Brenton P. Chasse, CSE, Alexander S. Nichols, CSE, Colin C. Morrisseau, EE, and Zachariah G. Boynton, EE

EquiPack: An Injury Reducing Smart Backpack

***Abstract*—The goal of EquiPack is to provide users of our backpack with real time suggestions on how to properly position and wear the backpack. Hospitals across the US report over 7,000 annual emergency room visits due to backpack related injuries. EquiPack aims to reduce the number and onset of short- term and long-term backpack-related injuries.**

I. Introduction

Every year there are over 7,000 emergency room visits due to injuries related to backpacks. This is up over 330% from 1996 [1]. These issues are caused by improper adjustment and wearing of the backpacks [2]. In the past people have simply been recommended to carry less weight or have had to resort to other means of carrying the weight such as rolling bags or weight bearing service animals.

These options are frequently not useful under all conditions. For instance rolling bags are not particularly useful over rough terrain. Ideally people would be able to properly use their bags and therefore carry a full load without chance of injury. Additionally, there are no viable solutions for adventurous people such as hikers or scout troops. This is where EquiPack comes in EquiPack. EquiPack is a simple solution that allows users to see how well their bag is adjusted and provides feedback on how the bag could be fit better.

II. Design

A. Overview

The key to preventing backpack related injury is to provide the user with enough information for them to correctly position their bag. In order to provide the user with this knowledge we have decided to equip our bag with an array of load sensors along the back and straps (Figure 2.1.1). A micro-controller collects and collates the sensor data and transmits it over Bluetooth to an Android application. From there, the application can utilize the processing ability of the mobile device's hardware to quickly process the data and present the user with visual queues necessary for proper adjustment of the bag. In the early stages of our project design, we also looked at a backpack that could adjust the straps automatically while the wearer is moving, thus decreasing back stress. After more research, this was decided to be unnecessary. Back stress could be reduced by simply adjusting the straps properly and limiting the weight of the backpack holds [4]. EquiPack was divided into four main subsections; Weight Sensors, Embedded Programmable Hardware, Weight Analytics Algorithm, and Android Application.

Figure 2.1.1: Sensor locations

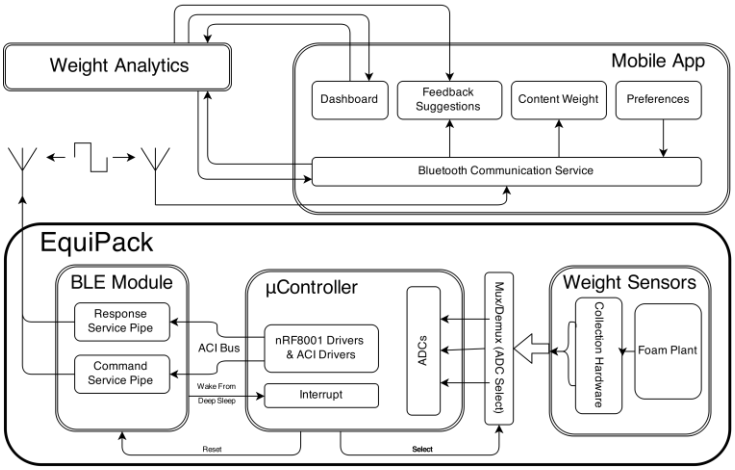


Figure 2.1.2: Block Diagram of EquiPack

The EquiPack system can be broken into multiple subsystems, as can be seen in Figure 2.1.2. Each of these subsystems has their own specific requirements and specifications.

Sensors were specified to be able to handle weights ranging from 1lb to 100lbs. This ensures that intense pressure caused by improper loading, or general abuse of the Equipack solution, would not destroy the sensors. In a similar manner, sensors are required to have a robust housing; they must withstand wear due to time and the elements. Sensors must also be robust when exposed to the elements, for accurate data will be necessary in all conditions. Measurements made from the sensor network must be accurate enough to sense weights in 1lb increments. Finally the sensors must have a small footprint and low power consumption, ensuring long (weeks) battery life and non-invasive integration into the physical bag.

The goal of the weight analytics is to convert the forces determined by the sensors into useful information. The requirements for the weight analytics block are as follows: able to determine the total weight, able to approximate the loaded bag's center of mass, and able to determine the best ratio between pressure on the shoulders and the back that could be achieved by adjusting the straps. The center of mass is required to determine whether objects are aligned correctly in the solutions physical bag. Having a center of mass far from the back will cause a user to lean forward in order to compensate and maintain balance.

The total weight of a pack has been shown to be to the largest cause of backpack related injury. Doctors recommend

B. C. Author from Plainville, Ct (e-mail: bpchasse@gmail.com).
A. N. Author from Quincy, Ma (e-mail: alex.nichols1066@gmail.com).
C M. Author from Syracuse, Ny (e-mail: cmorrisseau@twcny.rr.com).
Z. B. Author from Cambridge, Ma (e-mail: zboynton@verizon.net).

having a weight limit of anywhere from 10% to 25% of the user's bodyweight in a backpack [5]. The focal point of the EquiPack solution is its ability to optimize the strap positions. Strap adjustment suggestions give the user the option of quick feedback by providing them with an easy way to prevent them from becoming injured.

