UNIVERSITY OF WISCONSIN – MADISON

POGO

Pulse oximetry, on the go

Department of Biomedical Engineering

Advisor: Dr. Amit Nimunkar
Client: Dr. Fred Robertson

Chris Fernandez (Team Leader)
Olivia Rice (Team Communicator)
Rafi Sufi (BSAC)
Nick Glattard (BWIG/BPAG)

[10/09/2013]

Abstract

A pulse oximeter is a device that allows for the observation of basic respiratory function through a noninvasive measurement of oxygen saturation in arterial blood. Patients with chronic diseases such as high risk congestive heart failure, chronic obstructive pulmonary disease, or severe asthma do not require continuous monitoring in a hospital, however, could greatly benefit from periodic oxygen saturation monitoring to determine their state of health. Effective blood oxygen monitoring is challenging in the home because current pulse oximetric technologies typically have a limited data collection range, require patient cooperation and do not automatically share vital data with physicians. On a high level, the patient wears the POGO device, which automatically collects their oxygen saturation reading and then transmits the data to a Xively server. This data is then pulled from this server, uploaded and stored on a Heroku cloud database where physicians can access it through the Django web framework and bootstrap interface. Eventually, it is desirable to replace Xively with our own TCP server. Physicians will be able to access this website from their cell phone, personal computer or other configured device. Down the road, it may be possible to apply this type of data transmission to other key health indicators such as ECG, blood sugar level and temperature. Ultimately this device could improve a patient's quality of life and provide insight to trends regarding the decline of health in patients with various diseases.

Table of Contents

PROBLEM STATEMENT 3
BACKGROUND 3
DESIGN MOTIVATION 6
CLIENT REQUIREMENTS 7
DESIGN CONSTRAINTS 7
DESIGN ALTERNATIVES 8
DECISION MATRIX & DISCUSSION 9
FINAL DESIGN 10
TESTING 13
FUTURE WORK 15
REFERENCES 18
APPENDIX 19

1.0 Problem Statement

Many patients affected by chronic diseases such as congestive heart failure, chronic obstructive pulmonary disease, or severe asthma would greatly benefit from frequent blood oxygen saturation monitoring by a physician. Effective monitoring is challenging in a home setting because current technologies typically have a limited data transmission range, require patient compliance and do not automatically share vital data with physicians. This POGO device will automatically collect and transmit patient's blood oxygen saturation data from any location with 2G cellular coverage, to one or more physicians simultaneously through the website application we intend to design. Last semester the POGO device was prototyped and tested. Although the device met key client specifications, an application and user interface is needed for physicians to view patient oximetry data in real time. The device improves patient quality of life by providing freedom of mobility, a hands-free lifestyle and has the potential to reduce hospital readmissions and emergency room visits.

2.0 Background

Pulse oximeters measure arterial blood oxygen saturation. The oximeter sensor contains two LED lights on one side and a photo detector on the other, passing two different wavelengths of light through the patient's skin. For adult patients, sensors are typically placed on the fingertip or the ear lobe.[1] Red (660 nm) and infrared (940 nm) light is cast through the skin and is absorbed by hemoglobin in the bloodstream. The LED lights typically flash in an alternating fashion so a single photo detector is able to measure each of the light intensity levels. With a known absorption of both infrared and red light, a ratio of oxygenated hemoglobin to deoxygenated hemoglobin can be calculated. This ratio is what is commonly referred to as a patient's oxygen saturation reading (SpO_2).[2] The raw data collected by the photodetector is integrated and can be displayed both graphically and numerically as a plethysmograph and numerical reading accordingly.

Pulse oximetry is an essential practice for physicians to assess a patient's health. Studies have shown that a patient with oxygen saturation levels below 90% for an extended amount of time can lead to an increased chance of mortality. When considering health and safety of the patient, it is highly recommended that they be admitted to the hospital when oxygen saturation levels have declined to 93%.[3] However, in an at home setting, patients without constant monitoring are often unaware that their oxygen saturation levels have declined past this threshold. Eventually the patient will become severely deprived of oxygen and in the case of CHF, will start to experience heart failure and be admitted to the emergency room or possibly die. This high-risk situation could be avoided if physicians were aware of their patient's SpO_2 levels and could make an early intervention decision.

2.1 Previous Work

Last semester, a thorough patent search was conducted to see if anyone had acquired space in automatic SpO_2 transmission over the cellular network. However, there was no such patent that indicated this form of communication. The oximeters currently on the market for a home care setting either don't transmit data, use Bluetooth for data transmission to a device nearby, or are directly connected to an alternate device that is used for the transmission of data such as a cellular phone or computer. The Pogo is different from all current technologies because it is a stand-alone device that requires minimal patient compliance. The patient simply has to wear the device and their SpO_2 readings will be automatically collected and transmitted to their physician. There are no range limitations as there is with Bluetooth and there is no dependence on the functionality and battery life of other devices such as a cellular phone. In Figure 1 is a photo of the device we created last semester. This device has the capability to collect a signal from the oximeter sensor, and send it over the cellular network to a TCP server called Xively.

