.		
Inputs:	The athlete class where the sample will be added to.		
Outputs:	Success(1) or failure(-1)		
Calls:	Package, Send Message (GUI), Receive Message (GUI), Unpackage, New Sample, Append PosiTrack Athletes Samples to File, Write PosiTrack Athletes to File		
Called By:	GUI VB program		
Definition:	int sample (CAthlete * athlete)		

Get Signal Strength

Function:	Get Signal Strength		
File:	PosiTrackGUI.cpp	**Includes:**	-
Description:	The VB GUI gets the signal strength from the DLL		
Inputs:	The athlete class where the RSS was added and the AP ID.		
Outputs:	Signal strength		
Called By:	GUI VB program		
Definition:	int getSignalStrength(CAthlete * athlete, int APID);		

Get Acceleration

Function:	Get Acceleration		
File:	PosiTrackGUI.cpp	**Includes:**	-
Description:	Returns the acceleration to the GUI.		
Inputs:	The athlete class where the acceleration was added. A pointer to the acceleration structure.		
Outputs:	Success(1) or failure(-1)		
Called By:	GUI VB program		
Definition:	int getAcceleration(CAthlete * athlete, acceleration * accel)		

Generate Location

Function:	Generate Location		
File:	PosiTrackGUI.cpp	**Includes:**	-
Description:	Returns the location that has be calculated.		
Inputs:	The athlete class where the location is to be added. A pointer to a location structure.		
Outputs:	Success(1) or failure(-1)		
Calls:	Calculate Location		
Called By:	GUI VB program		
Definition:	int generateLocation(CAthlete * athlete, location* loca;)		

Generate Speed

Function:	Generate Speed		
File:	PosiTrackGUI.cpp	**Includes:**	-
Description:	This function returns the calculated speed of the DT to the GUI		
Called By:	GUI VB program		
Definition:	int generateSpeed(CAthlete * athlete, velocity * velo)		

Shutdown Diagnostic Tool

Function:	Shutdown diagnostic tool		
File:	PosiTrackGUI.cpp	**Includes:**	-
Description:	This function sends a message to the DT asking it to shutdown.		
Called By:	GUI VB program		
Definition:	shutdownDiagnosticTool(CAthlete * athlete)		

Delete Athlete

Function:	Delete Athlete		
File:	PosiTrackGUI.cpp	**Includes:**	-
Description:	This deletes all the athletes in the DLL.		
Called By:	GUI VB program		
Definition:	deleteAthlete(CAthlete * athlete)		

Delete Samples

Function:	Delete Samples		
File:	PosiTrackGUI.cpp	**Includes:**	
Description:	This deletes all the sample stored in the athlete.		
Definition:	void deleteSamples(CAthlete * athlete)		

Write PosiTrack Athletes to File

Function:	Write PosiTrack Athletes To File		
File:	PosiTrackGUI.cpp	**Includes:**	<fstream>
Description:	This function is used if the number of samples reach a maximum value the samples can be stored in a certain file.		
Inputs:	A pointer to the athlete and the file name to be written to.		
Outputs:	Returns nothing		
Calls:	Nothing		
Definition:	void writePosiTrackAthletesToFile (CAthlete * athlete, char *filename);		

Append PosiTrack Athletes Samples to File

Function:	Append PosiTrack Athletes To File		
File:	PosiTrackGUI.cpp	**Includes:**	<fstream>
Description:	Appends the samples to a preexisting file.		
Outputs:	Returns nothing		
Calls:	Nothing		
Definition:	void appendPosiTrackAthletesToFile (CAthlete * athlete, char *filename);		

Display PosiTrack Samples

Function:	Display PosiTrack Samples		
File:	PosiTrackGUI.cpp	**Includes:**	<iostream>
Description:	This function takes the class CAthlete and outputs the information that has been put into the class to the terminal. The Athletes information, points to its sample link list, and the list of samples to the terminal. Uses to tests the CAthlete class and give confirmation of storage.		
Inputs:	A pointer to the athlete.		
Called By:	displayPosiTrackAthletes,		
Definition:	void displayPosiTrackSamples (CAthlete * athlete)		

