Health C²UBE

Physical Therapy Cadence Compliance for UBE Device

Team Members:
Daniela Falcone
Sophine Kouleya
Christos Kombouras
Kozeta Mimani
Zeid Muhammad
Nick Stamos
Gregory Tuballes

Capstone Advisor:
Prof. Masoud Salehi

Collaborators:
Prof. Charles DiMarzio – ECE
Prof. Karen Hutchinson - PT
0.1 ABSTRACT
Amputees and patients with myelopathy have difficulties with the accessibility of health care facilities for physical therapy. Patients are prescribed exercise regimens to do within their own home usually with an Upper Body Ergometer (UBE) to assist in their rehabilitation. Statistically, disabled patients have lower heart rate variability (HRV) and tracking such measures is imperative due to the fact that measurement of said data is a strong indicator of cardiovascular health. The UBE is a common cycling exercise machine made for amputee patients or those with myelopathy.

Therapists complain of patients not following their prescribed home workout regimens and of poorly self-logged progress reports. Our objective is to fabricate a device, Health C^2UBE, to monitor and evaluate compliance of a patient during cardiac rehabilitation in addition to extracting “concrete” data for analysis.

Health C^2UBE will be used together with a UBE to monitor the heart rate, heart rate variability and revolutions per minute (RPM). The Health C^2UBE will be capable of exporting and storing the results of the patient to a server by way of Ethernet. It makes it possible for physical therapists or other medical professionals to monitor and view the information from a distance outside the hospital or clinic. The device is unique since it allows a physical therapist to monitor the progress of a patient’s workout regimens via the web; whereas other similar devices do not. In our following report, we will illustrate our guidelines and design, in which engineering can continue to improve the quality of life for those in medical need.
# 0.2 TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 Abstract</td>
<td>2</td>
</tr>
<tr>
<td>0.2 Table of Contents</td>
<td>3</td>
</tr>
<tr>
<td>1.0 Introduction</td>
<td>4</td>
</tr>
<tr>
<td>1.1 Problem Formulation</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Targeted Areas of Measurement</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 Heart Rate</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2 Heart Rate Variability</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Today’s Technologies</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Health C³UBE vs. Competing Products</td>
<td>6</td>
</tr>
<tr>
<td>2.0 Analysis</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Initial Setup</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Simple Interface</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Internet Connections</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Automatic Notifications</td>
<td>7</td>
</tr>
<tr>
<td>2.5 Ergonomics &amp; Safety</td>
<td>8</td>
</tr>
<tr>
<td>2.6 Size</td>
<td>8</td>
</tr>
<tr>
<td>2.7 Power Consumption</td>
<td>8</td>
</tr>
<tr>
<td>3.0 Hardware</td>
<td>8</td>
</tr>
<tr>
<td>3.1 Heart Rate Monitor</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Cadence Sensor</td>
<td>9</td>
</tr>
<tr>
<td>3.3 Gumstix Computer</td>
<td>9</td>
</tr>
<tr>
<td>3.3.1 Input Interface</td>
<td>9</td>
</tr>
<tr>
<td>3.3.2 Output Interface</td>
<td>10</td>
</tr>
<tr>
<td>3.4 The Interfacing of Major System Components</td>
<td>11</td>
</tr>
<tr>
<td>3.4.1 Operation</td>
<td>11</td>
</tr>
<tr>
<td>3.4.2 Signal Processing/data Transmission</td>
<td>11</td>
</tr>
<tr>
<td>3.4.3 Power</td>
<td>11</td>
</tr>
<tr>
<td>4.0 Technical &amp; Software Approach</td>
<td>12</td>
</tr>
<tr>
<td>4.1 System Overview</td>
<td>12</td>
</tr>
<tr>
<td>4.2 Data Collection &amp; Manipulation</td>
<td>12</td>
</tr>
<tr>
<td>5.0 Website</td>
<td>14</td>
</tr>
<tr>
<td>6.0 Budget Analysis</td>
<td>15</td>
</tr>
<tr>
<td>6.1 Initial Prototype</td>
<td>15</td>
</tr>
<tr>
<td>6.2 Envisioned Final Product</td>
<td>15</td>
</tr>
<tr>
<td>7.0 Future Improvements</td>
<td>16</td>
</tr>
<tr>
<td>7.1 C++ vs. Python</td>
<td>16</td>
</tr>
<tr>
<td>7.2 New Improved Gumstix</td>
<td>16</td>
</tr>
<tr>
<td>7.3 New Parsing Location</td>
<td>16</td>
</tr>
<tr>
<td>7.4 UTC Time</td>
<td>17</td>
</tr>
<tr>
<td>7.3 Webcam Integration</td>
<td>17</td>
</tr>
<tr>
<td>7.4 Resistance Measures</td>
<td>17</td>
</tr>
<tr>
<td>Appendix A - Gumstix Coding</td>
<td>18</td>
</tr>
<tr>
<td>Appendix B – Website Code</td>
<td>24</td>
</tr>
<tr>
<td>References</td>
<td>29</td>
</tr>
</tbody>
</table>
1.0 INTRODUCTION

1.1 Problem Formulation
This biomedical project was chosen at the request and in conjunction with the Northeastern University Physical Therapy department for amputees and patients with myelopathy undergoing cardiovascular rehabilitation. Myelopathy is an injury to the spinal cord that results in paralysis from the point of injury and below and thus often restricting patients to wheelchairs.

