brACE

Team ACE
Alexandrea Mellen : almell@bu.edu
Chris Feldman : feldman3@bu.edu
Emily Lam : emilylam@bu.edu

Abstract

We implemented a slouch detection wearable that encourages proper posture through negative reinforcement. Our device detects when the user slouches and sends that information to an Android App via Bluetooth. The Android App alerts the user through a loud alarm and notification on the phone that he or she is slouching, and keeps track of the number of times the user slouches per day. The alarm will continue to beep obnoxiously until the user stands up. Once the user stands up, the gumstix recognizes this and sends a message to the Android App to silence the beeping. The inconvenience of having to stand up whenever the user slouches will motivate the user to slouch less, resulting in better back posture.

1. Introduction

We are Team ACE, and for our EC535 Final Project we used a Gumstix board to build a slouch detection wearable that we call brACE. Our device works to improve work flow, especially for people who sit in front of a computer all day. Whether its leaning back, leaning forward. or hunched shoulders, the most comfortable seated position is usually some sort of slouch. The most comfortable position, unfortunately, is not always the best position. Bad posture, visually marked by a slouched forward back and protruding head position[1], can result in back and neck pain and even something called Upper Cross Syndrome[2], which can cause joint dysfunction as well as weakened muscles in a person's neck and back. With an increasingly white collar work force, combatting slouching and other bad posture habits is more important than ever.

Our brACE device looks to tackle this problem head on, reinforcing better posture habits by notifying users through their Android device when they are slouching and then forcing them to stand before sitting back down to fix their posture and continue working. The brACE is worn around a user's neck, sitting flush against them, high on their back. The device uses an accelerometer to detect when a slouch begins and then via a bluetooth connection to the user's Android device sets off an alarm that will not silence until the accelerometer detects that the user has stood up. Once the alarm has ceased, the user can return to work, pressing the button on the device to reset the slouch detection algorithm. We believe that our system utilizes the perfect amount of negative feedback to discourage the user from continuing to slouch.

[1] http://www.backandneck.ca/the-negative-effects-of-slouching/
[2] http://www.muscleimbalancesyndromes.com/janda-syndromes/upper-crossed-syndrome/

Originally the plan was to not silence the alarm until the accelerometer detected long term movement on behalf of the user. The intention here being that the user would be punished by the system by needing to take a walk every time he or she slouched. In the end, we scrapped this idea in favor of simply standing up to end the alarm. We thought that forcing users to walk around every time they slouched would be too detrimental to workflow for a device intended to improve health and productivity. Instead, standing up provides a lower level of negative reinforcement, but should not cut into a user's normal routine. We feared that forcing the user to take a long walk would become frustrating, and while we need the system to implement negative reinforcement we also need users to want to continue using the device.

The final product is a great representation of our initial vision. We are able to connect and pass information to a customized Android application via bluetooth, as well as detect both when a user begins slouching, and when they stand up to correct for this poor behavior. The biggest drawback to our final system was that we were not able to bypass login on boot and run our software automatically. Doing this would have allowed us to let a user power on the device and then simply connect to Bluetooth from their phone and begin using brACE without ever seeing the command line. Unfortunately, when trying to implement this during the late stages of development, we ran into a problem where the Gumstix stopped booting at all. Once we replaced our Gumstix, it was too late to continue trying to implement the auto login so our current prototype must be powered on and started from the command line. This does not limit the functionality of our device, only the convenience. If we were to continue development, making sure that you could power on and use the device without Minicom and the command line would be a top priority.