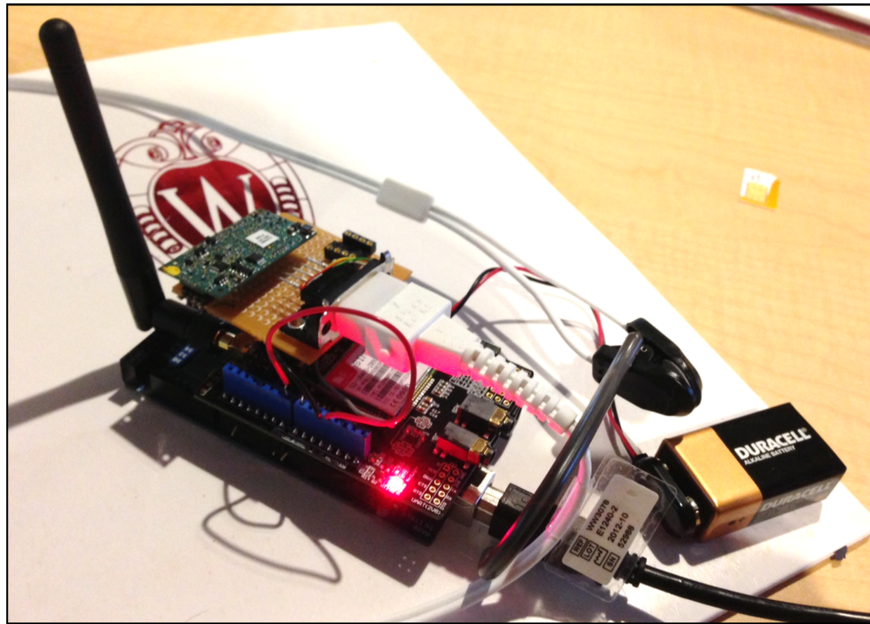

Figure 1: Pogo device complete with an oximeter sensor, BCI OEM board, Arduino microcontroller and GSM shield.

A power usage test was conducted as well as a test of time required for the device to change state from completely off to being able to process commands that initialize data transmission. Figure 2 shows the results of the power consumption tests. It is shown that power does not exceed 630 mW for extended periods of time. This low power consumption level indicates safe temperatures for patients.

All prototype components connected at 4.5 Volts	Current (mA)	Power (mW)
GSM off and Arduino idle	69	310
GSM connecting/ sending	140	630
GSM module on and idle	120	540
Average if sending once every 15 mins with Arduino power on	75	340
Average if sending every 15 minutes with complete power down between transmissions	12	68

Figure 2: Shows a comparison of current in various modes of the prototype.

It was also shown that it took 16.1 seconds on average for the device to change state with a standard deviation of 1.21 seconds. The time allotment before commands were sent was set to 19.7. This allowed for three standard deviations of the mean to be covered which will allow for 99.7% of the attempts to upload data to be received correctly by the GSM Shell if a normal distribution is assumed. Using this 19.7 seconds we see the average current draw is 135 mA over the course of the 80 seconds between power on and power off. If we average this over 15 minutes with the system powered off in between each packet of data sent, the average current over time is 12 mA. With alkaline AA batteries with a battery life of 2700 mAh, this leads to a conservative approximation of a weeklong battery life.

The team attempted to show that the SpO_2 and pulse rate values calculated by the OEM system (Smith's Medical BCI) were not corrupted during communication to the host device (Arduino). The Arduino was receiving a signal from the BCI processor however, the BCI was sending a consistent flag indication "No finger present in sensor" whether or not a finger was placed appropriately in the sensor and therefore no accurate data was collected. This issue is being addressed this semester and further testing will result.

3.0 Design Motivation

The data transmission process between BCI, Arduino and GSM Shield has been tested and verified last semester. However without a front end system to view, store and analyze this data, the system is rendered useless. The front end system will allow the user to access patient data stored in a database as well as real-time data visualization in an easy to access user interface. With the use of a front end system, the physician will be able to remotely monitor their patients' oxygen saturation and other biometrics at any time. The interface will also allow the physician to set critical thresholds for each patient and set alarms to notify caregivers if thresholds have been surpassed. This level of intervention can categorize the Pogo as a telehealth technology.

Telehealth technologies are emerging in healthcare markets because of their ability to reduce healthcare costs as well as provide prevention techniques to enhance the quality of care. Telehealth technologies can be described as the use of electronic information and telecommunications technologies to support long distance clinical health care[4]. A study conducted in 2008 by the Department of Health in the UK focused on health care costs between patients who were treated with standardized health care equipment and patients who were treated with telehealth technologies. The study concluded that there were reductions across various cost categories as well as a 45% decrease in mortality rate (Figure 3).

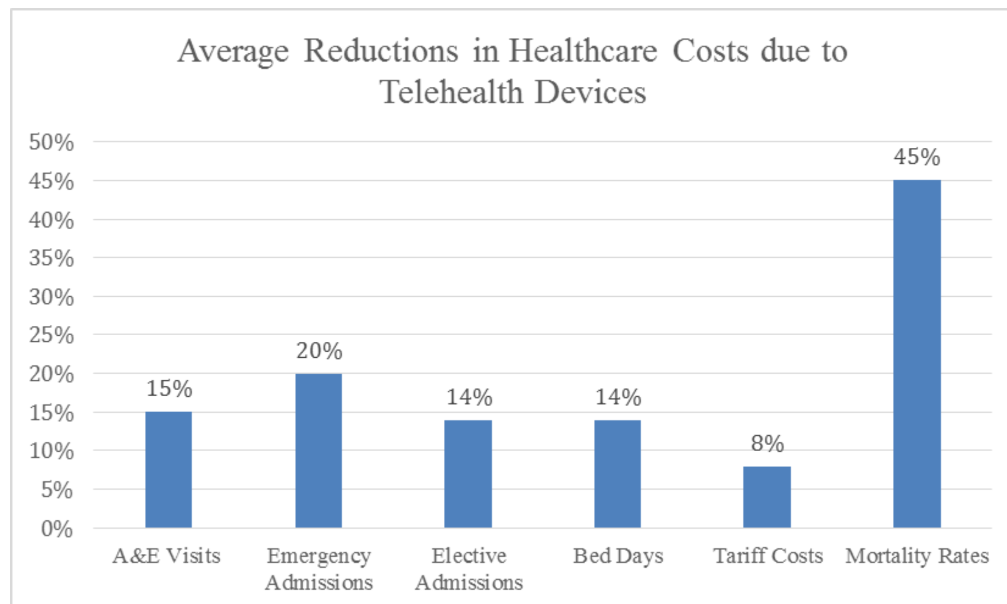

Figure 3: Department of Health study showing various reductions in healthcare costs.