For our demographic, rehabilitation programs can be prescribed to be completed independently in the confines of a patient’s own home with the use of the Upper Body Ergometer or UBE, which is a common exercise machine for said demographic. However, with the prescription of at-home exercise, compliance with the recommended workout can require enforcement. Currently, patients are encouraged to keep log books describing their workout. However, this can lead to falsified information or poorly kept records which in turn can cause the patient to not reap the benefits of his or her own rehabilitation program. This is where our project comes in. It is designed to kill two birds with one stone:

1. To ensure that workouts are completed regularly by patients.
2. To provide therapists with concrete data in an efficient manner in which they can better analyze their patient’s progression and adjust it accordingly. Such data will include the exact time of each heart beat, BPM, Heart Rate Variability, Speed (RPMs) & time.

1.2 Targeted Areas of Measurement

1.2.1 Heart Rate
Heart rate is the most common way to monitor an individual’s pulse. Standard calculation consists of taking a sample of heart beats in a minute to find the beats per minute (bpm). Monitoring the heart rate bpm is a good standard to measure cardiac adaptability of athletes. Much of what a resting or target heart rate should be are based on an individual’s age. For example:

A 20 year old user should be capable of having a maximum heart rate of \((220 - 20)\) or 200 bpm.

Depending on the intensity a target heart for a 20 year old user can be established:

- 50% intensity \(\rightarrow .50 \times \text{max heart rate} = 100\text{bpm}\).
- 70% intensity \(\rightarrow .7 \times 200 = 140\text{bpm}\). \[1\]

1.2.1 Heart Rate Variability
Heart Rate Variability (HRV) is the measurement of beat-to-beat changes in the heart rate. Looking at figure 1, the heart rate variability is calculated as the time between the “R” wave of the PQRST wave and the next, referred to as the “R-R interval.” Concordantly, we can extrapolate the bpm if one were to assume that the time between each R wave was consistent for a whole minute. Therefore, between heartbeats we can estimate the change in bpm \[2\]. This approach is a calculation of HRV in the time domain. Calculations in the frequency domain are commonly used in industry; however, for our application, time domain analysis will only be possible. Looking at this small
sample of 4 heart beats in 2.5s, it can be deduced that the user is undergoing some stress, whether it’s physical or nervous, and that the heart is compensating by increasing in rhythm.

**Figure 1: Example of R-R intervals**

Understanding the HRV is important for several reasons. It can provide information as the instantaneous changes the heart undergoes when the user is forced to adapt to a changing physical environment (i.e. a user running on a flat terrain switching to an incline or decline) as opposed to having a comparison heart reading based on a sample during a 60 second sample.

Furthermore, “systems-oriented models propose that HRV is an important indicator of both physiological resiliency and behavioral flexibility, reflecting the…capacity to adapt effectively to stress…it has become apparent that while a large degree of instability is detrimental to efficient physiological function, too little variation can also be pathological (attributable to the cellular level). An optimal level of variability…is critical to the inherent flexibility and adaptability that epitomizes health function,” [2].

Historically, HRV measurements have been used specifically for it’s predictability of mortality for patients suffering acute myocardial infarction as well as users with post infarction (blood clot) and diabetes [3, 4]. It is a preferred method of assessing ANS “control of the heart,” due to its non-evasiveness.

Moreover, the Institute of HeartMath has established that the autonomic nervous system (ANS), which regulates the SA node of the heart and therefore heart rate, can be influenced by emotional states of an individual. In fact, the IHM’s have conducted studies of emotional changes by monitoring changes in HRV [2].

**1.3 Today’s technologies**

A widely used device that many athletes use is a Polar™ Chest Belt sensor. The chest belt straps around the user’s torso with the transmitter located at the top of the solar plexus. Two electrodes within the strap then monitor the electrical signal of the heart at the skin of the user. Polar™ also offers digital watches that can receive the wireless signal and is the most common application of the chest belt sensor. Software embedded within the watch will then calculate the user’s heart rate and display it on its LCD. This is an advantageous technology since the user will have real-time data always available to them. However, this device does not provide HRV; only real-time heart readings.

MarkOfFitness™ offers the MF-180. The MF-180 is a small hand strap with a LCD display. Unlike the chest belt sensor, the MF-180 measures the heart rate via plethysmography. In other words, instead of measuring the electric signals of the heart, the MF-180 calculates heart beat off an algorithm dependent on how the volume of the hand and fingers change. These changes in volume are due to the systolic blood pressure as a result of the pressure of the heart pumping blood [5]. As mentioned in the previous
example, the downfall of is that this product only provides the user with real-time heart rate readings in bpm. Furthermore, this type of heart rate sensor is not commonly used in the heart monitoring industry since some websites I have encountered have suggested that this type of method is less than ideal. Moreover, heart rate readings via plethysmography are usually taken at the leg and not the hand.

