Smart Gloves – Intelligent Motion-tracking Weightlifting Gloves and Phone Application

Group 22 - Andrew Smith, Scott Suarez, Jason Surh, Luis Gamarra

Dept. of Electrical and Computer Engineering
University of Central Florida, Orlando, Florida
32816-2450

***Abstract* – Smart Gloves, an independently designed and originally inspired Senior Design project, was created with one goal in mind: to provide a structured boost to a user's weightlifting workouts. Fusing IMU motion tracking and wireless communication technologies, exercises are monitored and feedback is provided to users, helping to maximize workout efficiency, reduce the risk of injury, and boost motivation. A small glove-mounted PCB, containing an IMU and Bluetooth-capable microcontroller, wirelessly communicates with a companion smartphone application powered by Unity software to track the orientation and movement of each individual glove given parameters specific to individual exercises. All hardware was designed to avoid interference with a user's range of motion, and the software was designed to be intuitively usable by weightlifters of all experience levels.**

***Index Terms* – IMU, Bluetooth, microcontroller, digital filters, PCB, gyroscope, accelerometer, application, Unity, iOS, motion tracking**

I. Introduction

Weightlifting is not often seen as an activity requiring a great deal of thought, with those inexperienced to the sport incorrectly assuming that repeatedly picking up and putting down weights involves little to no technicality. In reality, every individual exercise requires the use of proper form in order for a weightlifter to maximize the efficiency of the exercise and reduce the risk of injury. The concept of "form" can be loosely defined as the correct way to perform an exercise, including everything from range of motion to angle of rotation. Proper form is often something that weightlifters learn through trial and error over long periods of time and can potentially turn beginners away from weightlifting. Smart Gloves were designed to provide assistance with the development of proper form by tracking the orientation, angular rotation, and linear acceleration of each glove worn by the user and providing audio feedback when the movement detected is uncharacteristic of parameters set for each individual exercise.

In addition to providing basic form tracking, Smart Gloves and the companion smartphone application help users better structure their workouts. Individual exercises are saved on the application, with user-determined parameters such as number of sets to be performed per exercise and the number of repetitions per set. This allows users to plan workouts in advance and track certain exercises with the simple push of a button on the application. This, in addition to other features such as rest timers that users can choose to activate between sets, will keep users motivated during workouts by taking out much of the guesswork that goes into planning effective workouts.

II. Hardware Design

A. Overall System Design

Smart Gloves were designed with ease of use in mind; as an item that users would bring with them into the gym, it was necessary to avoid designing a device that would interfere with workouts and be an overall inconvenience to work out with. As such, several basic hardware specifications were outlined at the conception of this project to dictate the overall design and ensure that this goal was met by our final prototype. Table I, shown below, displays these specifications.

Table I: Preliminary Hardware Specifications

Requirement	Specification	Value
1	Small Size	< 90 x 90mm
2	Impact Resist.	Up to 10kg
3	Sensor Latency	< 0.5s
4	Battery Life	2 hours min.
5	Comm. Range	2 meters min.

To meet these specifications listed above, Smart Gloves hardware was designed to be as simplistic as possible while still providing all the necessary sensory data and computational ability for our software. This kept the physical size of the PCB small enough to avoid interference with a user's exercises and minimized current draw from included major components to maximize battery life. A simplified block diagram of the system is seen in Fig. 1.