In addition to proving cost reductions are attributed to utilizing telehealth devices, the study also states at least three million people who suffer from long-term conditions such as heart

failure, chronic obstructive pulmonary disorder (COPD) could benefit from using telehealth technology[5]. The Pogo, termed as a telehealth device, will be able to reduce costs and improve healthcare intervention techniques.

4.0 Client Requirements

The main client requirement described for this semester is to create a web development framework to be able to store and view real-time patient data that is being transferred from the pulse oximeter. This web development framework will need to feature several key components. First, a method of retrieving the data from the GSM shield is needed. Next, a method of storing the data is required in order to handle multiple patient data and view recent trends of biometrics. Finally, a method for viewing and interacting with patient data is needed, which will be user interface.

In terms of content, the client specified several key components that a physician needs on the web application in order to accurately assess the patients health. Listed below are the initial criteria needed to be displayed.

- A method to retrieve the patient identifier number which allows the physician to access medical records
- The ability to view the patient's biometric trends within the past month
- Most recent waveforms of heart rate, oxygen saturation
- User adjusted alarm set points
    - Alarm notification to caregiver
- Storage for patient data
- Signal quality indicator to indicate accuracy of data

A more refined criteria list will be developed by performing contextual interviews with physicians at the ambulatory wing of the UW Hospital.

5.0 Design Constraints

The website being designed must display all of the relevant data collected by the BCI sensor (blood oxygen level, heart rate, plethysmograph, etc) for each patient in a user-friendly interface. The data should be grouped by patient and should be accompanied by additional patient information such as a picture and contact information. The client should have the ability to set the real-time measurement thresholds for when they should be alerted, who should be alerted (patient, significant other, doctor), and how they should be alerted (text message, email, pager). Since the data being transmitted and received is private health information, it must be encrypted and clients must enter the site through a secure login page.

At this point in the project, the current design sends information through the Xively and Heroku databases. These services charge once our device uses more than the allotted memory, which the team would like to avoid. This really restricts the scalability of our project in the long run and avoiding these issues will be discussed further in the future work section.

6.0 Design Alternatives

To address the clients design specifications, the potential effectiveness of the *Pogo* patient data visualization program was evaluated on three different platforms; as a desktop computer application, a mobile application, or a website application. Each alternative would require a different application workflow process, set of development tools, and testing protocols. Moreover, each alternative impacts important application usability factors including where, when, and how often physicians are able to use this tool to supplement clinical decisions.

6.1 Alternatives Defined

Desktop Application: The desktop application alternative is defined as a program that is native to a single laptop or desktop computer, and is reliant on WiFi connectivity to receive and update patient information.

Mobile Application: As a design alternative, the mobile application is an 'App' that is native to a single tablet or smartphone. Depending on the specific hardware used, this alternative can be compatible with WiFi and Cellular wireless networks to receive and update data.

Website Application: The final design alternative is the website application. This is defined as a program that is hosted on web servers independent from any end-user device. Therefore, this application could be accessed through a web browser on any web compatible device.

6.2 Evaluation Criteria

In order to objectively assess which of the three potential alternatives would best suit the client and end-users needs, each alternative's efficacy was analyzed across 7 criteria. The two criteria deemed most critical are ease of use by physicians and reliability of the application. Ease of use encompasses the factors that determine the convenience to use the app. In order for this tool to be used effectively, the design must promote frequent enough utilization that patient trends can be recognized. Reliability relates to the probability of errors occurring on the end-user hardware or earlier the process of transferring information.

The next highest tier of criteria includes cross-platform portability and the team's ability to continuously deploy software updates. The applications' cross-platform portability depends on the end-user devices native operating system and preferred web client. Furthermore, the team sees value in having the ability to deploy updates to the program functionality quickly and

iteratively. The third and final tier of criteria encompasses the ease of application development, safety, and cost to the end-user. In this context, safety is defined as the likelihood that an end-user hardware error hinders a physician's ability to make a real-time intervention on a patient. Although this safety is a central part of the team's vision, the team is unable to take responsibility for patient care beyond creating the best possible tools for intervention.

7.0 Decision Matrix & Discussion

Alternatives & Criteria	Weight	Desktop Application		Mobile Application		Website Application	
Cost	10	5	10	5	10	3	6
Safty	10	2	4	4	8	5	10
Ease of Use	20	2	8	3	12	5	20
Continuous Deployment	15	3	9	4	12	5	15
Reliability	20	4	16	3	12	5	20
Portability (cross-platform)	15	1	3	1	3	4	12
Ease of Development	10	2	4	2	4	4	8
Total	**100**		**54**		**61**		**91**

Figure 4. The web application decision matrix showing the design alternatives as columns and evaluation criteria as rows. Category winners are shown in green, with second place shown in yellow, and third in red.

Following the analysis of each design alternative, the website application was found to be the best design by a significant margin. The web application was determined to be the leading design in the two most important evaluation criteria; ease of use and reliability. It was superior the desktop and mobile apps in terms of ease of use because it is interoperable across both of those platforms. In the case of the desktop, physicians must be beside their computer and have WiFi access, and in the mobile format they would be unable to view patient data on a desktop computer. Furthermore, in any cases where a physician's tablet, phone, or laptop was to die, their ability to monitor patients would also fail if they are using a native desktop or mobile app respectively. Because the web app could be accessed from any other device after authentication in the case of a failure, the web app was ranked first in the reliability category as well.

In addition, the web application was also shown to be the superior alternative in the second tier categories; cross-platform portability and continuous deployment. If the app was developed for a desktop or laptop computer, compatibility issues would arise unless versions of the app were created for each of the common operating systems; Windows, OSX, and Linux. Similarly, a mobile or tablet application would need to be compatible with numerous mobile operating systems, including Android, iOS, and Windows. In the case of the website application, the URL could be accessed from any operating system. Moreover, there are web development tools available which render a website compatible with all major browsers, including Google Chrome, Mozilla Firefox, Internet Explorer, and Safari. Next, the web application is the only of the three alternatives whose software updates capabilities rest fully in the hands of the design

team. This is advantageous because it enables the most flexible, iterative software updating and improvement process possible, and simplifies the user experience for the physicians, as they do not have to engage in that process actively.