Another device that is commonly used to monitor heart rate is the Holter monitor which is more commonly used today in the medical industry (i.e. monitoring patients who recently suffered a heart attack, patients with possible arrhythmia & patients starting new heart medication, etc). Holter monitors employ a series of electrodes on the user’s body to monitor the heart rate rhythm as opposed to the embedded straps of said Polar™ chest belt sensor. The holter monitor itself than records these readings, usually to flash memory, for usually a span of 24 hours [6]. The holter monitor is able to provide users and medical professionals with a wide range of data, including both heart rate and HRV. It is dependent on the user providing detailed accounts of their daily activities. While this device is more accurate it is also less versatile and available to common users. It requires specific electrode placement and is dependent on the user carrying the holter monitor; an undesirable feature for an active rehabilitating user. In terms of wide scale use, holter monitors are the most expensive of the aforementioned 3.

1.4 Health C²UBE vs. Competing Products
What all 3 of the previously mentioned products share is that they cannot provide real time data to a remotely located third party patron. The first two only provide information as the exercise is completed; not storing any data to memory. The latter of the 3 is by far the most useful for analytical purposes, however, as previously mentioned it is highly priced and requires a level of guidance for users from medical professionals for electrode placement. The Health C²UBE makes use of the technology of the Polar Chest Belt sensor coupled with a small computer which will allow it to provide internet accessible data to users at both ends. The final cost of the Health C²UBE will be significantly higher than the first two mentioned, however its use and application will be on a much higher scale. Compared to the holter monitor, the Health C²UBE is designed only to be used when exercising and only within a specific range of the portable receiver. However, the level of application for the contrasting demographics is far in-between. Full day provision of HRV data is not necessary for our demographic. Provision of such information is only needed to understand the cardiovascular health during rehabilitation of our handicapped patients and not for patients with heart disorders. However, in comparison, the Health C²UBE will cost significantly less and will also trump the Holter monitor in that it provides information through the internet and that it can be used in the home of our user indefinitely, where as the holter monitor is typically returned to medical professionals after a period of 24 hours for data acquisition.
2.0 ANALYSIS
In order to save the patient time and stress of having to commute to the Physical therapist’s office, we needed to design a device that is portable, requires little technical assistance, and can be easily mounted on any Upper Body Ergometer (UBE).

2.1 Initial Setup
The initial setup of the Compliance Device needs to be as user friendly as possible for the patient. Sensors shall be retrofitted onto the UBE by the Physical Therapist. No software installation required for the user since all regiment information shall be logged and accessible from the website.

- The device shall be able to operate on a standard 120V outlet.
- The device shall clear its memory after uploading a report; limiting the actions required by the user.
- Compliance with the prescribed rehabilitation does not require the user to take any other action than exercise. All reports and data are uploaded autonomously. However, the patient will need to log onto the website to view his/her own data.
- Reports and data that are uploaded are viewable and read-only files. The user is not able to tamper the data; further ensuring compliance.
- All reports that are uploaded are available on the website indefinitely. This allows users & therapists to access old data to help understand progression.

2.2 Simple Interface
We want the interface of the Compliance Device to be as simple as possible for the patient as well as for the Physical Therapist. After each exercise session the Physical Therapist as well as the patient, will be able to log on to the web site; using their user name and password to access personal information. The benefit of having the data uploaded to the web site is that it will allow the user to track the progress of their heart rate before - during – and after finishing the exercise. The data will be saved on an excel spreadsheet table format which will make it easy to read and understood by both parties (PT and patient).

2.3 Internet Connection
In order for the Compliance Device to provide the user (the patient or the Physical Therapist) with the appropriate data transfer, it needs to be connected to the internet via Ethernet. The connection output transmits data to network for data storage and outputs to website for analysis from physical therapist to monitor all said sensor output. A server will interface to a website with custom graphical user interfaces (GUI) for the therapist and patients.

2.4 Automatic Notifications
To save time for the users, the information should be automatically updated in real time, every time after each exercise session is over. This will give the patient the freedom of manually keeping track of their progress, and providing the Physical Therapist with real data which will allow him/her to successfully study the performance of her patients.
2.5 Ergonomics & Safety
All electrical components need to be properly insulated to avoid electrocution of the patient. Materials used for holding the electrical components and sensors in place should be non-allergenic, non-corrosive, breathable, comfortable and flexible for use during exercise. The sensors should be easily wearable as well as comfortable and free flowing as to not restrict patient during exercise. All wiring shall be kept to a minimal amount. The sensors and Gumstix computer shall be designed to be lightweight and easily portable for ease of use for the patient.

2.6 Power Consumption
The power consumption of the Compliance Device should be as low as possible. The receiver of the Polar™ transmitter currently requires 5V for operation. It is powered by an internal 9V battery. The Gumstix machine requires connection to a standard 120V outlet via an AC adapter.

3.0 HARDWARE

3.1 Heart Rate Monitor (HRM)
A critical design requirement for the heart rate monitor for this project is that it provides continuous and accurate heart rate monitoring while the patient is working out on the UBE. In addition, the HRM needs to be reasonably priced, is easy to use, and is comfortable for the patient. The ideal HRM for this design should be able to output its readings in a format that the Gumstix Single Board Computer can interface. After conducting research on the available HRM products in the market, a sensor by the Vernier Company proved to have the superior specifications for our design requirements.

