Activity Monitor

EECE 496 – Final Report

4/5/2010

Author: Jason Poon

Technical Supervisor: Dr. S. Gopalakrishnan

# Abstract

This report describes the design and development of an activity monitoring system. The system utilizes FireFly sensor nodes placed throughout an individual’s body to form a wireless sensor network. The accelerometer data collected from the wireless sensor network is eventually forwarded to a processing client. Using various forms of analysis, the client will classify the user’s motions under one of several categories. Such an activity monitoring system can be used to monitor an individual’s exercise habits.

Table of Contents

[Abstract 2](#_Toc258818243)

[Table of Figures 5](#_Toc258818244)

[List of Tables 6](#_Toc258818245)

[List of Abbreviations 7](#_Toc258818246)

[List of Abbreviations 7](#_Toc258818247)

[1. Introduction 1](#_Toc258818248)

[2. System Overview 2](#_Toc258818249)

[2.1. Features and Functionality 2](#_Toc258818250)

[3. Wireless Nodes 4](#_Toc258818251)

[3.1. Hardware 4](#_Toc258818252)

[3.1.1. FireFly Nodes 4](#_Toc258818253)

[3.1.2. FireFly Programmer 5](#_Toc258818254)

[3.2. Real-Time Operating System (RTOS) 5](#_Toc258818255)

[3.2.1. Comparison 6](#_Toc258818256)

[3.2.2. Installation 6](#_Toc258818257)

[3.3. Software 7](#_Toc258818258)

[3.3.1. Sensor Nodes 7](#_Toc258818259)

[3.3.2. Gateway Node 9](#_Toc258818260)

[4. Server 11](#_Toc258818261)

[5. Client 12](#_Toc258818262)

[5.1. SlipStream 13](#_Toc258818263)

[5.2. SensorList 13](#_Toc258818264)

[5.3. Progress Bar 14](#_Toc258818265)

[5.4. Modes of Operation 14](#_Toc258818266)

[5.4.1. Hardware Calibration 14](#_Toc258818267)

[5.4.2. Activity Monitor 18](#_Toc258818268)

[6. Future work 21](#_Toc258818269)

[7. Conclusion 22](#_Toc258818270)

[8. Works Cited 23](#_Toc258818271)

[Appendices 24](#_Toc258818272)

[A. Source Code 25](#_Toc258818273)

[B. Acceleration Graphs 26](#_Toc258818274)

# Table of Figures

Figure 1: System block diagram 2

Figure 2: FireFly Node with Attached Sensor Expansion Board 4

Figure 3: Firefly Debugging Board/Programmer 5

Figure 4: Sensor Node Sequence Diagram 8

Figure 5: Sensor Node Message Format 8

Figure 6: SLIP Packet Format 9

Figure 7: Gateway Node Sequence Diagram 10

Figure 8: Client System Class Diagram 12

Figure 9: Uncalibrated Accelerometer Data 15

Figure 10: Accelerometer output response relative to orientation to gravity 15

Figure 11: Voltage to Acceleration Conversion EQuation 16

Figure 12: Calibrated Accelerometer Data 17

Figure 13: Calibration Mode Sequence Diagram 18

Figure 14: Decision Tree for Classifying user Motions 20

Figure 15: Monitor Mode Sequence Diagram 20

Figure 16: Acceleration Graph Showing Difference between axes when Sensor is Horizontal vs. Vertical 26

# List of Tables

[Table 1: Nanork and LiteOS Comparison 6](#_Toc258815638)

[Table 2: Calibration Positions 16](#_Toc258815639)

# List of Abbreviations

|  |  |
| --- | --- |
| ADC | Analog-to-Digital |
| API | Application programmer’s interface |
| ASCII | American Standard Code for Information Interchange |
| B-MAC | Carrier sense media access protocol for wireless sensor networks |
| ISP | In-System Programming |
| PNG | portable network graphic |
| RTOS | Real-time operating system |
| RS-232 | Recommended Standard 232. Standard for serial binary single-ended data and control signals connecting a data terminal equipment and a data circuit-terminating equipment |
| SLIP | Serial line internet protocol |
| SPI | Serial Programming Interface |
| SVG | scalable vector graphic |
| UART | Universal asynchronous receiver/transmitter |
| UDP | User Datagram Packet |
| USB | Universal Serial Bus |
|  |  |

# Introduction

The project entails the development of an activity monitor where sensory data obtained via wireless nodes attached to the body are used to determine an individual’s movement. The following report outlines the behavior of the system.