2. Design Flow

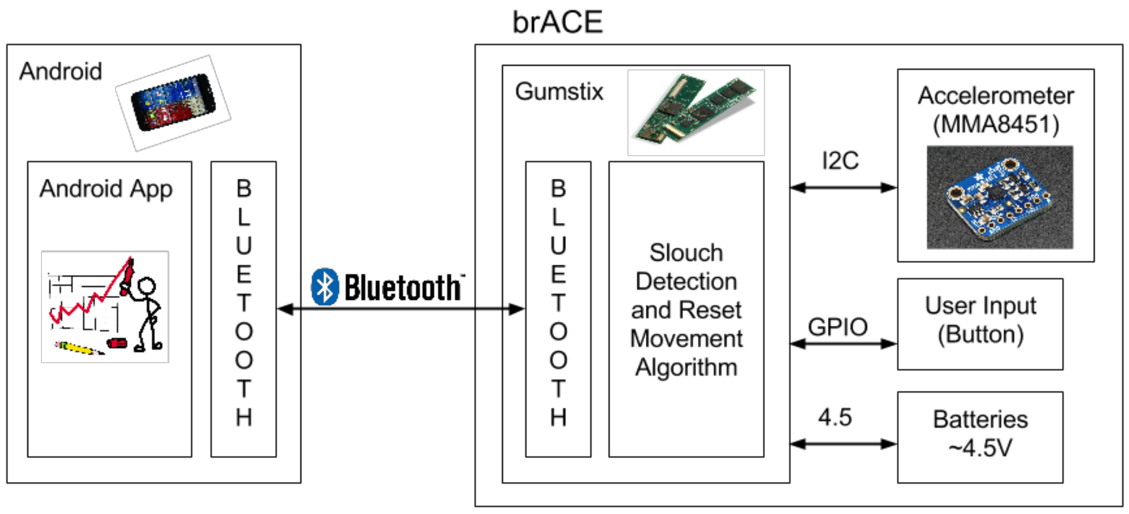


Figure 1: Four wire connection between the Gumstix and Accelerometer with 4.7kOhm pull-up resistors on the SDATA and SCLOCK line.

Our project implemented Gumstix Bluetooth communication, slouch and standing algorithms, accelerometer integration, mobile power management, a button module, an Android App, Android Bluetooth communication, and a 3D printed case. The Gumstix Bluetooth communication interacted directly with the Android Bluetooth communication and the slouch and standing algorithms. The accelerometer interacted with the slouch and standing algorithms and the button module. The mobile power management was integrated with the Gumstix board. Lastly, the Android App mingled with the Android Bluetooth communication, and the 3D printed case was completely dependent on the hardware assembly. Chris implemented the Gumstix Bluetooth communication and the button module, while working with Emily on the slouch and standing algorithms. Emily also implemented the accelerometer and the mobile power management. Alexandrea implemented the Android App, the Android Bluetooth Communication, and the 3D printed case.

3. Project Details

a. **Button Module** *(see kernel/mygpio.c and kernel/Makefile)*

We implemented the button into our design to allow for a user initiated reset of the device. The utility of this is two fold, the button allow for user input and also causes a restart of the slouch detection algorithm, saving the user's upright position on every press. The implementation was done with a simple kernel module written in c. On boot of the gumstix, we ran a start script that implemented the device node and inserted the module. Then once the device was connected to Bluetooth, we ran the user level program that would talk to the kernel module through SIGIO. Each button press throws an interrupt in the module, which sends a signal to the user level. The user level code's signal handler, running the slouch detection algorithm, resets the user position variables and sets the slouch condition to false. This is all we needed the kernel module to do because we had no other user interaction with the Gumstix, all other user interaction is done via the Android phone. The information sent to the phone is done at the user level.

b. **Bluetooth Implementation** (see /bluetooth/bluetooth.c and /bluetooth/Makefile)

Gumstix handles Bluetooth natively, which makes connections fairly simple. As long as the connection is initiated from the Android device, the Gumstix connects automatically. The issue that we ran into with this was that once the connection was established we could not run any user level programs until we used ctrl-c. This was a problem because though it did not come to fruition, we planned to have the system run our code automatically on boot and a connection to Bluetooth is required for our algorithm to run correctly. Had we used to the native implementation of Bluetooth we would not have been able to run our slouch detection algorithm after a connection. To combat this, we used a server program written in c that opened am RFCOMM socket to listen for Bluetooth connections[3]. This set up the Gumstix to

[3] http://uwf.edu/bowsnickiklewe/lac/docs/gumstixbluetooth.pdf

connect to any any incoming connection requests, just like it would natively, however in this implementation we were able to connect in the background so that we could still run user level programs. Once the devices were connected, sending data was as simple as writing to the /dev/rfcomm0 node.