The goal of the microcontroller is to provide EquiPack with a brain that will enable external communication and interfacing with the weight sensors and power systems. These two elements are in turn the heart of EquiPack, the systems that allow EquiPack to perform the key functionality setting it apart from all other backpacks. To ensure that the microcontroller and Bluetooth systems perform in a way that enhances the functionality of the pack and eases user interaction as much as possible, certain specifications need to be met.

The microcontroller should not make a disproportionate draw on battery power: it operates at 3.3V and should draw no more than 10mA of current when on. The same should be true of the Bluetooth Module when on: it operates at 3.3V and should draw no more than 10mA. In addition, the Microcontroller should only exit sleep mode when actively controlling power systems and reading sensor data, and the Bluetooth module should only leave sleep mode when EquiPack is in use. Finally, the relatively large and extensible number of sensors, which will be needed in the EquiPack, necessitates a scalably large number of native ADCs accessible to the microcontroller.

This system meets the power requirements, with the chosen microcontroller, the LPC824M from NXP Semiconductors, having power consumption 35mW [17], and the Bluetooth Module, the nRF8001 from Nordic Semi-conductor, having power consumption 30mW [18]. The LPC824M has 12 12-bit ADCs accessible in parallel. Both the LPC824M and nRF8001 are designed for low-power environments, and have highly efficient sleep modes, which can be entered and exited easily.

The goal of the EquiPack mobile application is to provide a user interface that displays feedback or suggestions to the bag wearer. This application must have an intuitive user interface, secure data storage and transfer, the ability to utilize Bluetooth Low Energy, the ability to send text alerts, an expandable code base, persistent customizable preferences, and reliability. The user interface must be easy to use for anyone who is familiar with the general design patterns of Android applications, allowing for easy user adoption. In making such a user interface, the application should be developed in such a way that new features can be easily added to the existing application, for an enterprise level application should always be expanding post release.

In order to communicate securely, effectively, and efficiently, the Android application must be written using an API that supports Bluetooth Low Energy. Additionally, the application must be developed in such a way that a user can enter custom preferences, which persist over multiple lifecycles of the application. For example, the user's weight is necessary to ensure the bag is not too heavy to safely wear. Last, the Android application must be intuitive to navigate, while providing visual representations of the analytics' feedback.

B. Weight Sensor Network

The purpose of the weight sensors block is to take information from the physical world, in this case weight, and convert it into an electrical signal that can then be processed by the rest of the system. This weight sensor block consists of two elements, a sensing element, and sensing circuitry. The implemented sensing element is conductive foam. This foam has the property that its resistance changes with compression [3]. Some preliminary data can be seen in the Figure 2.2.1.

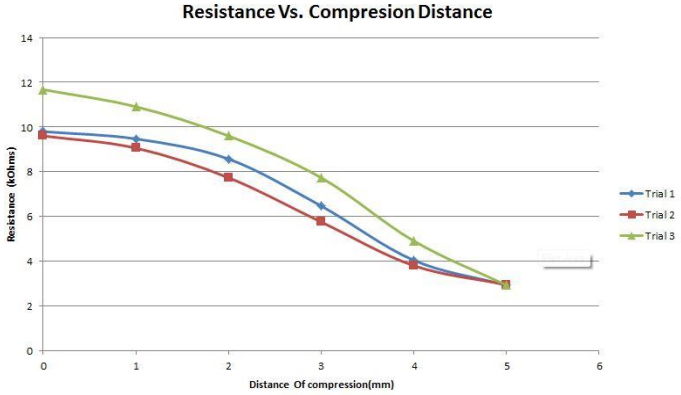


Figure 2.2.1: Resistance of foam vs Distance of compression for three different trials of compression.

In order to read this change in resistance a voltage divider is used to create a reference voltage that is fed into an op amp circuit. The foam has a capacitance intrinsic to it's construction. This capacitance prevents the voltage across the foam from changing instantaneously, for the internal capacitance must charge through the foams resistance. Thus, the foam's charging and discharging can be modeled as an RC network in which a resistor is in series with a shunt capacitor on the output. This RC constant currently dictates how quickly measurements can be taken from the network. This circuit amplifies the DC voltage applied to the input by the ratio of the resistors in the feedback network. At DC, the capacitor behaves like an open circuit. Therefore, the capacitor has no impact on the DC operation. At higher frequencies the capacitor behaves similarly to a short circuit. With the capacitor acting as a short the circuit behaves like a buffer. This reduces the total output noise by not allowing the noise to be amplified. More succinctly put, the addition of the capacitor improves the signal to noise ratio of the circuit. The schematic for this network is displayed in appendix 1A.

The op amp that was chosen was the LM324N. This op amp was picked because it allows for a single supply operation and low power consumption, parameters that are critical when operating in a battery powered system. Courses such 323,324, and 575 were all useful in the development of this circuitry. They imparted the knowledge to develop the appropriate transfer functions, choose a suitable circuit topology, and select necessary component values.

Additionally Physics 151 has been useful for understanding how the force on the foam will translate into its distance of compression. In order to test this block a weight rig will need to be setup that will allow the application of a known weight

to the sensor. Output voltage can then be measured versus applied weight in order to characterize the sensor.