The objectives for this project are to design and develop a system that is able to monitor a user’s activities based on data retrieved from accelerometers on wireless sensor nodes that are attached to an individual. This involves the development of a software framework on the wireless sensor nodes and a client on the gateway computer to process the data retrieved from the wireless nodes.

An activity monitoring system will provide a mechanism to unobtrusively monitor an individual’s daily activities. Additionally, the system can also be used to track an individual’s exercise routine.

The scope of the project is limited by the FireFly hardware. Being a wireless node, the FireFly sensor node possesses limited memory and processing power.

This report begins with a high-level overview of the system, and Section describes wireless nodes used for the system the hardware used for the system. Section presents the Server and Section , the client. Finally, Sections and outlines future work and offers concluding discussions.

# System Overview

The activity monitoring system is composed of three subcomponents: the wireless nodes, the server, and the client. Five wireless nodes are used to form the wireless sensor network. Sensory data obtained by the wireless nodes is transmitted via RS-232 to the server which then forwards the data to a client via UDP for processing.

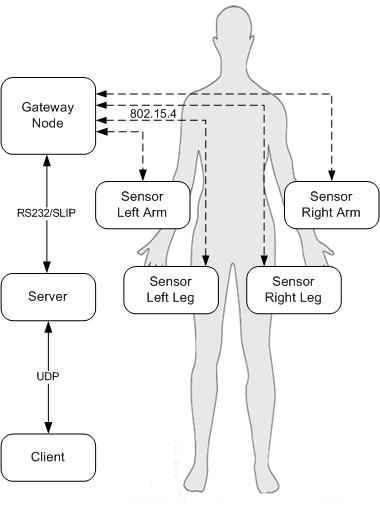


Figure : System block diagram

## Features

The system currently contains the following features and functionalities:

* Classification of the User’s Current Motion
  + Using the sensor data retrieved from the wireless sensor network, the system is able to classify a user’s motion as either walking, running, sitting, standing, or lying down
* Programmatic Generation of Acceleration Graphs
  + Line graphs of the acceleration are programmatically generated for each sensor node. The graphs are of scalable vector graphic (SVG) type but can also be converted to portable network graphic (PNG) type.
* Terminal-Based User Interface
  + The user interface prompts provides users with an informative display; the messages displayed on the terminal are also customizable.
* Calibration of Sensor Nodes
  + The sensor nodes can be calibrated such that the values read by the accelerometers are converted to a more understandable unit such as acceleration in G’s. Calibration data can be saved to or loaded from a file.

# Wireless Nodes

## Hardware

Two types of hardware components are required for the activity monitor: FireFly nodes and a FireFly debugging board.

### FireFly Nodes

The wireless sensor network is composed of FireFly nodes. Each node features an Atmel Atmega128L microcontroller and uses IEEE 802.15.4 for wireless communication . The nodes are powered by 2-AA batteries.

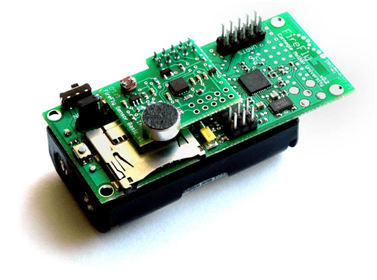


Figure : FireFly Node with Attached Sensor Expansion Board

The node itself does not have any sensors; however, a sensor expansion card can be attached via the analog-to-digital converter (ADC) port present on the Firefly node. The sensor expansion card features the following sensors:

* Light
* Temperature
* 3-Axis Accelerometer
* Audio
* Battery Level

The activity monitor requires only the use of the tri-axis accelerometer. The FireFly node uses the ADXL330 (Analog Devices, Inc.) which is capable of measuring acceleration up to 4G.

### FireFly Programmer

The FireFly debugging board is used to program the node as well as to send and receive serial data from the connected sensor node.