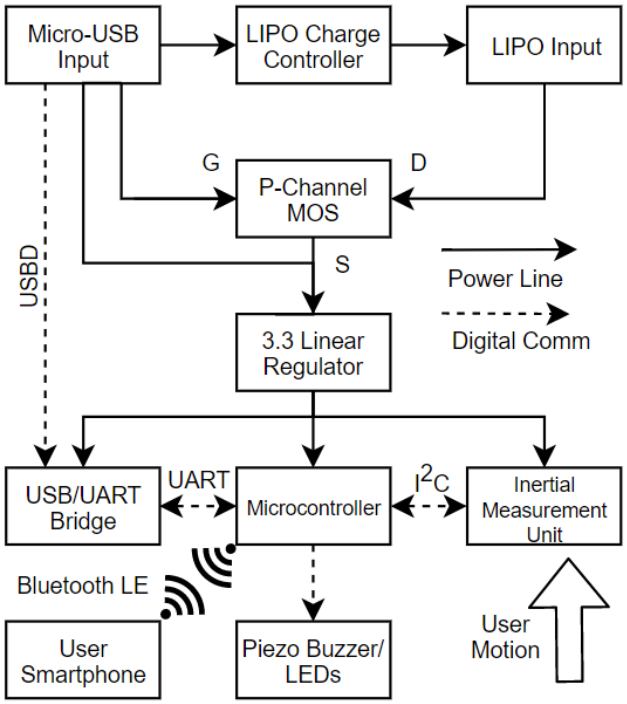

Fig. 1. Simplified block diagram of Smart Gloves Hardware

All blocks in Fig. 1 represent major/notable components that have been implemented in our PCB design, barring the User Smartphone for obvious reasons. The three major digital busses present are the USBD+/- busses from the micro-USB input to the USB/UART bridge, the UART busses from the USB/UART bridge to the microcontroller, and the I^2C busses from the microcontroller to the IMU.

The power line circuitry is set up so that while the device is connected to the micro-USB port, only current from the micro-USB input will pass through the 3.3V linear regulator while preventing current from the LIPO battery from doing so via the p-channel MOSFET. Additionally, while both connections are made current from the micro-USB connection will pass through an intelligent LIPO charge management controller and recharge the LIPO battery until the charge management controller detects full charge. After passing through the 3.3V linear regulator, current will be supplied to all three major digital components via a 3.3V power bus.

B. System Components

Inertial Measurement Unit – the primary sensor used in this design is the Bosch BNO055 9-axis Absolute Orientation Sensor. This MEMS sensor incorporates a triaxial accelerometer, gyroscope, and magnetometer into a single package; however, the sensors of primary concern for our implementation are the accelerometer and gyroscope. The triaxial accelerometer measures linear acceleration across 3 axes with use of a 14-bit ADC and four levels of sensitivity: +/- 2, 4, 8, or 16G. The triaxial gyroscope incorporates a 16-bit ADC to provide accurate angular rate measurements across 3 axes with sensitivity from +/- 125 to 2000 degrees per second. This device communicates with the microcontroller using I^2C communication protocols. Both SDA and SCL lines are connected to digital I/O pins on the microcontroller, with both communication lines powered by the 3.3V power bus via 10k pull-up resistors.

Microcontroller – the Bluetooth LE-capable microcontroller used in our design is the nRF52832. This chip is a 32-bit ARM Cortex-M4F based processor featuring single-precision floating point operations with a clock rate of 64 MHz, 512 kB of flash memory, and 64 kB of RAM. The integration of a Bluetooth transceiver allows us to save space on our board, and the transceiver itself is a 2.4 GHz radio transmitter compatible with 1Mbps and 2Mbps transmission modes. The maximum throughput with this connection is 120 bytes every 7.5ms, which is an acceptable benchmark for our purposes. As previously mentioned, this microcontroller communicates with the IMU via I^2C protocols and is connected to a configurable interrupt pin on the IMU. An active low reset pin is attached to the 3.3V power bus to keep the device running: if a reset is required this pin can be pulled to ground with a simple push button on the board. Other simple components, such as the piezo buzzer and status LEDs, are connected to the microcontroller with general purpose I/O pins.

USB/UART Bridge – to bridge the gap between the micro-USB input and microcontroller, we incorporated the use of the Silicon Labs CP2104 USB-to-UART Bridge. This device features an integrated USB transceiver that passes USBD+/- communication signals from the micro-USB input to the microcontroller via UART protocols and vice-versa. This is essentially a plug-and-play device, no external or internal configuration needed, that simplifies the process of uploading code onto the microcontroller.

3.3V Linear Voltage Regulator – we used the AP2112 3.3V linear voltage regulator to provide power at a steady 3.3V to all major system components. As described earlier, a p-channel MOSFET determines the input to this device between the micro-USB input and LIPO battery input. With maximum output current of 600mA, output accuracy of +/-1.5%, load regulation of 0.2%/A, and line regulation of 0.02%/V, this device is plenty reliable enough for our design implementation.