Display PosiTrack Athletes

Function:	Display PosiTrack Athletes		
File:	PosiTrackGUI.cpp	**Includes:**	<iostream>
Description:	Displays the athlete information along with the samples.		
Inputs:	A pointer to the athlete.		
Definition:	void displayPosiTrackAthletes (CAthlete * athlete)		

3.3.4.2 Algorithms

The functions and

Algorithms
File: Algorithms.cpp

Variables and Data Structures:

AP Data

Object:	AP Data		
File:	Algorithms.cpp	**Includes:**	NA
Class:	-	**Scope:**	global
Description:	This data structure is use to hold the location of the APs, and the transmitted signal strength. This data is used by the algorithm functions to determine the location of the DT.		
Used By:	Calculate AP Radius, Calculate Circle Intersection		
Definition:	struct APData *APInfo		

```
struct  APData{
        char * APID;                        // access point ID
        struct dimension location;          // location of AP
        int transmittedSignalStrength;      // signal Strength at AP
};
```

```
struct  dimension {
        double X;           // Values of X component
        double Y;           // Values of Y component
        double Z;           // Values of Z component
};
```

Functions:

Calculate Location

Function:	Calculate Location		
File:	Algorithms.cpp	**Includes:**	-
Description:	This function uses the algorithms described in [6] [7] [9] [10] to calculate the location of the DT from measured parameters.		
Definition:	calculateLocation( )		

Calculate Velocity

Function:	Calculate Velocity		
File:	Algorithms.cpp	**Includes:**	-
Description:	Uses the location and acceleration values to determine speed of the DT.		
Definition:	calculateVelocity( )		

3.3.4.3 Message Passing

This section is similar to the section 3.1.7.3. but for the GUI.

Client GUI
File: clientGUI.c

Functions:

Connect to Athlete Server

Function:	Connect To Athlete		
File:	clientGUI.c	**Includes:**	
Description:	Connects the GUI to the DT.		
Definition:	int connectToAthleteServer(int argc, char *argv);		

Receive Message (GUI)

Function:	Receive Message		
File:	clientGUI.c	**Includes:**	
Description:	Receive Message from the DT.		
Definition:	void recieveMessage (int newsocketfd, char *buffer, int sizeOfBuffer);		

Send Message (GUI)

Function:	Send Message		
File:	clientGUI.c	**Includes:**	
Description:	Sends a message to the DT.		
Definition:	void sendMessage (int newsocketfd,char *buffer, int sizeOfBuffer);		

Disconnect from Athlete Server

Function:	Disconnect to GUI Client		
File:	clientGUI.c	**Includes:**	
Description:	Disconnects from the DT.		
Definition:	void disconnectFromAthleteServer(int newsockfd);		

4. System Test Plan

4.1. Overview

The individual parts of our system will be tested first; after individual testing is complete we will do integration testing.

The use of this plan will help ensure that the hardware and software development, evaluation and acceptance standards are documented and followed.

For the software side of our project intends to perform the following types of tests:

- System Testing
  - All code will be tested to ensure that the application performs the required functions and outputs the required results.
  - Testing ensures the proper operation of each module before moving on to integrating the modules together.
- Integration Testing
  - Integrate small parts of the application and test after each part has been added.
  - Each part must be fully tested and fixed before integrating it with the rest of the application; this makes it easier to detect problems/bugs as well as making it easier to fix the bugs.

For the hardware side of our project we want to ensure the system operates under regular circumstances. Our environment is quite static, so we do not expect to encounter any extreme situations.

4.2. Access Points Tests

In the case of the access points we plan to test that they emit signals, and that the signal is stable within some limits.

Case 1: Wi-Fi signal strength measurement.