Finally, the web application was found to be the best alternative for two of the three third tier criteria; cost, ease of development, and safety. The desktop and mobile applications were both more cost efficient solutions than the web application because of the increased server architecture that would be necessary to manage all of the computation, data, and users on a pure website structure. The web application, however, benefits from the greatest flexibility in terms of tools, tutorials, and open source code that is available to aide in accelerating the application development. The mobile application development would be restricted to the native devices development program, for example the iOS development center SDK. Furthermore, desktop applications face the greatest complexity in building a full program from scratch. Finally, the web application has the best safety features because of its flexibility and portability across numerous devices and operating systems. In cases where physicians receive critical health notifications from the *Pogo* device, the response time can be expedited the most by this broad flexibility.

8.0 Final Design

8.1 Signal Acquisition and Processing (The POGO Device)

The device workflow begins with a BCI pulse oximetry ear sensor. This sensor transmits red and infrared light through the earlobe tissues into a photodetector. A BCI digital micro pulse oximeter board operates the sensor and receives the sensor output. The board processes the signal and separates time invariant parameters (tissue thickness, skin color, light intensity, and venous blood) from the time variant parameters (arterial volume and %SpO_2) to identify the pulse rate and calculate oxygen saturation. Oxygen saturation calculations are possible because oxygen saturated blood predictably absorbs less red light than oxygen depleted blood. Data packets, consisting of 11, 8-bit bytes, are created and output at a rate of 60 packets per second. Each data packet contains the following encoded measurements: a real time plethysmogram waveform, %SpO_2, pulse rate, a real time bar graph that indicates if the sensor is properly attached, and signal strength metrics. Another useful metric embedded in each data packet is a flag that triggers when the %SpO_2 or pulse rate are above or below programmed threshold levels. The physiological measurements are sent to the host device through a serial port.

The host device we are using for our prototype is the Arduino Mega 2650. We have programmed the sensor acquisition algorithm to obtain the physiological data using a series of bitwise logic operations. The team also programmed the Arduino Mega 2560 microcontroller board to operate in one of two modes. The first mode is debug and testing mode. In this mode, the user can use a terminal on their PC to change the mode of the BCI sensor processor, request data from the sensor processor as well as send any AT-command to the GSM Shell to change its

settings, connect to a website, send an SMS message, or request GSM Shell storage data. This communication can be accomplished by attaching a PC with the Arduino software to the Arduino Mega with a USB. The second mode places the Arduino Mega into a state where it continually cycles through a power-on, sensor data collection, securely connecting to the Xively website for data sending through a series of AT commands (refer to Appendix C), powering down, and remaining idle for 13.5 minutes. The device is powered by either a 9-volt battery or three 1.5-volt batteries, allowing it to be completely mobile.

8.2 Xively/PostgreSQL Database

Patient data is sent from the POGO to the Xively cloud platform as a service (PAAS) through a GSM websocket embedded in the Arduino code. This transmission is secure for two reasons. First, no identifiable patient data is transmitted and received by Xively, only an anonymously generated patient ID. Next, the transmission is only receivable by Xively because a secure API access key provided by Xively must be verified to initiate the transfer of data.

This patient data is then permanently stored in Heroku, a cloud database PAAS the Heroku database is implemented in PostgreSQL, a powerful system for managing object-relational databases. The data fields shown on the left of figure 5 are received by the Heroku PostgreSQL database when a Python script requests data updates from the open Xively API. Using the patient ID as an identifier, this data table is then related to a separately partitioned data table containing all of the relevant patient and physician contact information. This database partitioning is valuable because it enables the minimization of data redundancy, and correspondingly maximizes the possible efficiency for searching and querying the physiological data. Once data is available, it can be accessed for viewing and analysis through the Django framework.

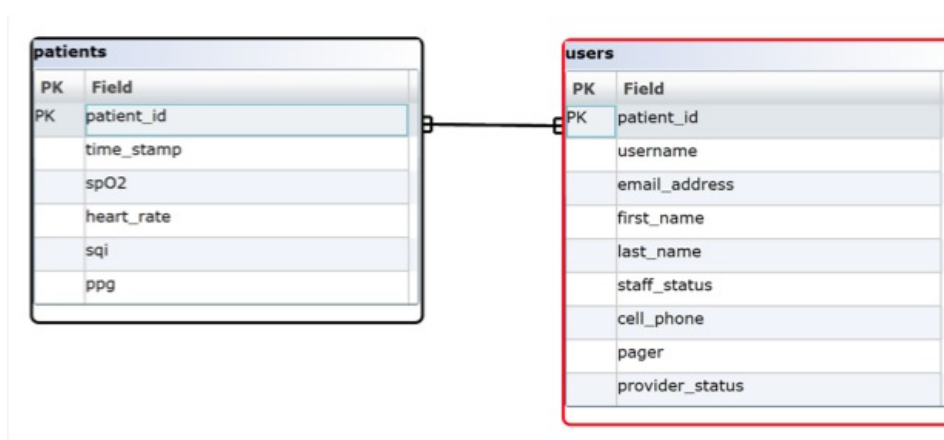

Figure 5. The left data table shows the data fields collected and transmitted from the POGO device. The data table on the right contains relational information and is partitioned separately from the physiological data.

8.3 Django Web Framework

Django is python-based web framework designed for developing database-driven websites. It acts as an intermediary between the web client and the database to perform data and http response and requests. Since it is built on python code, the framework is highly flexible and is supported by all platforms (Mac OS X, Linux, and Windows) and is used widely across the corporate atmosphere (Microsoft, Google, Sun).

The Django framework is built upon the Model-View-Controller (MVC) software architecture design. However it is slightly modified to a Model-View-Template design[6]. The theory behind this software design is to allow strict separation between each layer of the framework (Model, View and Template), making it easier to make changes in one layer without affecting the others. Each layer has a specific function which contributes to the web frameworks overall purpose.