The Vernier Heart Rate Monitor
- A chest belt transmitter, which the user wears around their chest
- A heart rate receiver, which receives the heart rate data wirelessly from the transmitter and outputs an analog signal of each heart pulse

![Heart Rate Monitor and Receiver](image)

The heart rate receiver sends a voltage of about 3V when a heart signal is detected and about 0V when no heart signal is detected. To use this sensor in our design, we created a program that counts the 3V signals that are outputted and this computes the heart rate.
We connected the Heart Rate Receiver to the Gumstix’s General Purpose Input Output so we could capture the input from it.

3.2 Cadence Sensor (RPM)

![Attached Cadence Sensor](image)

Figure 3: Attached Cadence Sensor

In order to measure the cycling cadence (revolution per minute) we decided to use a magnetic hall-effect sensor for implementation. This sensor detects the dips in the electromagnetic field when the pedals are revolving. We can then calculate cycling cadence by counting the number of dips over a function of time. In implementing the cadence sensor, we make further use of the General Purpose Input and Output pin configuration. To analyze data, we created a program that counts the number of electromagnetic dips created by the Hall Effect sensor and uses the SQLite database to gather and arrange the data.

3.3 The Gumstix Computer

This device is manufactured by Gumstix and is used for data acquisition and signal processing to measure heart pulses and cycling cadence. The Gumstix is quite small in size and is located in an enclosure to protect it from external damage, within close proximity of the UBE.

![The Gumstix with Peripherals](image)

Figure 4: The Gumstix with Peripherals

3.3.1 Input Interface(s):

The Single Board Computer device needs to receive input from the cadence sensor and the HRM sensor. The two devices both utilize the General Purpose Input Output pin configuration provided by the Waysmall Stuart expansion board.
- The Cadence Sensor uses GPIO pin set as ‘output’ and ‘high’ (3.3V). It uses another GPIO pin set as input, and tied to ground. When the input transitions from low to high, a software interrupt (IRQ) occurs and is recorded.

- The Heart Rate monitor uses a GPIO pin set to input. When the HRM sends a detected heart pulse, the GPIO pin goes from low to high, and a time is recorded.

3.3.2 Output Interface(s):
The primary output of the Gumstix is the data being transmitted to destined server/host location. The Single Board Computer features Internet connection via Ethernet. An additional connection board was needed to achieve this optimal status. Hence, our design incorporated the NetMMC expansion board for data transmission and Ethernet capabilities. In the figure below,

![NetMMC Expansion Board(s)](image)

**Figure 5: NetMMC Expansion Board(s)**

The NetMMC expansion board requires the Gumstix platform for connectivity and provides the combination of one standard RJ45 jack for 10/100 Ethernet and one compact flash adapter.
3.4 The Interfacing of Major System Components

![Diagram of Health C² UBE components]

3.4.1 Operation
During cardiac rehabilitation exercises using the UBE, the patient will wear a chest belt heart rate sensor. The sensor will transmit a pulse to the heart rate monitor’s receiver for every beat detected, which is signaled as a 3-volt pulse.

In addition, to measure the RPM, a magnetic hall-effect sensor is physically mounted on the wheel of the UBE itself. During one complete revolution of the wheel, the sensor detects any displacement in the electromagnetic field caused by the rotating magnet.

3.4.2 Signal Processing/Data Transmission
The information from both sensors is processed and stored with the use of the Gumstix computer. The data is then sent via Ethernet to a server for viewing and analytical purposes by the Physical Therapist or client.

3.4.3 Power
The Gumstix computer, during full operation (@ 400MHz) draws less that 250mA of current on average. For our design, including Gumstix recommendation, we used a 4V power adapter to power the Gumstix Computer and connected peripherals. For the heart rate receiver, a dc source (batteries) was an ideal solution due size, portability, and the requirement of less outlet connections.
4.0 TECHNICAL AND SOFTWARE APPROACH

4.1 System Overview
The Gumstix SBC used in this project operates as an embedded XScale computer using the Linux operating system which periodically uploads data to another server. Data acquired from the UBE and patient through the sensors are collected by the Gumstix, processed, and stored in it. All of the files used on the Gumstix to acquire and process data are written as shell scripts or in the Python programming language. Python was our language of choice in performing data manipulation due to the ease at which many different operations could be coded.

Patients using the Gumstix do not need to own a personal computer. All communication between the patient and the physical therapist regarding exercise data can be done over the phone. Those patients that do own a computer, however, would be able to log into the server through our website and view their saved exercise data.

4.2 Data Collection and Manipulation
Before the Gumstix SBC can begin collecting and processing data from the sensors, all of the files necessary to perform these tasks must be initiated. This is done through a basic shell script called startup.sh. The startup.sh file sets the GPIO pins for the heart rate and RPM sensor and registers their IRQs by loading hrm_driver.ko and rpm_driver.ko. These two kernel files are written and compiled in C, and are used to wait for an IRQ from its corresponding pins of the Gumstix. Every time an IRQ is recorded, a string identifier is Startup.sh also calls the programs parse-messages.py and generate-and-send.sh, which are used for gathering, manipulating, and transferring the heart rate and RPM data.