LIPO Charge Management Controller – the MCP73831 LIPO charge management controller intelligently manages the passage of current from the micro-USB input (while connected) to the LIPO battery connecter, allowing the battery to charge while both sources

are plugged into the board. The programmable charge current, which ranges from 15mA to 500mA, is set to 196mA with use of a 5.1k resistor connected between pins 2 and 5 on the device and the regulated output voltage at pin 3, the battery terminal, is 4.2V. Pin 4 is directly connected to the 5V bus from the micro-USB input and acts as the direct input source for the device. Pin 1 is a status pin connected to an orange LED and 1k resistor, drawing current when the device is charging and powering the LED to show its status. When the LIPO battery is fully charged and reaches an output voltage of 4.2V, the charger ceases to pass current through and the orange LED is powered off.

1200mAh LIPO Battery – the LIPO battery used in our design is rated at 1200mAh with a C5 rating. With an output voltage ranging from 4.2V fully-charged to 3.7V depleted, this battery possesses a maximum charge/discharge current of 240mA which works perfectly with our set 196mA charge current. To calculate the absolute minimum expected battery life, we first determined the maximum current draw of all major digital components and added them together along with a 5mA overhead. This produced a theoretical maximum current draw of 50mA, most likely much higher than the actual current draw during standard operation. Using the 1200mAh rating of the battery and assuming 70% efficiency, a theoretical minimum battery life of 16.8 hours was calculated. This number is higher in practice, as our device does not constantly draw 50mA out of the battery, and greatly exceeds our initial battery life requirement specification.

Miscellaneous Components – a standard micro-USB surface-mounted connector and a 2-pin surface-mounted LIPO connector were both used as power source inputs on this device. For glove-based audio feedback for users, a simple piezo buzzer was implemented via direct connection to a dedicated analog output pin on the microcontroller.

C. Board Design

Our board was meticulously designed to meet the specifications that were originally set in place for Smart Gloves hardware. The final design iteration of our board in EagleCAD, which reflects the final layout of our physical boards, can be seen in Fig. 2.

Two of these boards were fabricated, with one for each of the user's hands. The boards were designed with dimensions of 34mm x 67mm to fall within our original specification of 90mm x 90mm. These dimensions almost perfectly align with our chosen LIPO battery, which measures 62.2mm x 35.5mm. To package both the board and battery together, a simple 3D-printed housing was designed. This housing, measuring 39mm x 72mm, contains a top plate with 3mm mounting holes on each corner that the PCB is attached to and a hollowed-out bottom plate that the battery fits snuggly inside of. The top plate is attached to the bottom plate via clear packing tape, encasing the battery inside while leaving a hole for the battery wires to reach around to connect with the PCB on top. Leaving the PCB exposed on top of the casing was an intentional design choice, as we wanted it to be visible for all project demonstrations and presentations. An actual consumer version of this device would fully encase the PCB to provide full protection.