A working Bluetooth connection took us a long time to actually correctly implement. Quickly we were able to pair the Gumstix with the Android phone, but establishing an actual connection eluded us for a long while. The sample server program that we were working off of opened a socket and listened on Port 1. This was fine in theory, but with the other groups working in the lab at the same time, there was a number of devices trying to connect all at once and it caused us to accept random bluetooth connections from devices other than our own. We did not understand that it was the common port that was effecting us until we talked to another group that had solved a similar issue. A fix was as simple as changing the port we were listening for connections on to something less common. In the end we switched between ports 2 and 3, using which was less popular at the time. We also struggled for a long time to make this work without using the echo command. In time we realized that the message would only send with a terminating newline character at the end. It was actually quite interesting, once we sent a message with the newline character, the Android device also received all of the messages we had attempted to send previously.

c. Interfacing the Accelerometer *(see i2c/myi2c.c, i2c/myi2c.h, i2c/MMA8451.h)*

The accelerometer used in brACE is the MMA8451 by Freescale Semiconductor. This accelerometer was chosen because it provides a 14-bit resolution in the range of +/-2G, which is optimal in measuring small changes in acceleration. We were able to get the MMA8451 on a breakout board from Adafruit. The MMA8451 communicates with other devices via I2C. The gumstix verdex supports I2C natively, with the HIROSE breakout board breaking out the I2C-0 bus. The I2C bus and module are pre-initialized on bootup.

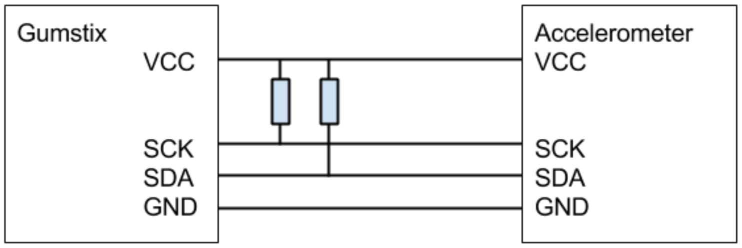


Figure 2: Four wire connection between the Gumstix and Accelerometer with 4.7kOhm pull-up resistors on the data and clock line.

The first issue we ran into with the accelerometer was getting the accelerometer to be responsive to the commands the Gumstix was sending it. Since the I2C protocol was natively implemented on the Gumstix, we thought sending a command would be as easy as writing commands to the I2C buffer. However, the accelerometer used a repeated start I2C protocol instead of the regular I2C protocol natively implemented on the Gumstix.

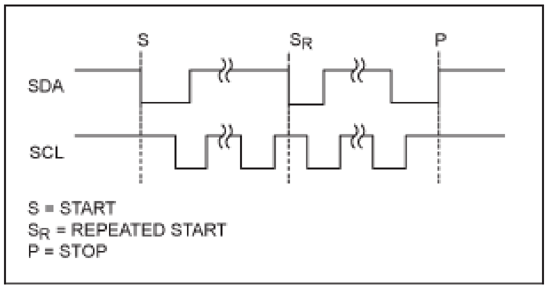


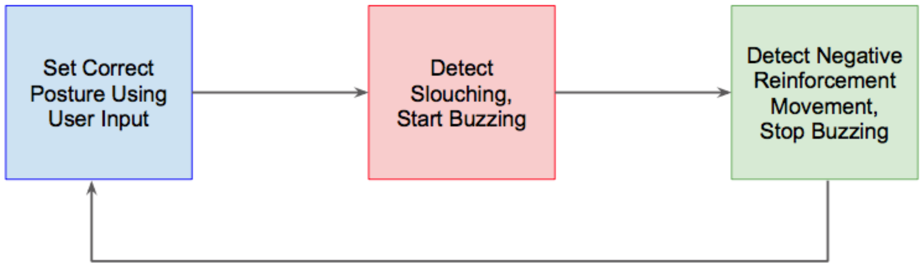
Figure 3: The repeated start I2C protocol differs from the regular I2C protocol in that a STOP condition is not needed before sending the START condition for a new sequence. This allows for the sending and receiving of data very quickly.

We solved the repeated start I2C protocol by using the the I2C_RDWR ioctl command instead of a regular read/write when sending commands[4]. This allowed for mixing read and write commands together, so that we could read the registers we wanted on the accelerometer by sending the command. We tested this by sending the read WHOAMI command, and the accelerometer successfully sent back the ID of the accelerometer.