> **Testers Instructions:** The tester will measure the Wi-Fi power at various distances. Do this for each AP.

Conditions: Indoors or outdoors, both with direct line of sight.

Expected Observation: As we are using similar routers as our AP's, the signal strength measurements should be very close in value for the same distance from the measuring device.

Case 2: Wi-Fi signal strength measurement with 2 or more AP's added

Testers Instructions: The tester will measure the Wi-Fi power at various distances when more than one AP is set up.

Conditions: Indoors or outdoors, both with direct line of sight.

Expected Observation: The signal strength is not be affected by the interference of other AP's. The measurements should yield similar values to Case 1.

4.3. Gumstix Tests

The Gumstix is the device to be worn by the players, thus there are a few physical aspects to be tested:

- The casing is water-resistant (the system will not be affected by the player perspiration)
- The straps are comfortable to wear
- The batteries will last for at least 3 hours under intense usage

We will test the batteries by actually using the device for a few hours; we will test the straps by asking a user to wear the Gumstix on their back while playing some kind of sport.

4.4. Diagnostic Tool Tests

For the diagnostic tool we have to ensure the code compiles, as well as ensure the functions work as expected, and output accurate results. We also need to ensure the diagnostic tool is able to handle message passing between Linux and Windows.

Case 1: Connectivity test and message passing test.

Testers Instructions: The tester will send dummy messages between the Gumstix and the GUI.

Conditions: Indoors or outdoors, both with direct line of sight. All device components should be turned on (computer, AP, Gumstix) and a wireless connection should be established between the computer and the Gumstix.

Expected Observation: The tester should be able to send and receive messages commands from the GUI to the Gumstix. This ensures the diagnostic tool is able to handle message passing between Linux and Windows.

Case 2: Function call test.

Testers Instructions: The tester will test several function calls, from Linux and from the GUI.

Conditions: Indoors or outdoors, both with direct line of sight. All device components should be turned on (computer, AP, Gumstix) and a wireless connection should be established between the computer and the Gumstix.

Expected Observation: This test ensures the tester is able to perform tasks such as getting the speed of a particular athlete, or shut down the system, from the GUI.

4.5. Graphical User Interface Tests

With the Graphical User Interface, we want to ensure the following:

1. Is the application easy to learn? What about the usability of our design? Are menus/buttons/displays where the user would expect them to be?
2. Are there any exceptions that haven't been handled? Will anything crash the program?

We will provide answers to the questions above by performing two additional types of tests on the GUI:

- User testing
    - As soon as the GUI is fully developed, a person outside the development group will test the application
    - Provides some feedback on the usability of the system
- Automated Unit Testing
    - Using the tools provided by Microsoft Visual Studio 2008
    - AUT ensures that there are no unhandled exceptions

Below are some of the tests we will perform on our GUI:

Case 1: Installation, start up, menu browsing and shut down.

Testers Instructions: The tester will install the application, then start it, browse through the menus and exit.

Conditions: Indoors or outdoors; need to use a computer running Windows XP or higher.

Expected Observation: The application should install with no errors. The application will start as expected; browsing through any menus will happen with no errors. At shut down, the used memory will be freed, no processes will be left hanging.

Case 2: Adding athletes.

Testers Instructions: The tester will choose a number of athletes to track, then input the athlete information or open an athlete file from the GUI.

Conditions: Indoors or outdoors; need to use a computer running Windows XP or higher.

Expected Observation: The tester should be able to open athlete files or input the athlete information from keyboard, as desired. The athlete information will be available to the Gumstix after all athletes are added.

Case 3: Detecting AP's

Testers Instructions: The tester will choose an athlete, then press the "Detect" button to detect surrounding AP's.

Conditions: Indoors or outdoors; need to use a computer running Windows XP or higher.

Expected Observation: The surrounding AP's will be detected. This information will then be displayed by the GUI.

Case 4: Using the GUI to track a moving person for a given period of time.