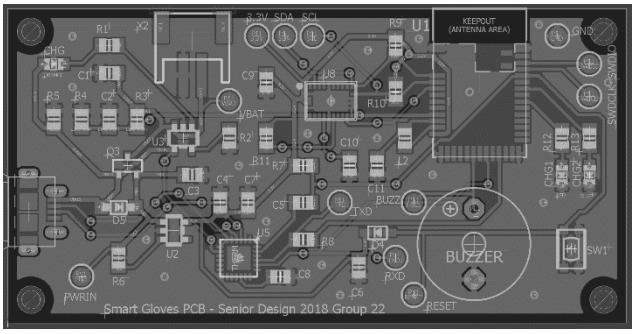

Fig. 2. Final Smart Gloves board layout, as seen in EagleCAD

The circuitry on the board was intuitively designed to take up minimal space, with components in similar subsystem groupings located as close as possible to reduce trace length. For example, referencing Fig. 2, almost all power regulation components/circuitry is located on the left third of the board, including both power source inputs and all regulation components such as the 3.3V linear regulator and charge management controller. Almost all digital components are located on the right half of the board, including the IMU and microcontroller. To determine minimum trace width, we calculated the minimum trace width necessary for the LIPO charging current, which at 196mA is the maximum current that would be passing through our board. Given standard trace parameters from IPC-2221, the required trace width for that current would be 1.67mils (thousandths of an inch). This trace width size is much too small for standard board manufacturers however, and as such we decided to use standardized trace widths ranging from 10mils to 16mils which would be plenty large enough for the charging current of 196mA and the maximum standard operation current of 50mA that was discussed earlier. For all major power buses, trace widths of 16mils were used to provide more than enough overhead for all current to pass through. Trace widths for digital buses such as the I^2C and UART buses ranged from 10mils to 12mils, more than enough for low-current digital signals to propagate.

One major design decision that simplified our board layout was the use of ground planes instead of manually

tying all component to a grounded source. Both the top and bottom layers were set as ground planes and connected at regular intervals with vias to ensure even grounding across the board. Any sections of the ground plane on the top layer that happened to be cut off from the rest of the plane by traces was properly grounded using vias to connect them to the bottom ground layer. Test points were implemented across the board at all major power and digital signal buses to provide ample room to properly diagnose and test any signals that might have behaved unexpectedly. Test points were also used for the flashing of the microcontroller; rather than incorporating a bulky JTAG connector into our design, the necessary pins were broken out as test points that wires were soldered to. Upon receiving our completed/populated boards, we realized that we had neglected to break out one of the necessary JTAG pins (SWO) on the microcontroller as a test point, but this was simply fixed by directly soldering a 30-guage wire onto the microcontroller pin itself. This was the only design flaw present in our board, otherwise it has functioned exactly as expected with no major problems or necessary reworks.

III. Hardware-Software Interface

The main communication channel for our board is over Bluetooth Low Energy through the microcontroller. There are two classes of devices in a Bluetooth connection. A central device (master) and a peripheral device (slave). A device classed as central can connect to multiple devices simultaneously and the slaves align themselves to the master clock intervals. A slave can only connect to a single device at time and is usually a peripheral (mouse, keyboard, ect). Each side communicates with the other during a connection interval at a minimum of 7.5 millisecond intervals called a connection interval. Deciding to use this technology would therefore guarantee a maximum of 133 updates per second in the ideal case. However, every platform isn't supported equally. For example, iOS 9.2 and iPhone 6, the smallest connection interval supported is 15 milliseconds while Android supports the standard minimum of 7.5 milliseconds. Due to this latency it is ideal that the majority of the actual leg work when it comes to data integration is done on the microcontroller.

Due to our design choices with Unity being our target platform some of our original intentions with Bluetooth were forced to be left by the wayside to accommodate for interaction with Unity. For example, we had originally anticipated higher throughput with Bluetooth communication by way of a more frequent sampling period however the unity libraries we were using to deploy the application abstracted out the Bluetooth protocols and set them all on single thread. This meant that we could only request a new sample manually once the callback for the last was called so queueing consistent communication proved to be less optimized than was anticipated with a native application. Another limitation from interaction with unity was with respect to actual data throughput. We had anticipated a 80-byte channel for each connection interval through initial prototyping however we were restrained to 20 bytes on the Unity application as that was the limit for a single characteristic in Bluetooth low energy.

To communicate with the Bluetooth sensor on the microcontroller the NRF52 actually has a custom Bluetooth stack already on-chip with accessor functions that have been nested into the Arduino environment through Adafruit's libraries. This allows us to quickly do things like Bluetooth configuration and set up scan for devices, so the actual communication side of things had been taken care of for the most part. We just needed to properly integrate it with our project by maintaining separate threads for the Bluetooth communication, BNO sensor readings and Piezo buzzer. We also had to package up the data and scale it properly since we were limited to 20 bytes as mentioned earlier. This meant scaling our Euler angles from degrees to 16 bit integers which didn't affect any precision or significant information loss since the BNO stores this data in 16 bit registers.