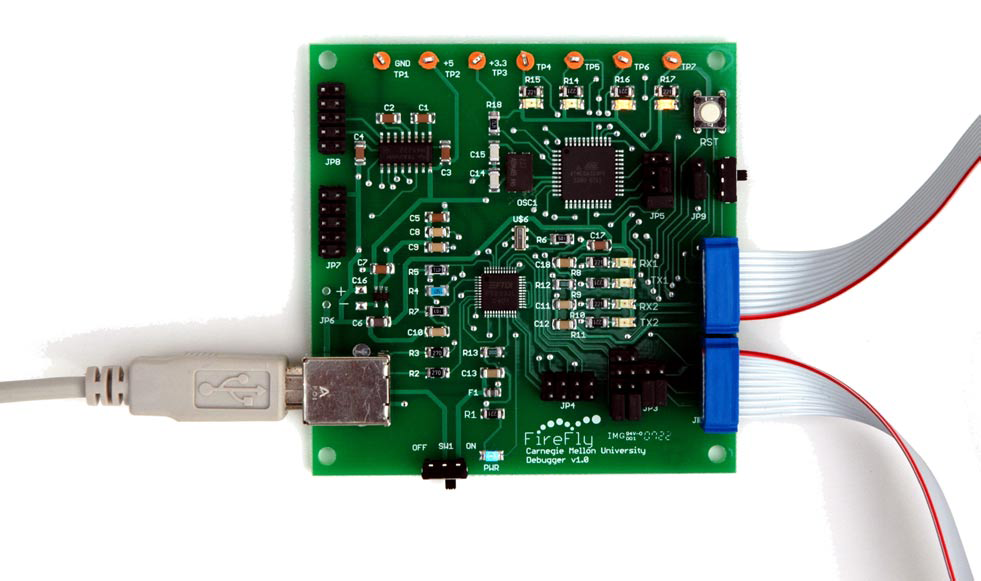


Figure : Firefly Debugging Board/Programmer

The two ribbon cables of the debugging board are attached to the sensor node’s header 1 (UART1) and header 2 (SPI/ISP). The USB cable is connected directly to the computer; once attached to the computer, two serial ports should be started: /dev/ttyUSB0 and /devttyUSB1.

## Real-Time Operating System (RTOS)

The benefit of using a real-time operating system (RTOS) in combination with the FireFly hardware node is that it allows abstraction of task scheduling and other low-level operating system functions. In particular, the usage of a RTOS allows tasks to be served in real-time.

### Comparison

Three major RTOS currently exist for wireless sensor networks: TinyOS, NanoRK and LiteOS. However, TinyOS applications are coded using nesC, a variation of the C programming. In order to limit the learning curve associated with building the activity monitor, TinyOS was not considered for use for the activity monitor. Table 1: Nanork and LiteOS Comparison offers a side-by-side comparison between NanoRK and LiteOS.

|  |  |  |
| --- | --- | --- |
|  | NanoRK | LiteOS |
| Installation | * Simple and straightforward | * Out-dated installation information * Buggy installer |
| Documentation | * Large website filled with documentation on various APIs | * Heavily focused towards Windows users |
| Community | * Aside from the project contributors, no community | * Small, but helpful community |

Table : Nanork and LiteOS Comparison

Due to the various problems experienced with LiteOS and NanoRK’s helpful documentation, it was decided that the activity monitor would be built on top of the NanoRK platform.

### Installation

The installation of NanoRK on the sensor nodes is a well-documented process. In order to build NanoRK, the following software tools are required:

* flex-2.5.4a-34
* bison-2.0-6
* byacc-1.9-29
* gcc-4.0.2-8
* gcc-c++-4.0.2-8
* make
* binutils-2.18
* gcc-core-4.2.2
* avr-libc-1.6.1
* avrdude-5.0

Upon obtaining the necessary software tools, the many example programs included with NanoRK can be used to test the installation process. Further instructions can be found on the website of NanoRK .

## Software

The wireless nodes can be classified as either sensory nodes or the gateway node. Sensory nodes are placed throughout an individual’s body and continually gather and wirelessly send accelerometer data to the gateway node. The gateway node is connected to a computer via the FireFly programmer and acts as a intermediary between the sensory nodes and the server running on the computer.