Our next issue was getting the accelerometer to read the entire two byte data of the x, y, z, acceleration data. The acceleration data is 14 bit, where the first eight bits are the MSB and the next 6 bits are the LSB. We were getting the correct data, however bytes were dropping. We soon realized the issue was in not declaring our buffers to the appropriate sizes for receiving more than one byte. After that, we were able to successfully parse the two bytes into a float value in the +/- 2 range. The MMA8451 also provided a lot of additional functionality we didn't use such as orientation detection.

I2C code was written in a very modular fashion. The commands/registers required to interact with the accelerometer were pulled from the MMA8451 datasheet and defined in MMA8451.h[5]. Next, commands were written in myi2c.c and myi2c.h to interact and parse the received data. We referenced the Adafruit's MMA8451 library when writing this code, but their code is different from ours, they wrote their code for Arduino and we wrote ours for Gumstix[6]. The commands in myi2c can then be called by the main.c file.

d. Slouch Detection and Reset Movement Algorithm *(see i2c/main.c)*


Set Correct Posture Using User Input
Detect Slouching, Start Buzzing
Detect Negative Reinforcement Movement, Stop Buzzing

[4] http://gumstix.8.x6.nabble.com/I2C-Gumstix-read-write-td576626.html
[5] http://www.freescale.com/files/sensors/doc/data_sheet/MMA8451Q.pdf
[6] https://github.com/adafruit/Adafruit_MMA8451_Library

Figure 4: The basic idea to our slouch detection and reset movement algorithm is three phase.

The first phase is to set the acceleration values for correct posture. This step is accomplished by having the user push a button when they are in a proper position. The acceleration values for this position are then saved as the default posture position. This is important because we don't want our system to rely on a hardcoded default position since the acceleration values can change depending on location, elevation, the user, and the way the device is placed on the user's back. So it was vital to our implementation that this value could be changed. Next, deviation in only the x-axis is compared to the default position's x acceleration value -- x-axis because the accelerometer is placed in the casing to always point the x axis downward. If the deviation is larger than a preset threshold, the algorithm will alert the Android App. The Android app will then notify the user that he or she is slouching by turning on an alarm. The gumstix on the other keeps checking to see if the user has stood up or not. This is checked by checking again the acceleration in the x-axis. A stand up movement is determined by checking if the x_acceleration is greater than a preset deviation for an extended period of time. This is checked using a count variable. The count is used so the user cannot activate a stand by just moving around in his or her chair. The user needs to actually stand up for the the alarm to turn off. Once, the user stands, the gumstix alerts the android app, which in turn turns off the alarm. The system know waits for the user to press the button again to restart the cycle when he or she sits down.