The model layer is the data access portion. This layer is responsible for the data of the project. The layer extracts the data from the database, validates it, and decides what are the behaviors and relationships between the data sets[6]. In terms of our design, the data layer will interface with the PostgreSQL database to extract information.

The template layer is the presentation layer. The data accessed from the data layer is in raw format and the template layer decides how the data showed be displayed and the format in which it appears. The layer possesses syntax similar to HTML in order to accommodate web designers[6].

The view layer is the business logic layer. View is responsible for accessing the data from the model layer and attributes the appropriate template for display. In other words, the view layers acts as an intermediary between the model and the template layer. The view layer receives an http request from the web client for example, the user clicking on a button to view a patient's heart rate, and sends the appropriate http response, in this case the heart rate.

8.4 Bootstrap Interface

Bootstrap is a framework that incorporates HTML and CSS-based templates for buttons, forms, typography, navigation and other various user interface components of a webpage. There are also JavaScript extensions that are compatible as well. Bootstrap is a completely free collection of tools for creating web applications and websites. So far a basic web layout has been designed. Seen in figure 6 below is the website that has been generated. This is stored on a personal computer server and is therefore not yet accessible via the Internet.

Figure 6: A basic website design with colors and various typography made possible through Bootstrap.

9.0 Testing

9.1 POGO Device

The POGO tests must verify that the device can accurately measure the patient's blood oxygen data and send it via the GSM network to Xively. First, is to test that the BCI pulse oximetry processing chip is working and that the Arduino can parse the information being received. The procedure is as follows:

Pulse Oximetry Accuracy Test

1. Attach both the BCI pulse oximetry sensor and a separate pulse oximeter to your body

2. Set Arduino to debug mode
3. Run program to retrieve data from pulse oximeter chip
4. Output values retrieved in the debug window
5. Values should be as follows:
    a. Real time plethysmogram waveform
    b. 8 beat %SpO_2 average: 0-99%
    c. 8 second pulse rate average: 30-254 bpm
    d. Signal quality indicator
    e. Flag triggers when pulse signal is below threshold level
6. Compare values with secondary pulse oximeter
7. If values are the same, then passes the test

The second test would be to send this pulse oximetry information through the GSM Shield, through the GSM network, to Xively. The procedure is as follows:

GSM Shield Test

1. Set Arduino to debug mode
2. Run program to retrieve data from pulse oximeter chip
3. Record data recieved
4. Run program to send data through the GSM Shield to Xively
5. Check Xively to see if the information shows up in the database
6. If the information is there, check if it is the same information that was sent
7. If the data sent was the same as the data received, then passes the test

9.2 MVT Functionality

Testing is also needed to verify that the Django web framework is appropriating the right data structures to the specified parts of the web page. The initial test will be a simple *runserver* command to check if the development server is operating. The procedure is as follows:

Development Server Verification Test

1. Exit out of all applications within the terminal.
2. Run python, verify version number 2.7.5
3. Run the command prompt **python manage.py runserver**
4. If there are any errors validating models, go back and check file directories within python and make sure **mysite.settings** exists.
5. If the command successfully executes, navigate to **http://127.0.0.1:8000/** in web browser to validate server operation.

This test will verify that all of the Django directories and files are correctly structured, with the settings.py file inside the mysite folder. The second test is to verify the view layer is able to

send and receive http requests/responses. The associated code will be italicized for further clarification.

MVT Transfer Test – http request/response

1. Within Django, create an application named oximeter.
    a. *python manage.py startapp oximeter*
2. Make sure the necessary files are located within the oximeter application. Edit the models.py file to include the necessary code
    a. *from django.http import HttpResponse*
       def home(request):
       return HttpResponse("Hello BME 400 Team and The Great Amit")
3. Now edit the settings.py file under mysite and enter in database configurations and add the app 'oximeter' to the list of installed apps.
    a. '*django.db.backends.sqlite3'* (Using the sqlite server)
4. Run the command prompt *python manage.py runserver* to run the development server and navigate to **http://127.0.0.1:8000/**
5. Verify that the view layer returned the http response "Hello BME 400 Team and The Great Amit".

10.0 Future Work

The POGO team has reached many significant milestones throughout the lifecycle of product development, but many more lay ahead prior to real world implementation of the POGO solution. Specific to the POGO device and the web application, clear incremental product development steps are outlined in the subsections below.

10.1 POGO

The first objective of future work is to resolve issues with gathering accurate data using the current pulse oximetry setup. The sensor and processing chip are able to get a signal, but that signal only defaults to a bit pattern that means the sensor is not connected. This is the final hurdle to a fully functional POGO device.

Once the POGO device can accurately send sensing data to the network, the next steps would be miniaturization for patient comfort/portability and power usage efficiency for battery life. In terms of miniaturization, the next step is to implement a circuit board that would replace the Arduino and all of its functions since the Arduino is used for ease of development, not implementation. This benefits the project in other ways as well, as miniaturizing our POGO processing system will also minimize energy consumption. Encapsulating the device in a durable and waterproof casing would protect the device from normal wear and tear.

Even farther down the line, once the real-time data visualization website up and running, it would be beneficial to look into adding different sensors that would then also send their data through the GSM network to be viewed by our clients. In theory, the user would be able to hook up any portable sensor as long as we had the correct processing chips and decoding algorithms.

10.2 Heroku

The first step in preparing the Heroku database for deployment is the development of a Python script to request data updates through the open Xively API. After this data transfer is validated, PostgreSQL database partitions should be optimized and their implementation tested by verifying that data transmitted from the *Pogo* is properly stored and assigned to the correct table values. In the long term, table parameters can be added to support other physiological signals including ECG, EEG, EMG, blood pressure, blood glucose, weight, and temperature to name a few. This would also involve the implementation of Python scripts that are able to recognize what types of data have been received through a key value, and that the different data types are stored separately and with accuracy.