A communication message from the microcontroller consisted of a total 19 Byte message sent every connection interval regardless of a BNO update or not. Although for every interval we would have an update from the BNO as the sampling frequency we were using to sample from the BNO was around 100Hz which was higher than the 15ms minimum connection interval from communication with the phone application. Of these 19 bytes the first accommodated for the message flag. As of typing this we only have one which we are sending over which is the byte 0x4F or the letter 'O'. We utilized this as an identifier on the receiving end to ensure the message information we were about to take in was a valid set characteristic in the form we expect before deserializing the message. The following 6 bytes were the relative orientation of the sensor stored in degrees as an unsigned integer (scaled by 100). The final 12 bytes were the raw acceleration values read in from the IMU relative to the xyz of the sensor with gravity factored out.

TABLE 2: EXAMPLE OUTPUT COMMUNICATION

Message	Orientation			Acceleration		
Flag	H	P	R	X	Y	Z
1B	2 B	2B	2B	4B	4B	4B

The communication inbound to the microcontroller was simplified as we only expected one message from the

application. If we received the byte 0x42 or 'B' from the application, we turned the buzzer on for a preset interval of 300 milliseconds. We chose 300 milliseconds because according to studies it takes approximately 280 milliseconds for an average person to respond to an auditory stimulus [1]. This would provide the user with enough time to react and correct their form and not waste the Bluetooth pipeline by unnecessarily continuously sending redundant data. On the phone application side of things, we would refresh the message every 200 milliseconds so the buzzer would not stop if the user had still not corrected their form.

To communicate with the BNO we are utilizing I^2C communication abstracted out with Adafruit libraries. Thankfully this means we utilize general function calls to request the data and set calibration values to the BNO. Which saved development time on initial configuration and prototyping.

The BNO has several useful features including a Gyroscope, Magnetometer, and Accelerometer. For our purposes we do not require use of the magnetometer as the calculations for the exercise we utilize are relative to the gloves and not absolute. Although it's worth noting that in the future it might be worth considering identifying the users gloves in relation to each other. As such, for our purposes we set the magnetometer to off and solely utilize the Accelerometer and Gyroscope.

The BNO has several operation modes we attempted to utilize where it attempts to fuse outputs together to produce a more accurate measurement. However, the fused output proved unreliable for our case as the Euler angles the BNO put out fluctuated greatly along the extremes of the rotational axis. While this output would do fine for the case of aircraft where the user doesn't expect rotation beyond 45 degrees, for our purposes it was unreliable as the user's orientation can vary greatly over the course of an exercise. We had also originally planned to integrate the accelerometer data to attempt to track positional data with the gloves, however the accelerometer output proved unreliable for meaningful integration. Therefore, we decided to still utilize the accelerometer data for repetition counting as an additional feature and to integrate the gyroscope data which is updated at 100Hz in degrees/second to track rotational data. The integration of the gyroscope data proved mostly reliable for our purposes.

IV. Software Design and Data Processing

A. Microcontroller Software Flow

Upon initial startup the microcontroller sets up the BNO and Bluetooth units for communication by first configuring the Bluetooth into peripheral mode with our UART profile enabled. The BNO is then enabled by setting the magnetometer off and attempting to restore the calibration data to gloves from a file saved on the microcontroller. If the file is not found the microcontroller will enter a calibration mode where it will expect the user to rotate the device along each axis until each active subunit (accelerometer and gyroscope) is calibrated. Once calibrated this information is stored on chip so future calibration is not necessary. Although it is worth noting that regardless of this calibration activity the BNO continuously calibrates during operation attempting to improve measurement accuracy. This continuous calibration, which includes factoring out the gravity vector from the accelerometer, is perhaps why the accelerometer data is so noisy due to incomplete elimination of the gravity vector as the device is rotated.

Once the BNO and Bluetooth units have been setup the unit enters a normal operation mode where we have three continuous threads running. These are for Bluetooth, BNO data fetching, and Piezo buzzer response respectively. The Bluetooth thread will remain inactive until the unit manages to maintain a connection with a central device. Once connected this thread will package up data from the BNO and send it over to the Bluetooth as well as respond to any messages received from the phone application such as setting up the timer for piezo buzzer.