Testers Instructions: The tester will choose an athlete from a given list then will start tracking by pressing a button on the GUI.

Conditions: Indoors or outdoors, both with direct line of sight.

Expected Observation: The GUI will graphically display the position of the moving person for a given period of time, within the error given by Case 6 of the Algorithm test.

4.6. Algorithm Tests

The following tests are developed to test the validity of the implemented algorithms.

Case 1: One dimensional distance verses power measurement.

Testers Instructions: The tester will measure the Wi-Fi power at various distances for a set number of samples. Do this for each AP.

Conditions: Indoors or outdoors, both with direct line of sight.

Expected Observation: This measurement will give the error distribution on the expected distance calculated from the RSS.

Case 2: Localization using just RSS

Testers Instructions: The tester will measure the Wi-Fi power from all APs at various locations around the playing field with a set number of samples. Compare the actual location with estimated location from the propagation model developed in case 1.

Conditions: Indoors or outdoors, both with direct line of sight.

Expected Observation: This measurement will give the approximate location of the DT. The calculated location is expected to be within the predicted error calculated from the error in case 1.

Case 3: Using probability filters to locate static positions.

Testers Instructions: The tester will measure the Wi-Fi power from all APs at various locations around the playing field with a set number of samples. This case the localization algorithm will use a probability filter to enhance the process. Compare the actual location with estimated.

Conditions: Indoors or outdoors, both with direct line of sight.

Expected Observation: The calculated location is expected to have smaller error than Case 2.

Case 4: Using dead reckoning to locate a moving person.

Testers Instructions: The tester will measure the accelerometer output from all Axis's walking and running at various speeds on a known path with a known starting spot.

Conditions: Indoors or outdoors, both with direct line of sight.

Expected Observation: The dead reckoning algorithm should be able to track the location of the person with the DT for a finite length of time before the error becomes too large.

Case 5: Using a combination of dead reckoning and RSS measurements to locate a moving person.

Testers Instructions: The tester will measure the accelerometer output and Wi-Fi power while walking and running at various speeds on a known path with a known starting spot.

Conditions: Indoors or outdoors, both with direct line of sight.

Expected Observation: The dead reckoning algorithm plus propagation model should be able to track the location of the person. The error should be within the lower range of error given by Case 4 and 2.

Case 6: Using a combination of dead reckoning and RSS measurements and probability filters to locate the moving Athlete.

Testers Instructions: The tester will measure the accelerometer output and Wi-Fi power while walking and running at various speeds on a known path with a known starting spot. The Tester will implement the probability filter.

Conditions: Indoors or outdoors, both with direct line of sight.

Expected Observation: The error when using this algorithm should have the least error of all the test cases.

5. Conclusion

The design specifications provided within this document will meet the functional requirements for the proof-of-concept device described in [1]. Also, through careful design, the specifications will also meet some of the requirements for the final product. Ideally many of the design solutions presented in this paper will be able to be used in the design specifications for a final product. To ensure that the required functional requirements will be met, a carefully thought out test plan has been devised for the PosiTrack team to carry out. Slight deviations from the specifications and test plan are expected but any major changes will require a revision to this document in order to reflect the changes.

6. References

[1] A. Hrehorciuc, J. Anderson, J. Valdes, R. Lynne, "*Functional Specifications for a System to Track Athlete Performance",* ENSC 440 Capstone Project, Simon Fraser University, Feb 2010.

[2] Mobile Whack Webpage – Laptop image source, Jan 18th 2010, http://www.mobilewhack.com/images/toshiba_satellite_a105_s4284_laptop.jpg

[3] J.M. Lee, "*Indoor Localization Scheme of a Mobile Robot Using RFID",* Presented at the 2005 International Symposium on Humanized Systems, Wuhan, China, 2005.

[4] C.-H. Lim et al., "A *Real-Time Indoor WiFi Localization System Utilizing Smart Antennas*", IEEE Transactions on Consumer Electronics, vol. 53, No. 2, May 1997.