C. *Weight Analytics*

The weight analytics block is a series of algorithms that are contained in the smartphone app. The first goal that the weight analytics subsection needs to achieve is to determine the total weight of the backpack. Initially, the plan for this was to develop an algorithm taking the sum of all the weights of the sensors and summing them in a way to determine the total weight of the bag. As research continued, there were too many variables that affected the total weight of the bag. Sensors were unable to determine the total weight of the bag without knowing the angle at which the bag was held.

Once it was determined that an additional sensor would be needed the next challenge was to determine what additional sensor could provide extra information. We decided to attach a load cell to the lower strap of the backpack. A load cell is a strain gauge attached to a soft piece of metal. when the metal is bent it changes the resistance of the strain gauge. This is the sensor used in common luggage scales and can be implemented using a whetstone bridge and an amplifier. If the backpack is sufficiently heavy the effect of friction is negligible as most contact is perpendicular to the force of gravity. The backpack then acts as a simple pulley, where tension along the backpack is equivalent to the force at the end of the rope (and in this case the load of the backpack). Since there are two straps the actual load is one half the entire weight. This theory was tested by disassembling a luggage scale containing a load cell and attaching it in between the lower strap of the backpack.

The next goal of the weight analytics subsystem is to determine the center of mass of the backpack. When it is used in the Android app, the center of mass will determine if the contents of the bag need to be adjusted closer to the front of the bag. Once the center of mass is determined, the distance away from the back can be set to a threshold, and if that distance passes that threshold, the user will be notified to adjust the contents of the bag.

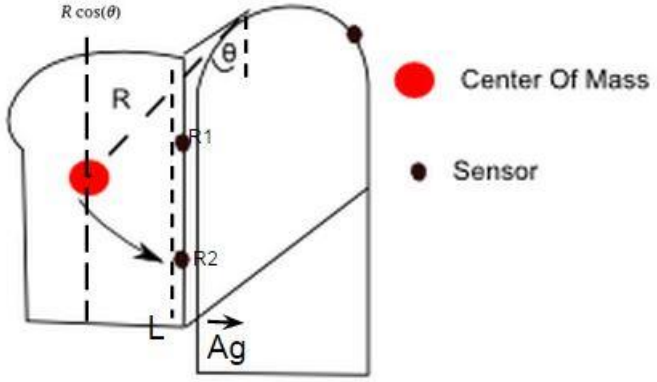


Figure 2.3.1: Free body diagram showing the torque vector of interest.

The center of weight is determined by using a single point of force along the back and comparing it to the weight of the pack when compared to the total force distributed along the back. The upper part of the strap acts as a hinge, which directs force downward towards the bag in the form of torque along the hinge. The final equation determined as:

$$R\cos(\emptyset) = \frac{L^3}{2\,m\,g}\left(\frac{p_1}{r_1^2} + \frac{p_2}{r_2^2}\right) + A$$

Here, L is the length, p is the pressure from a sensor, r is the distance from the top, and A is a constant determined by the force that the lower strap pulls the bag. The algorithm uses two points and averages them to prevent anomalies from occurring. This formula will have to be tested physically as it makes a lot of assumptions. It assumes the center of weight is all based on one point, it assumes the bag acts as a rigid body once settled and that the friction forces are minimal.

The strap optimization algorithm follows the fundamental rule that your shoulders are not meant to bear weight. The algorithm shown below first attempts to acquire symmetry between the two shoulders and then adjusts the straps to the minimal amount of weight on the shoulders [6].

The algorithm for the weight sensor block:

```
while(abs(left - right) > minimum balance
threshold)
    loosen higher pressure strap until equal;
//determine strap location by checking pressure on
shoulders
if lower strap sensors < upper strap sensors
    set strap location to high
else
    set strap location to low

while(max(shoulder pressure at t+1)< max(shoulder
pressure) at t)
    if strap location == low
        tighten both straps
    else
        loosen both straps
if any strap is above a safety threshold
    loosen both straps
    break;
```

Figure 2.3.2: Pseudocode for the strap optimization algorithm.

The smart backpack is able to determine the strap location based on the ratio between the upper and lower shoulder sensors. Pressure on the higher sensor is indicative of a low placement and pressure on the lower strap is indicative of a higher placement.

Verifying these algorithms was the most difficult process of the weight analytics subsection. While there are many software applications able to do weight distribution analysis for simple parts, there are limited options for large soft body systems. The cloth properties of a backpack make it much more difficult to analyze. As a solution to this, the computer animation software 'Blender' was used to model the backpack. Blender contains the comprehensive open source physics engine Bullet Physics, which is able to simulate large scenes with a vast amount of objects. It also has the ability to directly simulate soft body physics to mimic the cloth found on backpack straps.

Since blender is not analysis software, it does not have the ability to directly view active forces on an object, and

therefore the pressure acting upon two objects. In order to test these objects, a series of the point forces along the backpack were calibrated until the backpack stayed in equilibrium

To build these blocks skills learned from basic math and physics classes such as Physics 151 and Math 131, 132 and 233 were used for the hand calculations of these algorithms. In addition to that, methods derived from Data structures and algorithms were used to develop the optimization algorithm and employed a feedback mechanism similar to those discussed in Electronics II. Moving forward, these algorithms need to be implemented in C code for use in the smartphone app. An API will need to be developed so that the algorithms will be able to be easily edited and debugged in the design process.