The BNO thread will, once called, retrieve and store a accelerometer sample from the BNO then call an update function for the integration of the gyroscope data. We utilize the Trapezoidal rule [2] to integrate the gyroscope data from the BNO and dynamically update the period every 100 samples to maintain accuracy. We confirmed that we have received approximately 100 samples every second which matches up nicely with the BNO's 100Hz refresh rate so sample loss is minimized.

The piezo buzzer simply checks to see if the Bluetooth thread has setup the timer for it to go off. If the timer is greater than zero the microcontroller will configure the PWM (pulse width modulation) to output to the microcontroller until enough time has elapsed. After the timer drops below zero the microcontroller will stop outputting to the speaker.

B. Boundary Checking

The sensors on the Bosch BNO 5055 detect orientation in a range of 0-359 degrees. To determine if the sensor is currently in bounds, a simple check is done to see if the current orientation is within the boundary offset. When the exercise first begins, the current orientation is sampled and that becomes the reference orientation. The result is that the exercise tracking is then done relative to this reference orientation. As for determining if any subsequent

orientation lies inside the boundaries, we need to do a little math.

There is an interesting challenge that arises when checking if we are in bounds and the boundary offset crosses over the 0/360 degree threshold. An example being a reference angle of 350 degrees (represented by a solid line) and a boundary offset of 20 degrees (the boundaries represented by the dotted lines). These parameters make a lower boundary of 330 degrees and an upper boundary of 10 degrees. The following example illustrates the problem and our solution.

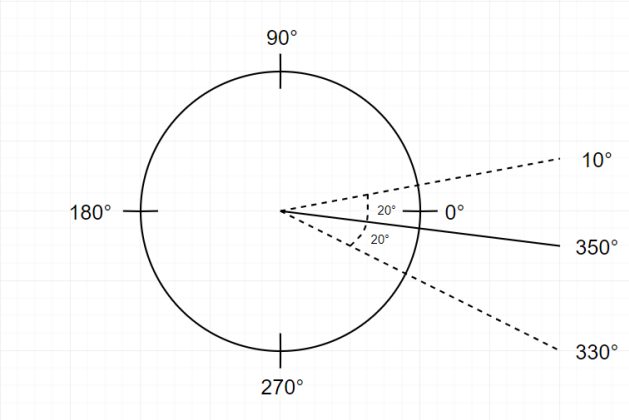

Fig. 3. Boundary Checking Diagram

It is difficult to do a mathematical comparison to see if the current orientation is between 330 degrees and 10 degrees as the value comparison isn't straight forward. Numerically, 330 is a greater value than 10. But in our situation, 330 degrees is the lower boundary. To get around this issue, we compare the current orientation with the reference orientation to see the difference between them is greater than the boundary offset.

To do this, we take the maximum value of the reference orientation and the current orientation and call it High. In a similar vein, we take the minimum value of the reference orientation and current orientation and call it Low. From here we need to have some temporary values called A, B, and C.

$$A = High - Low \quad (1)$$
$$B = 360 - High + Low \quad (2)$$
$$C = \min(A, B) \quad (3)$$

Values A and B would keep track of how many degrees we have traveled since locking in our reference orientation. Value C would be the minimum values of A and B. We are looking for the smallest travel distance from the reference orientation as this leads to the proper mathematical comparison with the boundary offset. If the value of C is less than the boundary offset, we are in bounds. If the value of C is greater than the boundary offset, then we are outside of the boundaries specified for the exercise.

C. State Machines

A state machine is a device that can be in one of a set of stable conditions depending on its previous condition and on the present values of its inputs. For Smart Gloves, everything from the stages of the exercise to the motion tracking of the hands are implemented and tracked using states.

The hands are tracked by determining if the hands are moving in an upward or downward motion with relation to the user. The various stages of the exercises are Pre, Set Up, In Progress, Rest, and Finish. Depending on user input during the exercises, the states of the various state machines are changed accordingly.

D. Repetition Counting

The repetition counting of the gloves are done through a combination of acceleration thresholds and the state machine tied to the motion of the gloves. If the sign of acceleration changes and the magnitude breaches a certain threshold, then we can change the motion state of the gloves from upward to downward and vice versa. Once the sign of acceleration changes as the hand motion return to its original position, then we increment the repetition counter.