As the sensors begin collecting data, the timestamps recorded for every heart beat and RPM are stored in the /var/log/messages file. The parse-messages.py file is written in the Python and initially reads through the /var/log/messages file looking for heart beat and RPM timestamps. For each timestamp found, the database table, capstone.db is updated with this data. The capstone.db file is a SQL table that holds the timestamps in microseconds as well as the counts per minute RPM and HRM. Once the patient has completed their exercise session, parse-messages.py deletes the /var/log/messages file in order to free up memory on the Gumstix.

The generate-and-send.sh file initially sleeps for ten minutes from the time when it was called. After the ten minutes, it generates a report from the capstone.db file and saves it as an Excel spreadsheet with the name “report-$(date-time).xls”. This is the final format of the patient’s exercise data. Generate-and-send.sh then FTPs the excel spreadsheet to a server and removes data from the capstone.db file to free up memory on the Gumstix. This process is repeated every 10 minutes until the patient’s exercise session is complete or the Gumstix computer is turned off. The block diagram in Figure # gives a clear picture of how these scripts interact in data collection and manipulation.
Gumstix turns on

/scripts/programs

/ect/init.d/Startup.sh is executed and calls /root/startup.sh

/root/startup.sh
- sets the GPIO pins
- registers IRQs by loading hrm_driver.ko and rmp_driver.ko
- starts parse-messages.py
- starts generate-and-send.sh

/root/parse-messages.py
- reads through /var/log/messages and updates the database table with the count for heart beats and RPMs
- deletes /var/log/messages when finished
- sleeps for 15 seconds, repeat

/root/generate-and-send.sh
- sleeps for X amount of time
- generates report from database contents
- FTP the report to the server
- removes the data from the gumstix database
- repeat

/kernel/modules

/root/hrm_driver.ko
A kernel module that prints a timestamp to /var/log/messages whenever GPIO pin 62 goes from low to high.

/root/rpm_driver.ko
A kernel module that prints a timestamp to /var/log/messages whenever GPIO pin 32 goes from low to high.

/data/manipulation

/root/capstone.db
- SQL table that holds timestamps in microseconds, counts per minute RPM, and counts per minute heart beats
- provides intermediate storage

/var/log/messages
- holds timestamps for RPM and heart rate as well as other important kernel messages
- intermediate storage

/root/report-$date-time.xls
- Excel spreadsheet generated from capstone.db
- final format of exercise data

Figure 7: Software Block Diagram
5.0 WEBSITE

The website is designed using Microsoft Visual Web developer 2005 Express edition. Our simple Web service is written in Visual Basic .NET, and uses Hypertext Transfer Protocol (HTTP) and Simple Object Access Protocol (SOAP) to carry messages back and forth between client and server. Unlike other Web-service capable programming environments,.NET does much of the work in the back end, meaning that you only have to write a few lines of code in order to create a working Web service.

In a .NET environment, the Web service host is Internet Information Services (IIS v5.0). The Web service listener is an ASP.NET page. ASP.net is a technology that allows programmers to create dynamic web applications. It can be used to create anything from small personal websites to large enterprise-class web applications. The service code can be developed in any .NET language. Aside from Visual Basic .NET, C# is another language used. The process of building a Web service in the .NET Framework can be divided into several components, starting with the Web Service Templates built into Visual Studio .NET.

To create a Web service in .NET, we open Visual Studio .NET, select New Project, and then choose the ASP.NET Web Service Template. Before clicking OK, we type a new name in the Location text box. This box creates a new directory structure under IIS for our Web service. In this example, we call our Web service website1, so the location textbox should be http://localhost/website1. Once we set the location value, Visual Studio .NET will automatically propagate the name through the rest of the class files, which makes it very difficult to change the name later.

Each template has two windows: A design window, and a source window. The Design window displays information as they will appear on the actual website while, The Source window is used to write and edit codes. Once the website is created, Visual Studio .NET starts at the Design window. Additional control such as a label, a table and a loginView can be dragged into the design page. The codes used to make these controls more functional are added on the source window. The page is built by pressing CTRL-f5.

In order to keep our website secure, we add a folder which we name Memberpages. To create a username, and password, we click on the security tab of the ASP.net configuration. Under permission, we select deny. This setting denies any user without login id and password the permission to access the records available in the webpage. For those who have login access, there is a link on the login page called Records. The revolution per minute (RPM), and heart rate variability (HRV) are available on this page.
6.0 BUDGET ANALYSIS
The total cost of goods to build the Initial prototype Physical Therapy Cadence Compliance for UBE Device is approximately $425.00. This includes the cost of the Gumstix Computer, Heart Rate Receiver, Polar Coded Transmitter, and high powered sigma magnet.

6.1 Initial Prototype
This prototype was designed to offer the most inexpensive approach for a physical therapist to monitor their patient’s progress. The most expensive component is the Gumstix Single Board Computer which contains an Intel processor (200/400MHz). In addition, the unit has 64 MB of SDRAM, 4MB of flash, it uses the Linux operating system and can be connected to using serial, USB, and wireless interfaces. This computer costs approximately $300.00 and serves as an efficient tool for signal acquisition, processing, storage, and data transmission due to its high performance, low power consumption and overall cost.