D. Microcontroller and Bluetooth Low Energy Hardware

This block is the brain of every EquiPack. The microcontroller reads sensor data from peripheral weight sensors positioned throughout the pack, and can compress and store the data, or send it directly via Bluetooth to a Peer device requesting weight data. While this is the main purpose of the block, it will also function to conserve battery by cycling power to peripheral sensors and inform users of battery level.

The implementation uses the LPC824M Low Energy Microcontroller from NXP Semiconductors, and the nRF8001 Bluetooth Low Energy Module from Nordic Semiconductor. These elements will be incorporated into the central PCB, which will eventually integrate the microcontroller, Bluetooth module, power systems, and analog signal processing circuitry.

Knowledge required to build and debug this block was acquired in Computer Systems Labs I & II (ECE353 & ECE354). Computer Systems Lab I focused on use of UART, which is an i*ntegral component of the custom* Automatic Control Interface (ACI) between the LPC824M and nRF8001. Computer Systems Lab I also necessitated use of a logic analyzer, which was used for debugging by reading multiple UART/Control channels on the ACI bus to verify message packets sent by and received from the Bluetooth module, as well as control signal timing parameters. Computer Systems Lab II involved gaining a greater understanding of Non-Volatile Memory, Interrupt Request Priority, and interfacing with peripherals via Memory Mapped I/O, all of which are important components of the overall design of the microcontroller code.

In addition to what has already been learned, proficiency needs to be gained with PCB layout and design, and effective low-energy sleep management for the microcontroller and Bluetooth Low Energy Module.

To verify the functionality of Bluetooth communication, a standalone Android Application was created, implementing BLE communication. This application was used to connect, pair, and send data to and from the microcontroller. The process, which could be monitored and debugged on both the Android Application (using the IDE Android Studio and the Microcontroller, followed the following process:

1. The microcontroller sends setup information to the Bluetooth module
2. The Bluetooth Module sends a “State Changed To Standby” Message
3. The microcontroller sends a “Start Broadcasting” Message, which will begin a Bluetooth Low Energy Broadcast event, making the Bluetooth Module discoverable
4. The application, listening for advertisement packets from devices, finds the EquiPack Bluetooth Device, and begins connecting
5. The Microcontroller receives a “Connection with Peer Device Successful” message
6. The App also receives a “Connection Successful” message, and begins broadcasting data to the microcontroller.
7. During debug, the content of these data messages can be verified
8. The content sent in response by the microcontroller can be verified on the App side during a concurrent debug session

Tests were also developed to verify the functionality of the microcontroller’s native Analog to Digital Converter (ADC). The 3.3V Analog Reference Voltage provided by the microcontroller was divided by a potentiometer and fed back into the ADC port. The value at the ADC register was polled by the microcontroller, prompted by a message sent from the Android application. This was parsed into a reference voltage and displayed in the application. Polling occurred as frequently as 20 times per second. It was seen that twiddling the Potentiometer between 0V and 3.3V triggered a commensurate value change in the Android application display.

E. Mobile Application

EquiPack’s mobile application block serves as a means to receive and store data, perform data analytics, and visually aid the EquiPack user in correctly wearing their backpack. This block can be thought of as the interface through which a user of an EquiPack can setup and use their bag to its full potential.

Throughout the development of this application, many software design principles (java in ECE121, data structures and algorithms in ECE242, large scale software engineering in ECE373, “user-first” design methodologies from industry experience) must be followed in order to produce an end product which is open-sourced (see Appendix 4a), easily accessible, expandable, and robust. As a developer of such a product, one must first become familiar with the design methodologies suggested by Google Inc., the maintainers of Android, prior to development to ensure the product does not begin down a path which will hinder its growth.

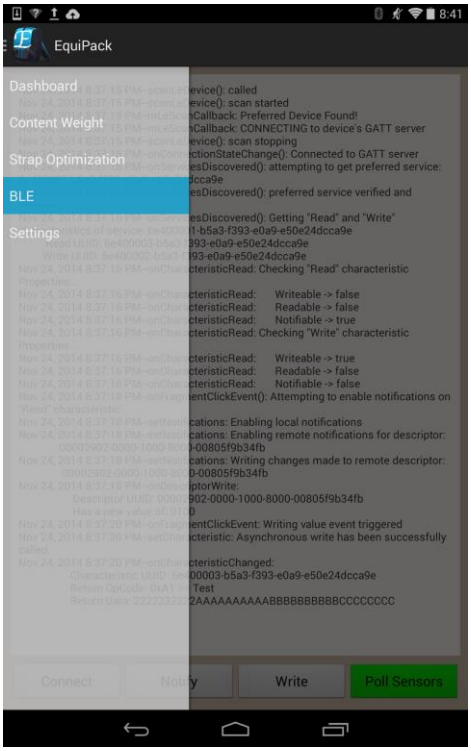


Figure 2.5.1: Equipack's navigation drawer opened while in the BLE Test fragment view

The top-level navigation scheme chosen for EquiPack's mobile solution is a Navigation Drawer (See Figure 2.5.1). A developer should chose this navigation scheme when providing the user with multiple views that are not tightly related from a user's perspective, but remain tightly bound internally. A Navigation Drawer allows for the distinction between different views while still allowing for them communicate with one another via a common "main" activity. For example, the feedback view and the settings view are not closely related in the user's perspective, but the settings are necessary for proper feedback. Additionally, a Navigation Drawer allows for expansion to a multi-leveled navigation scheme, thus making it possible to vastly expand the application's features and functions with ease [8].