[5] M. Ocaña, L.M. Bergasa and M.A. Sotelo, "*Robust Navigation Indoor using WiFi Localization*", Universidad de Alcalá, Madrid, Spain.

[6] V. Olivera, J. Cañas Plaza and O. Serrano, "*WiFi localization methods for autonomous robots*" in Robotica, vol. 24. UK: Cambridge University Press, 2006, pp. 455-461.

[7] Ladd A. M., Bekris K. E., Marceau G., Kavraki L.E., Wallach D. S., "*Robotics-Based Location Sensing using Wireless Ethernet*", *MOBICOM.02,* September 23.26, 2002, Atlanta, Georgia, USA.

[8] Senix Webpage – Acoustic transducer supplier, Jan 21st 2010, http://www.senix.com/products.htm

[9] Guoqiang M., Fidan B. "*Localization Algorithms and Strategies for Wireless Sensor Networks*", Information Science Reference, 2009, Hershey PA. USA.

[10] Dippold M. "*Personal Dead Reckoning with Accelerometers*" IFAWC2006 March 15-16, Mobile Research Center, TZI Universität Bremen, Germany.

[11] Gumstix Website – Original Overo Fire COM Picture, Mar 9th 2010, http://www.gumstix.com/store/catalog/images/view-O-FA-front.jpg

[12] Gumstix Website – Power Consumption, Mar 9th 2010, www.gumstix.net/Hardware/view/Benchmarks-power-temperatures/Power-Overo/112.html

[13] Wi2Wi W2CBW003 Specifications/Data Sheet, Mar 9th 2010, http://www.wi2wi.com/products/datasheets/W2CBW003_PB%20rev1.2.pdf

[14] VX Sport – Product Costs, Mar 9th 2010,
http://www.vxsport.com/content/vx-log™-–-hardware

[15] Gumstix Website – Original Tobi Expansion Board Picture, Mar 9th 2010,
http://www.gumstix.com/store/catalog/images/view-Tobi-front.jpg

[16] Gumstix Website – Tobi Expansion Board Schematics, Mar 9th 2010,
http://pubs.gumstix.com/boards/TOBI/PCB30002-R2549/

[17] Gumstix Website – 40-pin Expansion Header, Pin Description, Mar 9th 2010,
http://www.gumstix.net/Hardware/view/I/O-connectors-cabling/

[18] Sparkfun Website – IMU Original Picture, Mar 9th 2010,
http://static.sparkfun.com/images/products/09431-01.jpg

[19] Analog Devices, "ADXL335 Datasheet Rev. 0", 2009,
http://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf

[20] ST, "LY530ALH – Yaw Rate Gyroscope", Rev 2, July 2009,
http://www.sparkfun.com/datasheets/Sensors/IMU/LY530ALH.pdf

[21] ST, "LPR530AL – Dual Axis Pitch And Roll Gyroscope", Rev 2, July 2009,
http://www.sparkfun.com/datasheets/Sensors/IMU/lpr530al.pdf

[22] Sparkfun Website - 3.7V 1000mAh Original Picture, Mar 9th 2010,
http://static.sparkfun.com/images/products/00339-01-L.jpg

[23] Sparkfun Website – Maxim1555 Charge Board Original Picture, Mar 9th 2010,
http://static.sparkfun.com/images/products/00726-2.jpg

[24] Sparkfun Website – 6-DOF IMU Schematic, Mar 9th 2010,
http://www.sparkfun.com/datasheets/Sensors/IMU/6DOF-Razor-v11.pdf

[25] Sparkfun Website – Li-Ion Battery Charger Schematic, Mar 9th 2010,
http://www.sparkfun.com/datasheets/Batteries/LiPo-USB-Charger-v13.pdf

[26] Gumstix Website – Overo Signal Description, Mar 9th 2010,
http://www.gumstix.net/images//gumstix%20overo%20signals%20v1.0.pdf