10.3 Django Web Framework

The next steps in regards to the Django development is to interface the framework with a PostgreSQL database. As of now, the framework is setup with the pre-installed SQlite database which has a limited capacity of data storage as opposed to PostgreSQL. With this data integration, patient data can be generated and transmitted to the web client. The Django framework also needs to be connected to the web site. This can be done by configuring the IP settings along with specific port number of the website. Also the data will need to be displayed in the correct manner, which means the template layer also needs to be addressed. The data structure as well as representation will be coded here after database integration.

10.4 Bootstrap

Before the webpage layout is fully designed, physician input is necessary in order to create a desirable end product. Soon, physicians will be met with and potential layouts and variations of the user interface will be established. In order to receive the most valid, meaningful input, ideally physicians of all ages, including medical students will be surveyed. Their feedback will be qualitatively analyzed and their input will determine the final layout of the webpage. Configuration and meshing with the Django framework will be the next priority. Specifically coordinating with the “View” segment of Django complete with sending data and presentation layout requests and processing data and layout responses will be necessary for full functionality of the website. Ultimately, the patient’s data collected in real time will be available for physicians through the webpage designed with Bootstrap.

10.5 Conclusion

In the bigger picture, the POGO team has additional goals that can greatly maximize the impact of this device and solution. First, an institutional review board application must be compiled and filed prior to launching a pilot test in collaboration with the UW Department of Anesthesiology. Next, FDA 510K class II approval is needed in order to bring the device to market or facilitate any commercial distribution. This process involves completing a 24 page acceptance checklist which demonstrates substantial equivalence to predicate ear oximeter devices. Finally, the team is also working in partnership with the UW Law & Entrepreneurship clinic to secure intellectual property on the overall device and working process. Once this is in place, the team plans to participate in the Qualcomm Wireless Innovation Prize, the Burrell Business Plan, the Tong Design Awards, and the Perkins Coie innovative minds competitions.

In conclusion, the team is strongly motivated to finalize a working prototype, because the POGO device has the potential to positively impact the lives of thousands of chronically ill patients around the globe. By enabling continuous remote monitoring through unparalleled convenience, a detailed longitudinal data set can be aggregated on the aforementioned patients. With more quantitative information than ever, analysis can be performed to better understand the progression of these diseases, and even to uncover previously unknown indicators of oncoming cardiovascular decompensation.

References

1. Oximetry.org. (2002, September 10). Principles of pulse oximetry technology. Retrieved from http://www.oximetry.org/pulseox/principles.htm
2. Bailey, J., Fecteau, M., & Pendleton, N. (2008). Wireless pulse oximeter. Informally published manuscript, Worcester Polytechnic Institute, Retrieved from http://www.wpi.edu/Pubs/E-project/Available/E-project-042408-101301/unrestricted/WPO_MQP-Final_04242008.pdf
3. Masip, J. (2012). Pulse Oximetry in the Diagnosis of Acute Heart Failure. ScienceDirect.com. Retrieved December 1, 2012, from http://www.sciencedirect.com/science/article/pii/S1885585712001570
4. United States. Department of Health and Human Services.Telehealth. 2012. Web. <http://www.hrsa.gov/ruralhealth/about/telehealth/>.
5. United Kingdom. Department of Health. Whole System Demonstrator Programme. London: 2011. Web.
6. Holovaty, A., & Kaplan-Moss, J. (2009). The Django Book. Retrieved from http://www.djangobook.com/en/2.0/chapter05.html

Appendix
A
Product Design Specifications

Function:

This pulse oxitelemetry device will automatically collect and transmit patient's blood oxygen saturation data from any location with 3G cellular coverage, to one or more physicians simultaneously. The device will also be interfaced with a front end system, which will allow for the pulse oximeter data to be stored and viewed in real-time on a web interface. In doing so, the device improves patient quality of life by providing freedom of mobility, a hands-free lifestyle, and reducing hospital readmissions.

Client requirements:

Device:
- Wireless transmission from device to base station at predetermined intervals
- Comfortable design that will not burden day to day activities
- Battery life beyond 1 week for discontinuous monitoring
- Ability to customize data collection intervals and signal threshold notifications

Front End System:
- Ability to store large amount of patient biometric data
- Synchronized with patient medical records
- Real-time data visualization
- Intuitive user interface for ease of patient health assessment
- User adjusted alarm set points

Design requirements:

1. Physical and Operational Characteristics

a. *Performance requirements*: Primarily 24/7 monitoring, during day-to-day activities and while sleeping. Monitoring will consist of wireless signal transmission from the device to the cellular network and vice versa. Clinical and ambulatory settings would also be desirable.

b. *Safety*: The thermal state of the device cannot cause discomfort to the patient. Patients cannot be exposed to any harmful currents or voltages from the device. The RF exposure guidelines will be taken into account. Waterproofing the device to limit the likelihood of these events is strongly preferred. It needs to be thoroughly sterilized. Safety warnings will be included and Continua Healthcare Alliance standards will be considered.

c. *Accuracy and Reliability*: Precision and accuracy should very closely resemble the signal outputs of contemporary pulse oximetry devices. A specific signal tolerance from the wireless output relative to the wired output will be determined.

d. *Life in Service*: Signals must be transmitted by the device at least every 15 minutes, 24 hours a day, 365 days per year. Battery life must last longer than one week supporting these transmission intervals.

e. *Shelf Life*: Shelf life and life cycle of usage should be a minimum of 1 year.

f. *Operating Environment*: The device should not be exposed to temperature ranges, pressure ranges, humidity, shock loading, dirt or dust, corrosion from fluids, noise levels, insects, or vibration beyond those of clinical outpatients. Therefore the device should be encased.

g. *Ergonomics*: The device usages will be restricted to the heights, reach, forces, and operation torques standard to clinical outpatients. The user interface should be easy to use and provide the most important data in an efficient web layout.

h. *Size*: Device size and weight will ideally be comparable to or smaller than standard hearing aids, in order to fit comfortably on the ear to allow for minimal lifestyle disruption.

i. *Materials*: Any materials used cannot irritate skin, or be functionally disrupted by bodily fluids and oils.

j. *Aesthetics*, *Appearance, and Finish*: The device should be as close to the patients skin color as possible, in shape that snugly fits behind the ear, with a smooth, comfortable, soft texture and finish. The user interface will be built with bootstrap, which provides modern template styling for an aesthetically pleasing design.