### Sensor Nodes

The four sensor nodes are placed on an individual’s left leg, right leg, left arm, and right arm. Once turned on, the sensor nodes execute two tasks: the sensor task and the communication task.



Figure : Sensor Node Sequence Diagram

The sensor task samples the accelerometer at a rate of 6.67Hz. To ensure an accurate reading, 10 samples are collected and the average of the 10 samples is then transmitted to the gateway node. As a result, accelerometer data is sent to the gateway node at a rate of 0.667Hz. The averaged accelerometer data is formatted into a string (ASCII) . The message format is shown in Figure 5.



Figure : Sensor Node Message Format

In order to signal to the communication thread that data is ready to be sent, the sensor task sets the flag txPktReady to true. Once the txPktReady flag has been set, the slipstream task transmits the packet via the B-MAC protocol and the flag is unset when transmission is complete. The above process is repeated indefinitely.

Although the maximum communication range of the FireFly nodes is 100 meters, distances greater than 50 meters will experience heavy packet loss. As such, the sensor nodes should be placed no further than 50 meters from the gateway node.

### Gateway Node

The gateway node serves as an intermediary between the server and the sensor nodes. In order to allow for bi-directional communication between the gateway node and the server, the serial line internet protocol (SLIP) is used. The SLIP protocol is designed to work over serial and allows for detection of transmission errors of the datagram packets being sent and received from the sensor nodes . NanoRK’s implementation of SLIP is known as SLIPstream and uses a modified version of the SLIP protocol. The SLIP data format used by NanoRK is shown in Figure 6.

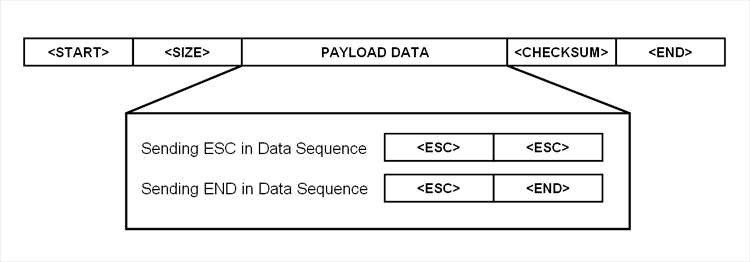


Figure : SLIP Packet Format

The call diagram of the gateway node is similar to that of the sensor node (see Figure 4). When a message is received from the sensor nodes, the packet is parsed into a SLIP packet. Setting the slipTxReady flag will then cause the SlipStream thread to send the SLIP packet to the server.



Figure : Gateway Node Sequence Diagram

# Server

The server is coded in C and performs two functions:

1. Act as a terminal program that displays debugging messages from the gateway node.
2. Serve as a UDP server that allows for other programs to communicate with the gateway node.

By default, the server will listen to SLIP messages from the programming board on /dev/USB0. When a message is SLIP message is received, the message is parsed and broadcasted to all connected UDP clients. The default listening port for the server is 4000.

Default options can be changed by passing in various arguments when starting the server. The possible program options can be found via the server’s help message:

Usage:

./startServer [-h] [-d] [-c DEVICE] [-p SERVER PORT]

Options:

-h Prints this message and exits

-d Turns on debugging. Shows SLIP packets and   
 incoming datagrams

-c Device in which programming board is connected

-p Port on which to listen for connections

# Client

The client is written in Python and communicates to the server via a UDP connection. As such, the client can reside on the same physical machine as the server or anywhere else within the network. Additionally, due to the nature of a UDP connection, it is also possible to have more than one client connected to the single server.

As shown by Figure 8, the client entails several components including communication with the server (SlipStream), parsing of accelerometer data (SensorList), progress bar (Progress\_Bar), and two modes of operation: hardware calibration (Calibrate\_Thread) and activity monitoring (Monitor\_Thread).