The next component is the polar coded transmitter which cost approximately $60.00. This Heart Monitor transmitter was chosen because the belt detects each heart contraction through two electrodes with Electrocardiogram accuracy and transmits the heart rate information to the wireless receiver with the help of a low frequency electromagnetic field. The Heart rate receiver then wirelessly receives the transmission, and sends a 3V pulse so it can be recorded. This device costs approximately $50.00.

The last device included in the Initial Prototype is the Sigma High Powered Magnet. This magnet is placed on the pedal, and the Hall-effect switch is placed on the body of the UBE. During one complete revolution of the pedal, the sensor detects any displacement in the electromagnetic field and an analog signal is detected. This magnet costs approximately $10.00, and it adds great value to the design by adding the ability to track revolutions per minute.

The Initial prototype for the Physical Therapy Cadence Compliance Device offers many features comparable to a Holter Monitor and costs approximately $1000-$1500 less. This would give significant space to manufacture the basic Health C-UBE while making a healthy profit.

6.2 Envisioned Final Product
The envisioned final product offers two additional features, and approximate price difference of $+250.00 making the unit material cost approximately $775.00. The first feature is to add a force transducer mounted between the spring, and the frame to measure the reaction force of the spring. The physical therapist can then use this resistance information to better specialize the patients workouts as they make progress. A Network Web Camera is implemented in this design to reassure that the patient follows the exercises as recommended from the Physical Therapist. The camera would easily mount on the bike, and take a picture every minute and then it will send it to a server via File Transfer Protocol (FTP). These additional features offer excellent improvements at an affordable cost.
7.0 FUTURE IMPROVEMENTS:
There are plenty of things that further improve this design. Increasing the speed between the software and hardware will yield much better and more accurate performance and calculations for the physical therapist. Some form of identity confirmation would also aid the physical therapist in knowing exactly who is using the UBE, and preventing falsified data. Other ideas include other data manipulations where the device may calculate BPM calculations, and plot R-R Intervals, as well as HR vs. RPM plots.

7.1 C/C++ vs. Python
Many regard C/C++ as the baseline for comparisons in execution speed. Python is very slow in comparison, and it is its sluggish speed that caused the device to occasionally skip over recording a beat. It is believed that re-writing the code in C/C++ will undoubtedly fix this problem making the UBE device more accurate in its time stamps [7].

Figure 8: Execution Speeds of Various Programming Languages

7.2 New Parsing Location
Currently, the system reads time stamps from var/log/messages. This is an inefficient manner in which the python file needs to search for timestamps given the large level of text generated by kernel messages. New drivers were written in C to have them be inserted into a new “log” file in var/temp/timestamps.txt. However, time restrictions did not allow for it to be proper implemented since parsing from a new log may introduce an additional step in “string slicing” when establishing the epoch time (or reference time).
7.3 UTC Time
The current system provides excel reports name as thedate.xls. However, the Gumstix computer did not sync to the UTC when it linked up to the internet and as a result reports were generated and dated near 1970; the Unix/Linux cut-off year.

7.4 New Improved Gumstix
Currently the Gumstix is running at 200MHz. A 400MHz version is available for only $40 or so dollars more. Future integration of this should yield more accurate data acquisitions similarly to how using a faster programming language would. This higher model has a comparable price.

7.5 Webcam Integration
Installing a webcam is definitely important in improving this design in the future. Physical therapists have complained that they are worried that some of their patients are falsifying their data and workouts by having someone else complete the workout regimens for them. The integration of a webcam, allowing the physical therapist to view who is using the machine, would alleviate this problem.

7.6 Resistance Measures
Future integration of devices that adjust and measure resistance for the UBE would definitely add to its value as a first class physical therapy device. This would allow the physical therapist to vary the workouts through a different aspect: resistance. A Vernier Spring Force sensor would easily be mounted on the UBE, and successfully accomplish this task.
Appendix A – Gumstix Coding

StartUp.Sh
#capstone controller script. runs everytime gumstix boots

cd /root

#kill any python and processes that might be around
#should kill bash too....but how to kill all bash processes
#but the current??
killall python # killall sends a signal to all processes running any of the specified commands.

#remove the drivers if they're already loaded
rmmod /root/rpm_driver.ko
rmmod /root/hrm_driver.ko

#remove any log messages
cat /dev/null > /var/log/messages #var/log/messages is removed and replaced with /dev/null
#rm /root/parse-messages-log
#rm /root/generate-and-send-log

#remove db contents
sqlite3 /root/capstone.db "delete from average_times" #why does the same command occur x3
sqlite3 /root/capstone.db "delete from rpm_times"
sqlite3 /root/capstone.db "delete from hrm_times"

###    RPM    ###
#set pin gpio30 to high and to output
echo "GPIO out set" > /proc/gpio/GPIO30

#set the other pin gpio32 to input
echo "GPIO in" > /proc/gpio/GPIO32

#register an interrupt for this pin
insmod /root/rpm_driver.ko

# HEART RATE SENSOR
#set the gpio62 pin to input
echo "GPIO in" > /proc/gpio/GPIO62

#register an interrupt for this pin
insmod hrm_driver.ko

#start the message-parser for /var/log/messages in background
/opt/python/bin/python parse-messages.py &

#start the generate-and-send report ---sends every 10 minutes to ftp site if there's a report
bash generate-and-send.sh 180 >> /root/status & #script files need bash before being called