When presenting the user with a visual element, there are different design approaches that one can take. For EquiPack's application, a Fragment-Activity scheme has been implemented. Fragments can be thought of as a modular section of an Activity, consisting of one or more user interface elements. In a Fragment-Activity scheme, a single activity can control the lifecycle of multiple fragments. This allows for all of the fragments to co-exist and communicate with each other as needed [10]. Furthermore, grouping many user interface elements into a single fragment (as opposed to a View-Activity scheme) allows for the reuse of user interface elements in different portions of the application. For instance, the Navigation Drawer fragment can be made visible while any other fragment is visible. This allows for the re-use of a pre-defined set of user interface elements for navigation, rather than reproducing these elements individually each time they are needed [9]. Lastly, using a Fragment-Activity scheme allows for improved user experience.

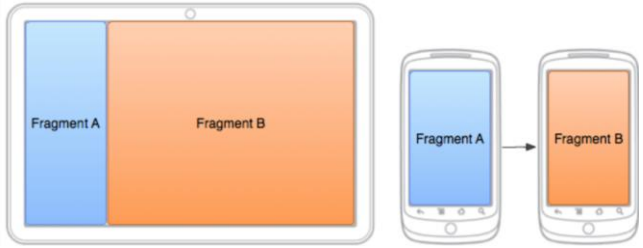


Figure 2.5.2: Two fragments can be re-used and displayed in a different manner on different classes of devices [9]

As can be seen in Figure 2.5.2, an application's main activity is able to display fragments in different fashions based on what type of device the application is being run on if the application uses a Fragment-Activity scheme. In a View-Activity scheme, two separate applications would need to be developed in order to produce the same observed behavior. Fragments allow EquiPack's application to better enable development across multiple classes of devices with a maximum percentage of reusable code [9].

The behavior of the application must be consistent with that of other popular Android applications as well as with the expectations of its user. This ensures a higher level of accessibility within our mobile application [9]. In addition to a global top-level navigation scheme, the hardware based navigation methods must also be implemented. Since a Fragment-Activity scheme was used, a back-stack was implemented in order to save each fragment transaction (the act of changing the visibility of a fragment or switching between different fragments). These transactions must be pushed onto a stack that is accessible to the Operating System so that the previous state of the application can be restored when the hardware's "back" button is pressed [7].

Additionally, a scheme for saving and restoring states of different views contained within a fragment has been implemented. The importance of such a mechanism will be touched upon briefly in the description of the Bluetooth Low Energy testing scenario. As more fragments are populated with changing views, this mechanism will be implemented in the new fragments as necessary.

This mobile application is intended to serve as a mediator between the embedded system of an EquiPack bag, and the wearer of an EquiPack bag. Sensor data collected by the embedded micro-controller is collated and transmitted via Bluetooth Low Energy (BLE) to the mobile device. Given that EquiPack is a low power consumption device, it is critical that the mobile application is built to run on hardware that can support BLE. For this reason, Android version 4.4 (KitKat) was chosen as the development API level. API level 18 provides a BLE API which was used to facilitate communication between the mobile device and the embedded system's Generic Attribute Server (GATT) [12].

Figure 2.5.3: The BLE Test fragment is the visible collection of views

For testing purposes, the mobile application currently features a fragment dedicated to the setup, connection, and communication associated with the BLE stack (Appendix 4b), which can be seen in Figure 2.5.3. When the "connect" button is pressed, the application makes a system call to ensure that Bluetooth Low Energy is supported on the device. If so, the Bluetooth hardware should be turned on. If it has not yet been turned on, the user is prompted to do so. When the hardware is enabled, the application begins to search for BLE devices. Once a device with a name matching the user preferred (preferences will be discussed shortly) name, an attempt to connect to that device's GATT server is made. Upon a successful connection, the application begins to read the GATT server's services. If the preferred service is found, then the characteristics of that service are read. If the preferred characteristics are present, the descriptors for those characteristics are read. If the descriptors of each characteristic match the anticipated descriptor [13] (write characteristic is writable, read characteristic is able to send notifications), then the "Notify" button in the BLE Test fragment is enabled.

When a user presses the "Notify" button, the application attempts to write to the read characteristics "notify" descriptor. If this descriptor change is successful, then the user will be able to click on either the "Write" or the green "Poll" button.
If the user presses the "Write" button, then the value that was set in the application's preferences will be written to the write characteristic. The user will then notice that the peripheral device echoed the value written to the write characteristic on the GATT server's read characteristic.

If the user were to press the green "Poll" button, the button changes its color to red, then the application writes to the GATT server's write characteristic a value containing the "Poll" operation code (0x02) as the first byte in the transmission. This triggers the embedded system to transmit its most recent readings on its ADCs. Once a notification containing the ADC data has been received, another identical write is performed. This cycle continues until the user presses the red "Poll" button, in which case the cycle is broken.