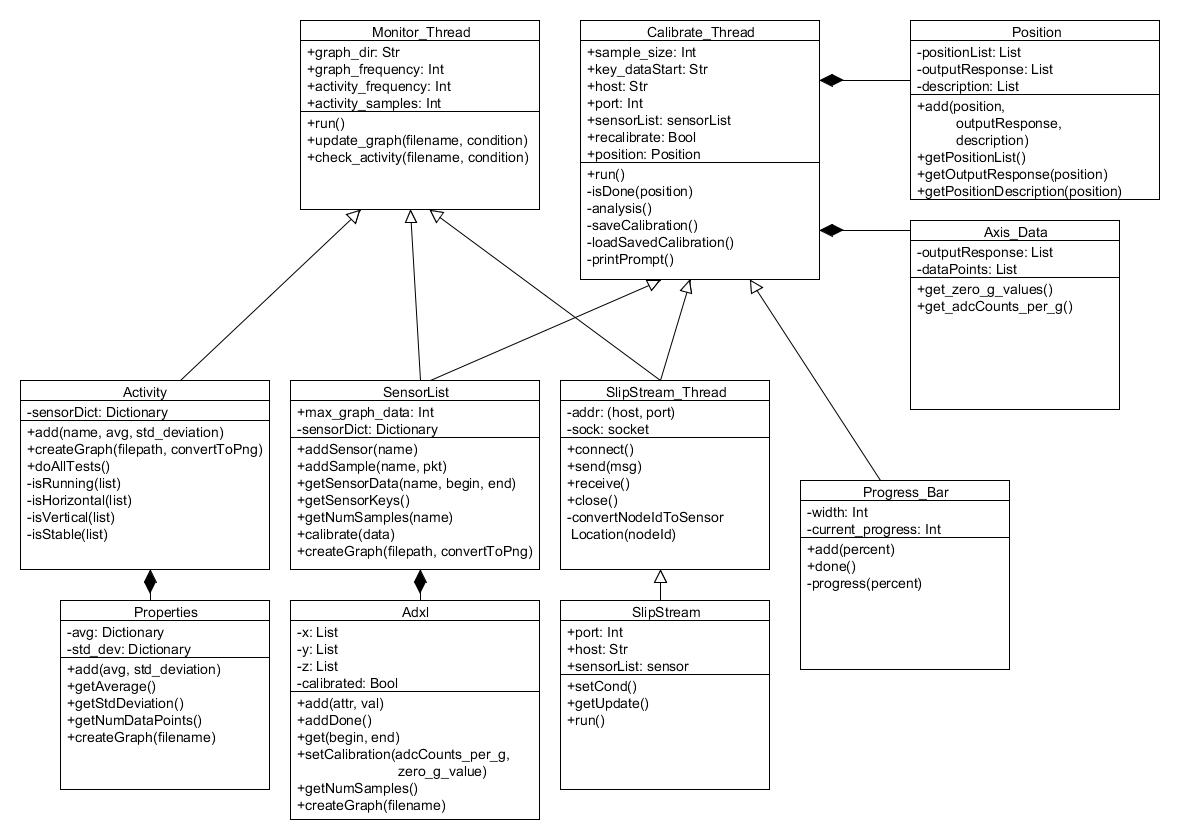


Figure : Client System Class Diagram

When executing the client, it accepts several arguments and options that can be used to configure start up.

Usage:

./startClient [-h] [-c] [-d] [-a Server Address]   
 [-p Server Port]

Options:

-h, --help

Prints this message and exits

-c, --calibrate

Recalibrates the sensors. If option is not set,   
 we use the previously saved hardware calibration data.

-d, --debug=

Logging levels. Possible options are 'debug', 'info',

'warning', 'error', 'critical'.

-a, --addr=

Address in which server is located (e.g. 127.0.0.1)

-p, --port=

Port in which server is located (e.g. 4000)

## SlipStream

The SlipStream\_Thread runs in an infinite loop listening for messages from the server. A parent thread can receive a callback event by calling setCond()with a Python condition variable as an argument. When data is received by SlipStream\_Thread, it will notify the parent thread via the condition variable object.

The client will attempt to connect to the UDP server located on the local host (127.0.0.1) on. These options can be changed with the “-a” and “-p” flags when starting the client.

## SensorList

As messages received from the server from the gateway node are sent in ASCII, it is necessary to parse them into a more usable format prior to performing any analysis. The SensorList class acts as a mechanism to parse and store the data received from the wireless sensor network.