V. Companion Mobile Application

In order to provide the user with an interface to customize their experience with the Smart Gloves, a companion mobile application was created. The mobile application was designed to be intuitive and easy to use. Users are able to select exercises to track, edit exercises and their metrics, and also connect and disconnect their Smart Gloves. The mobile application is an important part of the user experience as it provides the user with insights on what the Smart Gloves are tracking.

A. Mobile Application Design

Throughout gathering requirements for the application and coming up with solutions on tracking data from the Smart Gloves, the decision to begin development with the Unity game engine was made.

Unity allows the development of cross platform applications. This means that the application can be built to run in Android and iOS devices. The programming language used to develop the companion app was C# (C Sharp). C# is an object-oriented programming language that allows the use of classes to model objects along with

four of its main features – encapsulation, abstraction, inheritance, and polymorphism. Along with object-oriented features, the Model-View-Controller (MVC) design pattern was used. This design pattern is used to separate the concerns of the application. A model represents an object carrying data. In the case of the Smart Gloves, an example of a model would be the exercise list. This list is an object that carries the name of the exercise, the number of sets, start timer, rest timer, and goal reps. Since the app contains multiple pages, there are multiple views the make it up. There is the home page, settings page, edit exercise page, and many other pages. Views contains UI elements that users can interact with. The elements can be buttons, text fields, labels, and pictures. Lastly, there is the controller. The controller acts on both the model and the view. Controllers control the data that flows through the models and updates the views when data changes. The use of this design pattern was not intentional in the beginning but as more features were added to the application, using the MVC design pattern showed to be a great organizational tool and great way to structure the application's code.

B. Applied Background Knowledge

Skills and knowledge acquired through required courses were applied to the development of the Smart Gloves companion application. Fundamental concepts such as variables and constants were used to help facilitate the manipulation of data. Alongside variables and constants, control flow statements such as for loops, if-else statements and switch statements were used. Switch statements showed to be a great tool to use along with enumerations. Enums, short for enumerations, is a data type that consists of a set of named values. Enums were used to correctly figure out in which stage of the exercise the user is in. These stages can be setup, in progress, rest, and finish. Apart from exercise stages, enums were also used for correctly setting the exercise the user is working on. Through the use of switch statements, the correct parameters can be set allowing for cleaner and readable code. Data structures also showed to be extremely helpful to group and organize related data. Arrays and hash tables are the main data structures used. Though fundamental, these data structures provide great runtime. Furthermore, methods were written to decrease the level of code repetition while maintaining the responsibility of each method to be specific and simple. Doing so follows the separation of concerns principle. Following this design principle gave the benefit of simplifying development and code maintenance. In the cases where debugging and troubleshooting were needed, separation of concerns made it very clear and simple. The use of design patterns and best practices learned throughout the different computer science courses helped save a lot of time and decreased the number of bugs that were encountered.

C. Mobile Application Details

When the application is started, the user is introduced to the home page. The home page contains our Smart Gloves logo, along with three buttons used for navigation. The three buttons are Select Exercise, Edit Exercise, and Settings.