-RPM_driver.C

/ * Hrm_driver.c - GPIO
 * /
```c
#include <linux/ini.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/interrupt.h>
#include <asm/arch/pxa-reg.h>
#include <asm/irq.h>
#include <asm/arch-pxa/irqs.h>
#include <asm/arch-pxa/hardware.h>
#include <linux/time.h>

MODULE_LICENSE("Dual BSD/GPL");

//pin 32 recieves for RPM
static int gpio = 32;

//function gets called everytime an irq is registered
static irqreturn_t my_handler(int irq, void *dev_id, struct pt_regs *){
    struct timeval timecount;
    do_gettimeofday(&timecount);
    printk("RPM: %i.%i\n", (int)xtime.tv_sec, (int)xtime.tv_nsec);

    return IRQ_HANDLED;
}

static int rpm_driver_init(void)
{
    pxa_gpio_mode(gpio | GPIO_IN);
    set_irq_type(IRQ_GPIO(gpio), IRQT_RISING);
    request_irq(IRQ_GPIO(gpio), my_handler, 0, "my handler", NULL);
    printk(KERN_ALERT "Hello, World\n");
    return 0;
}

static void rpm_driver_exit(void)
{
    printk(KERN_ALERT "Goodbye, cruel world\n");
    free_irq(IRQ_GPIO(gpio), NULL);
}

module_init(rpm_driver_init);
module_exit(rpm_driver_exit);

hrm_driver.c

/* Hrm_driver.c - GPIO */

#include <linux/ini.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/interrupt.h>
#include <asm/arch/pxa-reg.h>
#include <asm/irq.h>
#include <asm/arch-pxa/irqs.h>
```
// pin 62 receives "high" from Belt Sensor
static int gpio = 62;

// function gets called every time an irq is registered
static irqreturn_t my_handler(int irq, void *dev_id, struct pt_regs *){
    struct timeval timecount;
    do_gettimeofday(&timecount);
    printk("HRM: %i.#i\n", (int)xtime.tv_sec, (int)xtime.tv_nsec);
    return IRQ_HANDLED;
}

static int hrm_driver_init(void)
{
    pxa_gpio_mode(gpio | GPIO_IN);
    set_irq_type(IRQ_GPIO(gpio), IRQT_RISING);
    request_irq(IRQ_GPIO(gpio), my_handler, 0, "my handler", NULL);
    printk(KERN_ALERT "Hello, World\n");
    return 0;
}

static void hrm_driver_exit(void)
{
    printk(KERN_ALERT "Goodbye, cruel world\n");
    free_irq(IRQ_GPIO(gpio), NULL);
}

module_init(hrm_driver_init):
module_exit(hrm_driver_exit);

---

# parse-messages.py

# parses /var/log/messages

import os, sys, time

def does_time_exist():
    epoch_time = line[47:]; # i still dont know what line 47 refers to
    readable_time = time.strftime("%m:%d:%H:%M", time.gmtime(float(epoch_time)))
    does_time_exist = os.popen("sqlite3 /root/capstone.db 'SELECT time FROM average_times WHERE time =\"" + readable_time + \"\"'").readlines()
    try:
        temp = does_time_exist[0]
    except:
        print "insert"
        os.popen("sqlite3 /root/capstone.db 'INSERT INTO average_times VALUES(\"" + readable_time + \\", 0,0)\")"
    return readable_time
def getTime():
    t = time.time()
    return t

def getMicro(t):
    fract = float("0."+repr(t).split(".")[1])
    usecs = ":%i" % int( fract /(.000001) )
    return usecs

def time_stamp(t, usecs):
    # t = time.time()
    # fract = float("0."+repr(t).split(".")[1])
    # usecs = ":%i" % int( fract /(.000001) )
    readable_time = time.strftime("%m:%d:%H:%M:%S"+usecs, time.gmtime(t))
    time_stamp = os.popen("sqlite3 /root/capstone.db 'SELECT time FROM average_times WHERE time =\"" + readable_time + "\"')
    try:
        temp = time_stamp[0]
    except:
        print "insert"
    os.popen("sqlite3 /root/capstone.db 'INSERT INTO average_times VALUES(" + readable_time + ", 0,0)'")
    return readable_time

#hrv1 = 0
while(1):
    try:
        infile = open("/var/log/messages");
        for line in infile:
            if "RPM:" in line:
                readable_time = time_stamp()
                readable_time_seconds = time.strftime("%m:%d:%H:%M:%S", time.gmtime(float(line[47:])))
            sql = "UPDATE average_times set rpm = (SELECT rpm FROM average_times WHERE time=" + readable_time + ") +1 WHERE time =" + readable_time + "; INSERT INTO rpm_times VALUES( " + readable_time + ", 0,0)"
            print sql
            os.popen("sqlite3 /root/capstone.db " + sql)
    elif "HRM:" in line:
        t = getTime()
        usecs = getMicro(t)
        readable_time = time_stamp(t, usecs)
        readable_time_seconds = time.strftime("%m:%d:%H:%M:%S"+usecs, time.gmtime(float(line[47:])))
    # difference = t - hrv1
    # time_stamp()
    # sql = "UPDATE average_times set hrm = (SELECT hrm FROM average_times WHERE time=" + readable_time + ") +1 WHERE time =" + readable_time + "; INSERT INTO hrm_times VALUES( " + readable_time_seconds + ", 0,0)"
    # hrv = "INSERT INTO HRV_times Values( " + difference + ", 0,0)"
    # print hrv
    os.popen("sqlite3 /root/capstone.db " + sql)
    # os.popen("sqlite3 /root/capstone.db " + hrv)
# hrv1 = t
# clear this file!
os.popen("cat /dev/null > /var/log/messages") # Replaces messages with a null file instead of deleting it