<u>2. Production Characteristics</u>

a. *Quantity*: 1.

b. *Target Product Cost*: Less than $100.00 to purchase, manufacture and distribute each device.

<u>3. Miscellaneous</u>

a. *Standards and Specifications*: FDA approval is required, IEEE wireless transmission certification is beneficial, and the Continua Healthcare Alliance certification is also beneficial. Before trials, IRB approval will be requested.

b. *Patient-related concerns*: Device will need to be sterilized on a monthly basis. Unprocessed patient pulse oximetry frequency responses must be transmitted over a secure network.

c. *Competition*: Masimo, Nonin, and Phillips pulse oximeter

B

Comments on code fragments are indicated with two successive '/' characters.

```
#include <SoftwareSerial.h>  //library used to connect GSM shell through digital ports
SoftwareSerial mySerial(11, 10); //initialize serial connection to GSM using Digital pin 11 for
RX, and 12 for TX.

//intializers for physiolgical paramters
int currentByte;
char signalStr;
boolean sensUnpl;
char plethysmogram;
boolean searchPulse;
boolean noFing;
boolean invalidPulse;
int pulse;
int spo2;
boolean invalidSpo2;
int avgMode;

int phonySpO2;

//initialize mode
//mode 0 = engineer/debug mode
//mode 1 = automatic sending mode
int mode = 0;

//method called by system on startup
void setup()
{
  //begins serial communication with GSM Shell at 1200 bps. Rate slowed down to resolve buffer
issues
  mySerial.begin(1200);

  //begins communication with PC terminal. Used in debug mode
  Serial.begin(19200);
  delay(500);

  //communication with SmithsMedical sensor processor at 19200 bps even parity, one stop bit
  Serial1.begin(19200, SERIAL_8E1);
}

void loop()
{
  //if in debug mode
```

```
if (mode == 0){


  //if user enters one of these characters into the terminal
if (Serial.available())
  switch(Serial.read())
 {
   //user has 10 seconds to type and enter any AT command to SIM900
   case 'g':
     SendATCommand();
     break;
    //sends text message
   case 't':
     sendText();
     break;

   //uploads sensor data to Cosm
   case 's':
     sendtoCosm();
     break;

    //power GSM Shell on or or off
   case 'p':
     power();
     break;

    //get read out of all sensor data on the terminal
    case 'l':
    sensorRead();
    break;

    //set averaging mode to 1,2,3, or 4
    case '1':
    Serial1.write('B');
    break;

    case '2':
    Serial1.write('C');
    break;

    case '3':
    Serial1.write('D');
    break;

    case '4':
    Serial1.write('F');
```

```
    break;

    //sets to automatic send mode (mode 2)
    case 'm':
    Serial.println("Entering automatic mode. Reupload or reset to re-enter config mode");
    mode = 1;
    break;
  }
  //shows any data the GSM Shell is sending to the Arduino
 ShowSerialData();

 }

 if (mode == 1){

 //do nothing for 15 minutes
 int mins = 0;
 while(mins < 15){
   delay(60000);
   mins++;
 }

   //power up GSM
   power();

   //wait for GSM to complete power up procedure to process commands
   delay(19700);

   //acquire data
   // well use random numbers unless sensor is hooked up
   sensorRead();
   delay(2000);

   //upload to server
   sendtoCosm();

   //power down
   power();

 }

}


//send AT commands
```

```
void SendATCommand(){

  //give user 10 seconds to enter command in terminal
delay(10000);

//read it and write it to the GSM Shell
while (Serial.available()){
  mySerial.write(Serial.read());
}
mySerial.println();
}


void sendText()
{
  Serial.println("send attempt begin");
  mySerial.print("AT+CMGF=1\r");    //put modem in text mode
  delay(100);
  mySerial.println("AT + CMGS = \"+40404\"");//select receiving address
  delay(100);
  mySerial.println("OFF");//the content of the message
  delay(100);
  mySerial.println((char)26);//ASCII code to end the message string
  delay(100);
  mySerial.println();
  Serial.println("send attemped");
  if (mySerial.available()){
    char incomingByte = mySerial.read();
    Serial.println(incomingByte);
  }
}



void sendtoCosm()
{
  /**Temperature data was attemped to be read and calibrated */
  double Vout = 0.0049*analogRead(9);
  double temp = ((Vout - 1.088)/(0.1213));

   Serial.print("Temp = ");
   Serial.println(temp);

  Serial.println("Pachube start");
  mySerial.println("AT+CGATT?");
  delay(1000);
```

```
 ShowSerialData();

 mySerial.println("AT+CSTT=\"wap.cingular\"");//attempt to set APN
 delay(1000);

 ShowSerialData();

 mySerial.println("AT+CIICR");//initiate wireless connection
 delay(3000);

 ShowSerialData();

 mySerial.println("AT+CIFSR");//see IP address of GSM Shell
 delay(2000);

 ShowSerialData();

 mySerial.println("AT+CIPSPRT=0");
 delay(3000);

 ShowSerialData();

 mySerial.println("AT+CIPSTART=\"tcp\",\"api.cosm.com\",\"8081\"");//begin connection to
Cosm's socket server using TCP on port 80
 delay(2000);

 ShowSerialData();

 mySerial.println("AT+CIPSEND");//begin send data
 delay(4000);
 ShowSerialData();


 //here are the contents of the data sent to Cosm