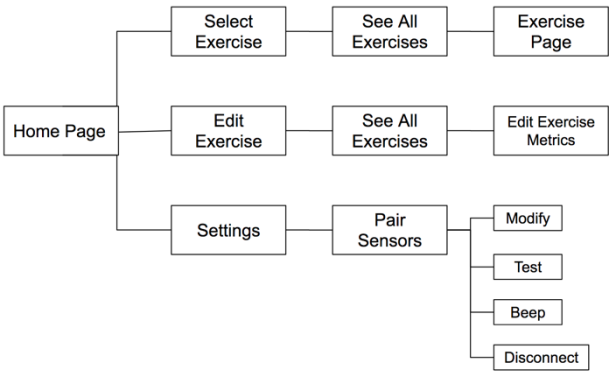

Fig. 4 Mobile Application Context Diagram

The Select Exercise button takes the user to the Select Exercise page where they are able to select the exercise they wish to complete. The See All Exercises button creates a drop-down list that the user can interact with and select the desired exercise. Once the user selects the exercise, the user is taken to the Exercise Page where the current stage of the exercise routine is shown. Alongside it is the timer that will count down on the rest and start timer and also count up on the time to measure how long the user takes to complete the exercise. This page also consists of a left and right cube. Each correlating to the left and right sensor on the gloves. These cubes are programmed to provide a visual representation of the glove's movements. Next follow the goal statistics. These goal stats display the metrics the user set to achieve when following a particular exercise. As described before these metrics are number of sets, rest timer, start timer, and reps per set. Lastly, is the current statistics section. This section displays the current set the user is in while also displaying the current number of reps for the particular set. None of these text field are editable by the user. The current statistics fields are updated data retrieved from the gloves while the goal statistics can be updated through the Edit Exercises page.

With that being said, the next option available in the home page is the Edit Exercise page. Like the Select Exercise page, the Edit Exercise page contains a See All Exercises button that displays a drop down with all the available exercises to be edited. After the user selects an

exercise, the View Exercise Details page is shown. This page contains the exercise name, the number of sets, start timer, rest timer, and goal reps. These fields can all be edited and stored by tapping on the save button. It is important to note that only integer values are to be entered in all text fields except for exercise name, which can be consisted of characters and numbers. To guard the user from errors, an error message is displayed while also highlighting the field with incorrect data. The values entered in this page become the goal metrics the user will want to achieve and will be displayed in the Exercise page.

Lastly, the Settings page contains a Pair Sensors button. This setting displays which glove is connected to the app. It also allows to sound the buzzer to provide feedback that the glove is indeed connected. For troubleshooting purposes, the modify button provides the user with detailed data such as x, y, and z axis orientation collected from the sensors. This allows to troubleshoot if the sensors need to be calibrated.

To conclude, each page contains a unique background image that relates to exercising and weight training. These images were used to theme the application and give a more personal look. The images used in the app are all free to use from the Unsplash website. Linked in the resources section, all the photos in the Unsplash website are free to use and do not require permission from or require providing credit to the photographer. These is explained in further detail in the license page of their website. Along with background images, the pages in the application contain black translucent buttons with white text. These colors were chosen since they contrast well with the colors in the background images. Lastly, all the main screens of the application contain a small info section that displays the current connection status of the gloves. This was designed to provide the user with quick feedback on whether the gloves are connected and therefore take action if needed.

VI. Conclusion

All aspects of hardware and software design mentioned in the previous sections of this report merge together to form a cohesive final prototype of our originally conceptualized Smart Gloves. With further development, this idea could potentially become a fully-fledged product that would provide a structured boost to user workouts of all varieties.

Luis Gamarra Jimenez is a senior computer engineering student at the University of Central Florida, currently working as a part time Software Engineer at Disney. Upon graduation, he will be start a full-time position with Disney focusing on mobile development and testing.

Andrew Smith is a senior electrical engineering student at the University of Central Florida. Upon graduation, he will be starting a job with Harris Corporation in Melbourne, FL as a part of the Space and Intelligence Systems Division.

Jason Surh is a senior computer engineering student at the University of Central Florida. During his time in college, he interned at Lockheed Martin as a Software Developer. After graduation, he will start a job as a Program Manager for the Xbox division of Microsoft in Seattle.

Scott Suarez is a senior Computer and Electrical Engineering student at the University of Central Florida. During his time in college, he interned at Voith Hydro as a Software Dev. After graduation, he will start a job as a Software Developer for Bing Ads.

VII. References

[1] -Aditya Jain, Ramta Bansal, Avnish Kumar, and KD Singh, *A comparative study of visual and auditory reaction times*, Int J Appl Basic Med Res, 2015

[2] - Bourne, M. (2015). *Trapezoidal Rule*. [online] Intmath.com. http://www.intmath.com/integration/5-trapezoidal-rule.php [Accessed 18 Feb. 2018].

[3] - *License, Unsplash.* [online] https://unsplash.com/license [Accessed 01 April 2018].

[4] - Unity Technologies. (2017) *Unity User Manual.* [online] https://docs.unity3d.com/Manual/index.html [Accessed 19 March 2018]