Each step that the application makes during the use of this BLE Test fragment is visibly printed in a text-based log that is visible over the majority of the screen. At any point, if the user were to switch to a different fragment or navigate away from the EquiPack application, then the fragment will save the contents of its TextView (View containing the log) and the states of all of the buttons. The contents of the TextView and the states of the buttons are then restored the next time that the fragment is made visible. If the application is quit, then the contents of this fragment are lost and the default for each view is loaded the next time the fragment needed.

Preferences are a vital part of any modern application. They serve as a mechanism for users to enter customizable information. The information entered in preferences is intended to be that which is unique to each user of the application. Thus far, a user is able to enter the name of their EquiPack's advertised GATT server, the UUIDs for the read and write characteristics or their bag, a 40 character long hex string to be written, and their weight. If these settings are set to something valid, then the application begins to use the new values immediately. These preferences are stored locally and are reloaded each time the application starts up.

One test procedure that has been performed to ensure the functionality of the most implemented features at once (during MDR) is as follows (Appendix 4b):

1. Disable Bluetooth
2. Start up the application and set the user preferences to something valid and unique to your EquiPack bag.
3. Quit the application.
4. Start up the application and notice that all of the preferences are still populated with the values you previously entered.
5. Navigate to the BLE Test fragment via the Navigation Drawer.
6. Press the "Connect" button.
7. Enable the Bluetooth hardware when prompted.
8. Wait for a successful connection to the GATT server.
9. Wait for the successful verification of the read and write characteristics.
10. Take note of the Log contents and test button states.
11. Navigate away from the BLE Test fragment.
12. Navigate back to the BLE Test fragment and notice that the log contents and test button states are restored to the same state that they were previously in.
13. Press the "Notify" button and see that notifications have been successfully enabled.
14. Press the "Write" button. Notice that the value echoed back from the GATT server matches the value entered in the preferences.
15. Navigate to the preferences and change the write value.
16. Press the "Back" (hard) button to navigate back to the BLE Test fragment and press the "Write" button again.

17. Notice that the new write value is echoed back by the GATT server.
18. Press the "Poll" button and notice that live ADC data is being displayed in the log. These values change as the ADC reading change (Demoed with potentiometers connected to the ADCs).
19. Press the "Poll" button again to stop polling.

If all of the application behaviors are as described in the above user scenario, then the core features of this application have been tested and verified

III. Project Management

EquiPack's design and development team consists of two Electrical Engineering majors and two Computer Systems Engineering majors who have known one another since the fall of 2011. Internally, our project can be divided into a software-based component and a hardware based component. Alexander Nichols and Brenton Chasse have excelled in prior software based group projects such as those associated with ECE354 and CS377. Colin Morrisseau and Zachariah Boynton have excelled in prior hardware based group projects such as those associated with ECE213 and ECE324. Our team's history of working together is echoed in how our team assists one another internally.

The team meets at the beginning of each week with our advisor, Professor Chris Salthouse, to discuss the progress we made over the previous week. Furthermore, we discuss our planned progress for the upcoming week. In addition, the team meets twice a week. Monday evening meetings are scheduled as needed in order to align the team's progress for the week. Wednesday evening meetings are scheduled every week in order to discuss each individual's progress as well as any challenges or roadblocks that any member may be facing. The team's frequent group meetings ensure that each individual's ideas for EquiPack's future are aligned, both within the group and between the team and its advisor.

Zachariah is an Electrical Engineering major with an interest in circuit design. His past work has been focused on analog and radio frequency design. He has experience in both industry and research environments. As a result of these prior experiences, Zachariah is responsible for the weight sensors block. This block is dependent on analog circuitry to interface with the weight sensing material. Zachariah has worked with Colin to determine the specifications for the sensors in order to ensure the success of the weight analytics. Zachariah has also worked with Alexander on the hardware interface between the microcontroller and the Bluetooth unit

Colin is an EE with a relatively strong background in software engineering and an interest in applying his knowledge of physics to an engineering challenge, such as this one being solved by this team.

Colin's math minor helped him acquire the skills necessary to develop algorithms for the weight analytics. Colin's broader interest in the fields of electrical and computer system is crucial for this subsystem because it overlaps with every single other subsystem. As an intern at Verizon this summer, Colin developed algorithms for sorting large amounts of statistical data into useful and reliable information.

Alexander is a CSE with a passion for embedded systems development. As team leader for both his Computer Systems Lab I and II groups, Alex took a front seat role in the design, development and debugging of large projects written in C and Verilog, interfacing with a number of diverse microcontroller systems and technologies.

Brenton is a CSE with a Computer Science minor who is most at home when engineering software. Although his experience with software engineering has been focused on C++ and JavaScript applications, he has a strong passion for producing high quality software and is open to the challenge of developing in an unfamiliar environment. With prior experience in industry level user experience based user interface development (Design thinking development methodology), industry level object oriented software engineering, and embedded software design, it is natural for Brenton to take on the role of designing and implementing an Android based application consisting of a user interface and Bluetooth communication, for retrieving feedback from and interacting with an EquiPack bag. Additionally, Brenton has proven industry experience leading a team of peers, thus it was natural for him to take the team's management role.