  mySerial.print("{\"method\": \"put\",\"resource\": \"/feeds/128148/\",\"params\"");//this patient
is feed 128148
 delay(1000);
 ShowSerialData();
 mySerial.print(": {},\"headers\": {\"X-PachubeApiKey\":");
 delay(1000);
 ShowSerialData();
 mySerial.print("\"Cr-
hi8SpaO81oqfMpj8gDTOVIbGSAKx4cWx1YzNDRXdEcz0g\"}");//pachubeapikey
 delay(1000);
```

```
 ShowSerialData();
 mySerial.print(",\"body\":");
 delay(500);
 ShowSerialData();
 mySerial.print(" {\"version\": \"1.0.0\",\"datastreams\": ");
 delay(1000);
 ShowSerialData();

 //phony values to simulate a valid reading on the Smith's Medical Processor
randomSeed(analogRead(0));

//spo2 = random(91,95);
pulse = random(50, 65);

phonySpO2 = random(94, 100);

 mySerial.print("[{\"id\": \"SimulatedPulse\",\"current_value\": \"");
 mySerial.print(pulse);
 mySerial.print("\"},");
 delay(1000);
 ShowSerialData();

  mySerial.print("{\"id\": \"LiveSpO2\",\"current_value\": \"");
 mySerial.print(spo2);
 mySerial.print("\"},");
 delay(1000);
 ShowSerialData();

 mySerial.print("{\"id\": \"SimulatedSpO2\",\"current_value\": \"");
 mySerial.print(phonySpO2);
 mySerial.print("\"},");
 delay(1000);
 ShowSerialData();



 mySerial.print("{\"id\": \"AvgMode\",\"current_value\": \"");
 mySerial.print(avgMode);
 mySerial.print("\"}");

mySerial.println("]},\"token\": \"lee\"}");


 mySerial.println((char)26);//end the text of the send
 delay(5000);//wait for reply
 mySerial.println();
```

```
  ShowSerialData();

  mySerial.println("AT+CIPCLOSE");//close connection
  delay(100);
  ShowSerialData();


}

 //shows any data the GSM Shell sends to Arduino Mega (used for Debug)
void ShowSerialData()
{
  while(mySerial.available()!=0)
    Serial.write(mySerial.read());
}


//powers on or off gSM Shell
void power()
{
  pinMode(9, OUTPUT);
  digitalWrite(9,LOW);
  delay(1000);
  digitalWrite(9,HIGH);
  delay(2000);
  digitalWrite(9,LOW);
  delay(3000);
}


//read decode data from sensor
void sensorRead(){


  //initalize variable, flags
  currentByte = -12;
  signalStr = 'a';
  sensUnpl = false;

  searchPulse = false;
  noFing = false;
  invalidPulse = false;

  invalidSpo2 = false;
  avgMode = 0;
```

```
Serial.println("Collecting data...");

Serial1.flush();
int iterations = 0;

//incoming data from sensor processor
 while(iterations < 3000){

   if(Serial1.available() > 0) {

     //assign incoming byte to a char
     char incomingByte = Serial1.read();

     //check for sync byte
     if((incomingByte & 0x80) == 0x80) {
       //this is the first byte
       currentByte = 0;

      } else {
       //if not the sync byte, increment
       currentByte++;
     }

     if(currentByte == 0) {

       char sensUnplTrue = (incomingByte & 0x20);
       if (sensUnplTrue == 0x20){
         sensUnpl = true;
       }
         signalStr = (incomingByte & 0x08);
     }
     if (currentByte == 1){
       plethysmogram = incomingByte;
     }
     if (currentByte == 2){
       char searchPulseTrue = (incomingByte & 0x20);
       if (searchPulseTrue == 0x20){
         searchPulse = true;
       }
       char noFingTrue = (incomingByte & 0x10);
       if (noFingTrue == 0x10){
         noFing = true;
       }
     }
     if (currentByte == 3){
```

```
        if (incomingByte == 0x7F){
          invalidPulse = true;
          pulse = (incomingByte);

        }
      }
      if (currentByte == 4){
       spo2 = incomingByte;
       if (incomingByte == 0x7F){
       invalidSpo2 = true;
       }
      }
      if (currentByte == 5){

      char average = (incomingByte & 0x03);
       if (average == 0x00){
        // Serial.println("4 beat Sp02%, 8 beat pulse rate average");
        avgMode = 1;

       }
        if (average == 0x01){
        // Serial.println("8 beat Sp02%, 8 beat pulse rate average");
        avgMode = 2;


       }
        if (average == 0x02){
        // Serial.println("16 beat Sp02%, 16 beat pulse rate average");
        avgMode = 3;

       }
        if (average == 0x03){
        // Serial.println("16 beat Sp02%, 8 beat pulse rate average");
        avgMode = 4;

       }
      }
     iterations++;
    }//end if available

}//end iterations

//print out readings

Serial.print("SpO2: ");
Serial.println(spo2);
```

```
  if (invalidSpo2){Serial.println("Invalid Sp02");}

  //  Serial.print("Pulse: ");
   // Serial.println(pulse);

  if (invalidPulse){Serial.println("Invalid Pulse");}


  if (avgMode == 1){
    Serial.println("4 beat Sp02%, 8 beat pulse rate average");
  }
  if (avgMode == 2){
    Serial.println("8 beat Sp02%, 8 beat pulse rate average");
  }
  if (avgMode == 3){
   Serial.println("16 beat Sp02%, 16 beat pulse rate average");
  }
  if (avgMode == 4){
    Serial.println("16 beat Sp02%, 8 beat pulse rate average");
  }
  if (noFing){
    Serial.println("No Finger in Sensor");
  }
  if (searchPulse){
    Serial.println("Searching for pulse");
  }
  if (sensUnpl){
    Serial.println("Sensor unplugged");
  }

    if (avgMode == 1){avgMode = 4;}
 if (avgMode == 2){avgMode = 8;}
 if (avgMode == 3){avgMode = 16;}
 if (avgMode == 4){avgMode = 16;}


}//end sensorRead
```