The SensorList class contains a private member variable named sensorDict. sensorDict is a dictionary whose keys are that of the possible sensor positions: “left\_arm”, “left\_leg”, “right\_arm”, and “right\_leg”. The value associated with each key is an Adxl object which holds the axis information for each sensor node. A feature of the sensorList class is the ability to programmatically generate line graphs of the data in the Adxl object. The graphs are generated using a Python module titled CairoPlot . Examples of these graphs are shown as Figure 9 and Figure 12.

## Progress Bar

The Progress\_Bar class is used to provide users with a graphical interpretation of the current progress of a certain task. The progress bar is composed of ASCII characters and its appearance is similar to that of below:

[============ ] 40%

[======================================] 100%

## Modes of Operation

The client can be run in two modes of operation: hardware calibration and activity monitoring.

### Hardware Calibration

The values returned from the ADC (e.g. accelerometer) are analog voltage values. Figure 9 depicts the voltage values for each axis as the wireless node is transitioning from lying flat on a table to standing upright. An important thing to note is that each axis has a different reference value; as such, it is necessary to standardize the data points.

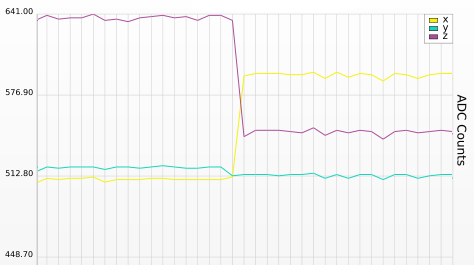


Figure : Uncalibrated Accelerometer Data

In order to standardize the data points, we much calibrate each sensor node such that we can determine a reference level. As shown by Figure 10, the output response of the accelerometer is dependent on its position relative to gravity.

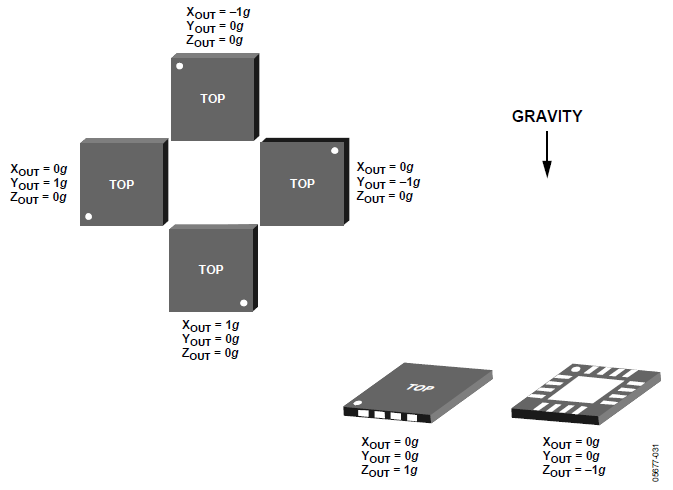


Figure : Accelerometer output response relative to orientation to gravity

In order to retrieve enough data samples to calibrate each axis, each wireless node is placed in 3 positions. This provides us with three different datasets for each axis.

|  |  |  |
| --- | --- | --- |
|  | Position Description | Output Response (x, y, z) |
| 1 | Lying flat horizontally | 0, 0, 1 |
| 2 | Lying upright with ‘FireFly’ text on the node upright | 1, 0, 0 |
| 3 | Lying on its side with LEDs on top edge | 0, 1, 0 |

Table : Calibration Positions

For each axis, the reference voltage is set to the voltage where the output response is equal to zero. The conversion rate of voltage to acceleration for a particular axis can then be calculated using the equation:

Figure : Voltage to Acceleration Conversion EQuation

v1 and r1 respectively refer to the voltage read and the output response at one of the calibrated position for a particular axis. The result of such a calculation will have units of voltage/acceleration (g). Figure 12 depicts the same scenario as Figure 9; the sensor node is transitioning from lying flat on a table to standing upright. An important thing to note is that the data points are now in units of acceleration and each axis can be relatively compared against one another.

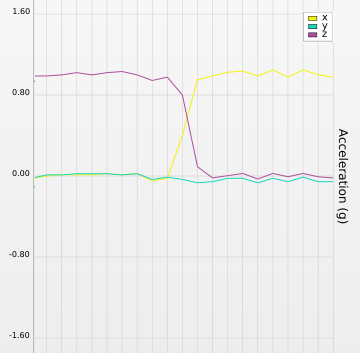


Figure : Calibrated Accelerometer Data

Assuming that there are no major changes in altitude, the reference voltage and conversion rate remain constant for each axis. As such, the calibration data for each sensor node is saved to a file to be loaded in the future. pickle, a Python object serialization module, is used to save and load the calibration data.