Since Brenton took the role of Android developer prior to having ever developed for an Android environment, Alex, who has extensive experience with Android development and has previously implemented the Android BLE stack (although not for the same API level as is used within the EquiPack application) and is Brenton's housemate, was available to answer Brenton's questions about the design principles of Android when such questions arose.

Figure 5.1: Timeline of EquiPack Integration

IV. Conclusion

Thus far, the BLE communication protocol, which will be used to transmit raw sensor data from the EquiPack's sensor network to the user's mobile device, is fully integrated. The functionality of this communication represents the completion of critical integration between the embedded system and the mobile application. Without a means to transmit data to and from these two subsets, the project would have much difficulty coming to completion in a timely manner.
Furthermore, the core fragments and navigation needed to develop a user friendly Android application have been put in

place, leaving the application in an excellent state to expand it's visual and functional aspects.

The microcontroller and Bluetooth Module are able to establish a robust connection with an Android Peer Device, capable of weathering multiple disconnection events. It is also able to read multiple ADC channels in parallel and stream these data over Bluetooth to an Android Peer Device at 20 baud.

The sensor network has been able to accurately take weight data for lighter weights (under 5lbs). Going forward the sensors will need to be outfitted with housing to ensure robustness of the system. Sensors will also need to be modified to accommodate a wider range of weights. Finally sensors will need to be interfaced with the microcontroller so that data can be read quickly and accurately. Foreseeable issues will be ensuring quick readings from the sensors due to the nature of charge dissipation in the foam. Once the sensor design is finalized, the process of integration will begin.

Within the Weight analytics subsystem, all algorithms have been developed. They will needed to be tested in real world settings, to make sure that the assumptions made while calculating the models are correct. This testing will be performed once the weight sensors have been implemented into the actual backpack and creating a prototype we can test with. In addition to that, an algorithm for predicting and removing the *RC* decay caused by the sensors will need to be developed in order to speed up the poling time of the software.

As team EquiPack moves forward, the team will be collecting the sensor network data and transmitting it over the implemented BLE stack to the Android application. It is here that the analytics can be performed. Visual representations of the analyst's suggestions and feedback will be presented to the user. Additionally, any extreme usages of the backpack will trigger the sending of a text message from the User's Android phone to an application subscriber's cellular device.

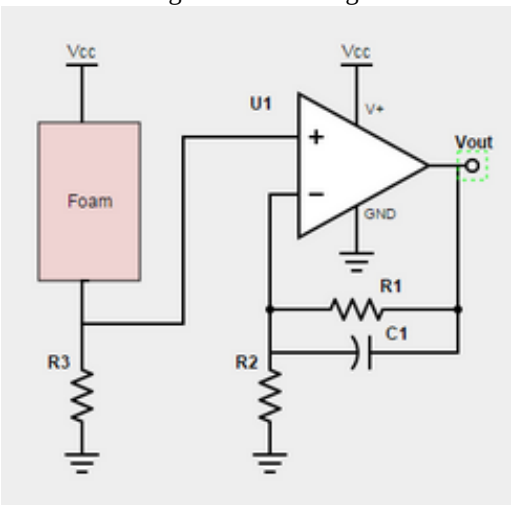
There are still non-trivial amounts of work to be performed in order to: collect reliable and consistent sensor data from the sensor network embedded within the backpack itself; analyze and provide feedback to the user via the mobile application based off of the sensor data received over BLE from the embedded system (Appendix 6a); provide a battery powered, weather resistant, backpack embedded sensor/computational network.

After integrating the sensor network with the embedded micro-controller, the sensor data can be used within the weight analytics to relate the sensor data to the EquiPack's position on the user's back. Once this relationship has been made, the Android application can display useful information to the user in order for the user to optimize the position of their backpack in order to minimize potential health risks.

During this integration, iterative steps need to be made in order to minimize our embedded footprint (PCB design, fabrication, and verification), make our embedded system weather resistant (hydrophobic coatings), and embed our sensor and computational network within a physical backpack.

APPENDIX

1A.) The schematic diagram for the weight sensor block.

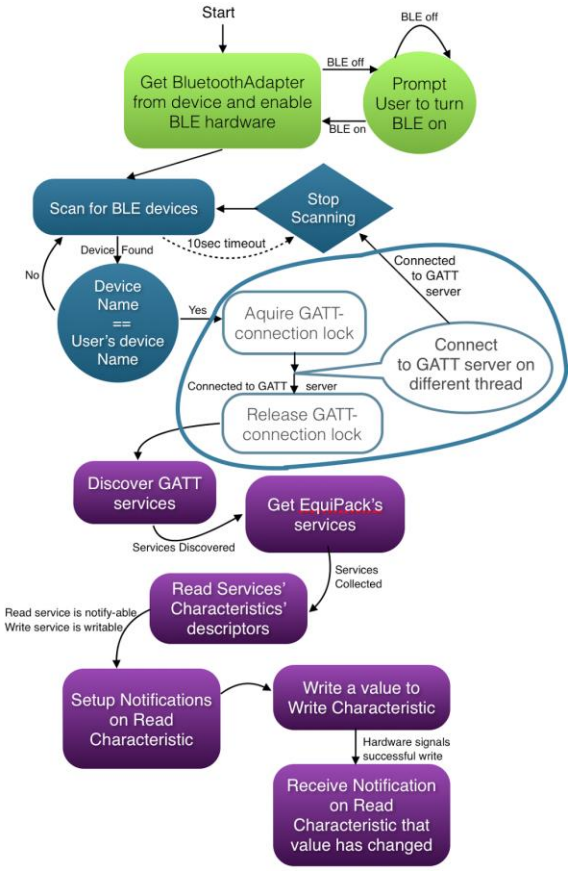


4a.) EquiPack Mobile Companion GitHub URL.