# sleep for awhile
# print "sleeping for .2 sec"
time.sleep(.2)

except KeyboardInterrupt:
    # continue
    print "ok"
    raise

Generate-And-Send.sh

## Generate-and-Send.py
while true; do

    # sleep before sending the report
echo "sleeping for $1 seconds before sending report"
sleep $1

    # generate times report from db
    sqlite3 -separator "    " capstone.db "SELECT * FROM average_times ORDER BY time" > times.txt
    timescount=`sqlite3 -separator "    " capstone.db "SELECT count(*) FROM average_times"
    echo $timescount

    # should it send or not?
    if [ "$timescount" = 0 ];
        then
echo "count 0"
    else
        echo "SENDING!"
    fi

    time" >> times.txt

    # make the report from the template and times
    thedate=`date +%Y-%m-%d-%H-%M` cat template.txt times.txt > "report-$thedate.xls"
    echo "generated report-$thedate.xls"
    if [ $timescount != 0 ]
    then
        /opt/python/bin/python nftp.py eceserver "report-$thedate.xls"
echo $?
        echo "sending report-$thedate.xls"
    else
        echo "not sending, no data"
    fi

    echo "removing DB contents from gumstix that was sent, and remove report file"
sqlite3 capstone.db "delete from average_times"
sqlite3 capstone.db "delete from rpm_times"
sqlite3 capstone.db "delete from hrm_times"
rm "report-$thedate.xls"
#!/usr/bin/env python2.3

CreateTables.Sql

Createtables.sql
CREATE TABLE average_times(
time varchar(100) default NULL,
rpm int(50) default NULL,
hrm int(50) default NULL);

CREATE TABLE rpm_times ( 
time varchar(100) default NULL );

CREATE TABLE hrm_times ( 
time varchar(100) default NULL );

template.txt
Heart rate and RPM Report
Capstone Group 6
Date/Time RPM Heart Rate

S98ntpd/n #sets the system clock using ntpdate
#!/bin/sh
#
# Set the system clock
#

start(){
    if( test -x /usr/bin/ntpdate )
        then
            echo "Setting the clock from the timeserver..."
            /bin/sleep 5
            /usr/bin/ntpdate ntp.ccs.neu.edu
        fi
}

stop(){
    echo ""
}

restart(){
    stop
    start
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart|reload)
restart
::
*)
  echo $#Usage: $0 {start|stop|restart}"
  exit 1
esac

exit $?
Appendix B – Webpage code

login.aspx <%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
    Protected Sub Login1_Authenticate(ByVal sender As Object, ByVal e As System.Web.UI.WebControls.AuthenticateEventArgs)
        End Sub
    </script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
<body bgcolor="teal"></body>
<form id="form1" runat="server">
<div>
</div>
</form>
</body>
</html>

<%@ MasterType virtualpath="~/Master1.master" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server"></asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2" runat="Server">
<h1>&nbsp;<asp:Label ID="CompanyName" runat="server" Text="Label"></asp:Label> Web site.</h1>
<p><asp:LoginView ID="LoginView1" runat="server" BackColor="LightBlue"></asp:LoginView>

</p>
</asp:Content>
</asp:Content>
Here you will find a list of data awaiting your review. Please click on the link below to access different records.

<a href="http://www.ece.neu.edu/~atribole/index.php?dir=reports/">Records!</a>

Our team designed and built a device to monitor and evaluate compliance of cardiac rehabilitation exercises for amputees and patients with myelopathy. This device is specifically designed in conjunction with a common exercise machine for our targeted demographic: an Upper Body Ergometer (UBE). The device obtains data such as heart rate variability (HRV), revolution per minute (RPM), and the time spent on the UBE, and sends it through a secure connection for further analysis by the therapist. In order to protect patients' confidentiality, all users must sign in.

Please use the tag below to sign in:

Thank you for visiting our site.
Physical Therapy Compliance Device Database

Our team designed and built a device to monitor and evaluate compliance of cardiac rehabilitation exercises for amputees and patients with myelopathy. This device is specifically designed in conjunction with a common exercise machine for our targeted demographic: an Upper Body Ergometer (UBE).

During different exercises, the device will obtain data such as heart rate variability (HRV), revolution per minute (RPM), and the time spent on the UBE. The data will be sent through a secure connection for further analysis by the therapist.
The website was constructed to help therapists access their patients' record during exercise on the Upper Body Ergometer.<br />
In order to protect patients' confidentiality, therapists are required to sign in.<br />
Please use the tab below to sign in:
References

Articles & Journals:

Product Specifications:

Reports
Sample reports are downloadable at www.ece.neu.edu/~atribole