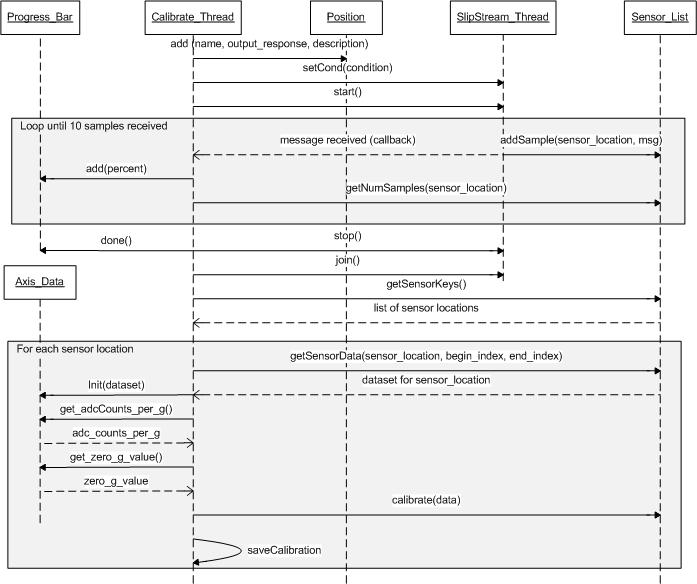


Figure : Calibration Mode Sequence Diagram

### Activity Monitor

With each sensor node properly calibrated, it is now possible to analyze the position of each sensor node to determine the user’s current activity. The sensor data undergoes operations: preprocessing, feature extraction, and classification.

#### Preprocessing

With the accelerometers placed on the human body, some noise is to be expected. As such, the data received by the client from the sensor network is actually the average of 10 samples. Additionally, the continuous accelerometer data stream is divided into running sets of 5 samples. As the client receives the sensor packets in no specific order from the sensor nodes, it is difficult to keep a data stream of a particular sensor location (e.g. left arm) in sync with that of another location (e.g. right arm). As such, the accelerometer data stream is segmented depending on elapsed time; for every 2 seconds that have past, the previous 5 samples of each axis for every sensor location is used for feature extraction and classification.

#### Feature Extraction

Following the segmentation of the data, the sensor data is analyzed using two metrics: average and standard deviation. Although more complex statistical and spectral data can be extracted from the data (e.g. energy, correlation), the calculation of the mean and the standard deviation is computationally inexpensive and sufficient enough to classify the data into one of five activities: standing, sitting, lying down, walking, or running.

#### Classification

An individual’s motion can be classified under two categories: stationary or moving. The stationary set of activities includes sitting, standing, and lying down whereas moving activities include running and walking.

Using the average and the standard deviation extracted from the dataset, it is possible to classify the user’s motions into one of activities.

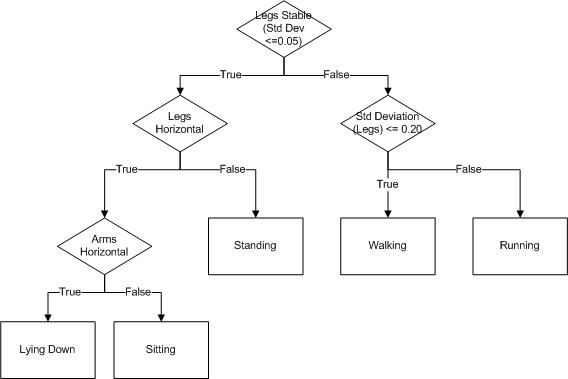


Figure : Decision Tree for Classifying user Motions

Figure 14 shows the decision tree that was used to classify the user’s motions. For each sensor positioned on the leg, if the standard deviation of each axis is below a threshold value of 0.05 then the motion is deemed to be stationary; otherwise, it is moving. The sensors are deemed to be in a horizontal position if the x-axis is equal to 0 ± 0.05.