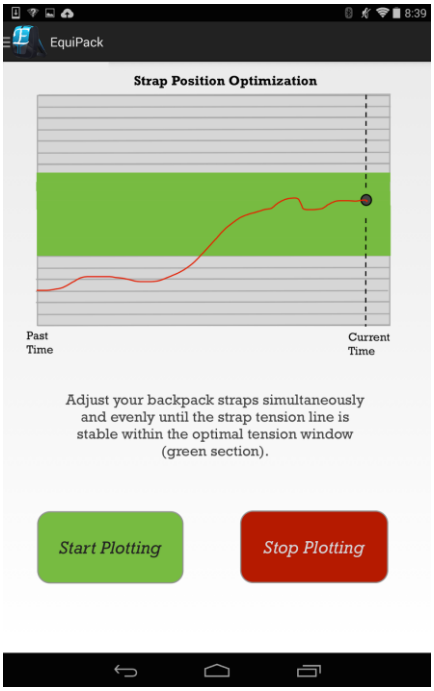
https://github.com/bpchasse/EquiPack

"MDRFeatureAdditions" branch was the source code used for the MDR demo.

4b.) Flow chart depicting the actions taken within the EquiPack app in order to use BLE



6a.) Preliminary design for presenting strap position feedback to the EquiPack mobile user.



ACKNOWLEDGMENT

Advisor: Chris Salthouse (Professor – UMass Amherst)
Evaluator: Israel Koren (Professor – UMass Amherst)
Evaluator: Qiangfei Xia (Professor – UMass Amherst)

REFERENCES

[1] M. Arnsdorff. *Mounting Research on Backpack Use* [I.C.P.A. Newsletter, May-June 2002] http://www.arisewellness.com/pdfs/BackpackUse.pdf

[2] Hasbro Children's Hospital *Backpack Safety* [Online]. Available: http://www.hasbrochildrenshospital.org/backpack- safety.html

[3] Inventables. *Conductive Foam Product Page* [Online]. Available:https://www.inventables.com/technologies/conducti ve-foam

[4] C Devroey, I Jonkers, A Becker, G Lenaerts, A Spaepen(2007) *Evaluation of the effect of backpack load and position during standing and walking using biomechanical, physiological and subjective measures* Ergonomics Vol. 50, Iss. 5

[5] K McCarthy, *Back to School Safety Tips* [Online]. available http://www.arspecialty.com/dr-mccarthy-offers- back-to-school-backpack-safety-tips/

[6] Negrini, S., & Negrini, A. (2007). Postural effects of symmetrical and asymmetrical loads on the spines of schoolchildren. Scoliosis, 2, 8. doi:10.1186/1748-7161-2-8

[7] Google Inc. *App Structure* [Online]. Available: http://developer.android.com/design/patterns/app- structure.html

[8] Google Inc. *Navigation Drawer* [Online]. Available: https://developer.android.com/design/patterns/navigation-drawer.html

[9]Google Inc. *Building a Flexible UI* [Online]. Available: http://developer.android.com/training/basics/fragments/fragme nt-ui.html

[10] Google Inc. *Fragments* [Online]. Available: http://developer.android.com/guide/components/fragments.ht ml

[11] Google Inc. *Views* [Online]. Available: http://developer .android.com/reference/android/view/View.html

[12] Google Inc. *Bluetooth Low Energy* [Online]. Available: https://developer.android.com/guide/topics/connectivity/blueto oth-le.html

[13] Bluetooth SIG, Inc. (2014). *Descriptors* [Online]. Available: https://developer.bluetooth.org/gatt/descriptors/ Pages/DescriptorsHomePage.aspx

[14] Texas Instruments. *LM342N Data Sheet*. [Online]. Available: http://www.ti.com/lit/ds/symlink/lm124-n.pdf

[15] Costa, J.C., M. Oliveria, A. V. Machado Machado, S. Lanceros-Mendez, and G. Botelho. *Effect of Antistatic Additives on Mechanical and Electrical Properties of Polyethylene Foams*. 2009. Print.

[16] T. C. Carusone, D. A. Johns and K. Martin, *Analog Integrated Circuit Design* J. Wiley, 2nd edition. Print

[17] NXP Semiconductors Technical Staff, *LPC82x Product Data Sheet*, NXP Semiconductors, 2014

[18] "nRF8001 Bluetooth® low energy Connectivity IC", *nRF8001*. [Online]. Available: http://www.nordicsemi.com/eng/Products/ Bluetooth-R-low-energy/nRF8001. [Accessed: Oct. 15, 2014]