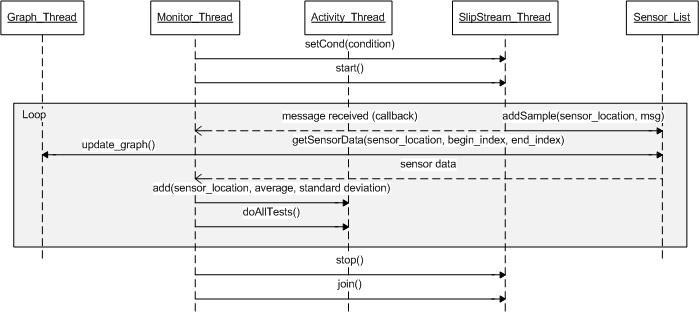


Figure : Monitor Mode Sequence Diagram

# Future work

The activator monitor system can be extended to recognize upper body movements in conjunction with the lower body movements currently recognized. As sensors are already being placed on a user’s arm, the feature extraction and classification process can be expanded to include an analysis of an individual’s arms.

As the feature extraction currently employed by the system is very limited, the analysis can be expanded to include a Fourier Transform analysis and correlation between each pair of axes. By working further on the feature extraction, it may also be possible to differentiate between more movements such as brisk walking and jogging.

Furthermore, a graphical user interface would greatly benefit the end-user. The user interface could visually display the current position of the nodes as well as a node’s status including current battery level.

# Conclusion

The activity monitor is a real-time system that is able to recognize an individual’s various activities. The system uses FireFly nodes that are placed throughout an individual’s body to form a wireless sensor network. The wireless sensor network continuously samples and sends accelerometer readings. The gateway node and server act as intermediaries where the client is responsible for analyzing the data.

Following the calibration of the accelerometer data, the client uses mean and standard deviation to categorize the motions of the sensor nodes as walking, running, standing, sitting, or lying down.

The design and implementation of the activity monitor project has been a rewarding and useful learning experience. Through this project, I have gained invaluable knowledge in software design and development

# Works Cited

1. **Rowe, Anthony, Mangharam, Rahul and Rajkumar, Raj.** *FireFly: A Time Synchronized Real-Time Sensor Networking.* 2005.

2. Nano-RK. [Online] March 7, 2010. [Cited: Februrary 20, 2010.] http://www.nano-rk.org/.

3. *Versatile Low Power Media Access for Wireless Sensor Networks.* **Polastre, Joseph, Hill, Jason and Culler, David.** Baltimore : s.n., 2004.

4. **Romkey, J.** A Non-Standard For Transmission Of IP Datagrams Over Serial Lines: SLIP. *Network Working Group: Request for Comments 1055.* 1988.

5. CairoPlot. *CairoPlot 1.1.* [Online] September 16, 2008. [Cited: February 20, 2010.] http://linil.wordpress.com/2008/09/16/cairoplot-11/.

6. *Physical Activity Recognition Using a Single Tri-Axis Accelerometer.* **Lee, Me-hee, Kim, Jungcae and al, et.** San Francisco : s.n., 2009. World Congress on Engineering and Computer Science.

7. *Real Time Human Activity Recognition Using Tri-Axial Accelerometers.* **Krishnan, Narayama, Colbry, Dirk and al., et.** Sedona, AZ : s.n., 2008. Sensors Signals and Information Processing Workshop.

8. *Activity Recognition from Accelerometer Data.* **Ravi, Nishkam, Dandekar, Nikhil and all, et.** 2005. American Association for Artificial Intelligence.

# Appendices

## Source Code

The source code is available at http://github.com/jpoon/eece496.

## Acceleration Graphs

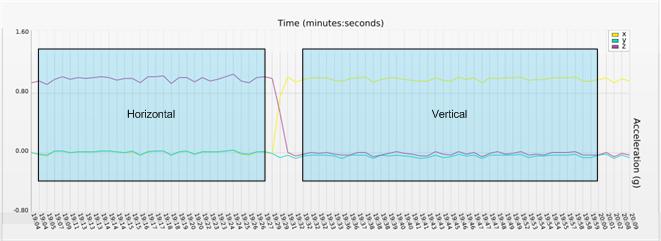


Figure : Acceleration Graph Showing Difference between axes when Sensor is Horizontal vs. Vertical