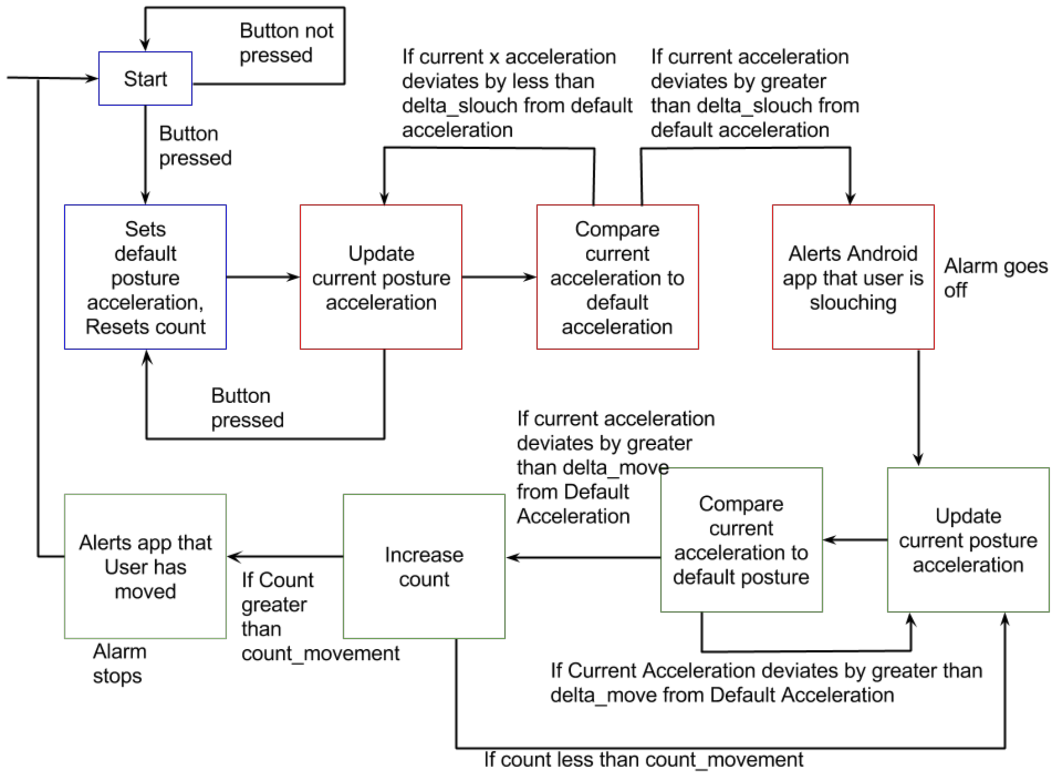

Button not pressed
Start
Button pressed
If current x acceleration deviates by less than delta_slouch from default acceleration
If current acceleration deviates by greater than delta_slouch from default acceleration
Sets default posture acceleration, Resets count
Update current posture acceleration
Compare current acceleration to default acceleration
Alerts Android app that user is slouching
Alarm goes off
Button pressed
If current acceleration deviates by greater than delta_move from Default Acceleration
Alerts app that User has moved
Increase count
Compare current acceleration to default posture
Update current posture acceleration
If Count greater than count_movement
Alarm stops
If Current Acceleration deviates by greater than delta_move from Default Acceleration
If count less than count_movement

Figure 5: Detailed state diagram for the implementation of the slouch detection and reset movement algorithm

e. Android App and Android Bluetooth Communication *(see brACE_AndroidApp.zip)*

We were originally planning on making an iOS App for the project, however, we ran into issues with Apple's Bluetooth requirements. The gumstix boards run a Bluetooth 2.1 profile, which is restricted under Apple's regulations. There are five standard options when dealing with this issue.[7]

1. MFI (Made for iPhone) Program
    a. Apple has a licensing program available to create AirPlay accessories which provides a lot of tools for developers to make hardware-software integrated apps, however, typical cost for the program starts at $10,000 and requires an application process. Therefore, this option is not viable for this project.[8]
2. CoreBluetooth Framework
    a. This public framework is only for use with BLE devices. Since the gumstix boards are not BLE, we cannot use this framework.[9]
3. GameKit Framework
    a. This public framework provides some very easy to use, convenient bluetooth features. When we implemented it on an iOS device, setup was very straightforward. However, the system was not capable of recognizing the gumstix board. As it turns out, the GameKit Framework is only available for communication between two or more iOS devices.[10]
4. Bluetooth Manager Framework
    a. This private framework is able to recognize devices other than iOS and gather basic information about them. When we implemented it, we were able to not only identify the gumstix board on the iPhone, but we could also find information about it such as the name, major and minor number, and address. However, we were unable to connect and send messages to and from the gumstix board. This framework is made for the External Accessory Program (through Apple), which allows the developer to connect through Bluetooth to certified devices. A certified device is considered one which has their authentication chip, and unfortunately the gumstix are not certified, so this is not a viable option.[11][12]

[7] http://www.pocketmagic.net/2012/07/bluetooth-and-ios-use-bluetooth-in-your-iphone-apps/#.VHOJUoslA_8
[8] https://developer.apple.com/programs/mfi/
[9] https://developer.apple.com/Library/ios/documentation/CoreBluetooth/Reference/CoreBluetooth_Framework/
[10] https://developer.apple.com/LIBRARY/ios/documentation/GameKit/Reference/GameKit_Collection/index.html
[11] http://stackoverflow.com/questions/23579226/connecting-to-other-bluetooth-device-by-beetee-app
[12] https://github.com/michaeldorner/BeeTee

5. Jailbreaking
    a. The final option for connecting via bluetooth to a gumstix board is to jailbreak the iPhone and connect with something such as btstack. Btstack allows iOS devices to connect to any bluetooth device. We have a jailbroken iPhone available, so this may be a viable option. However, if we were to bring this product to market, this would not be a viable alternative and we would be more inclined to explore option (1).[13]

In order to get around these issues, we decided to switch platforms to Android.

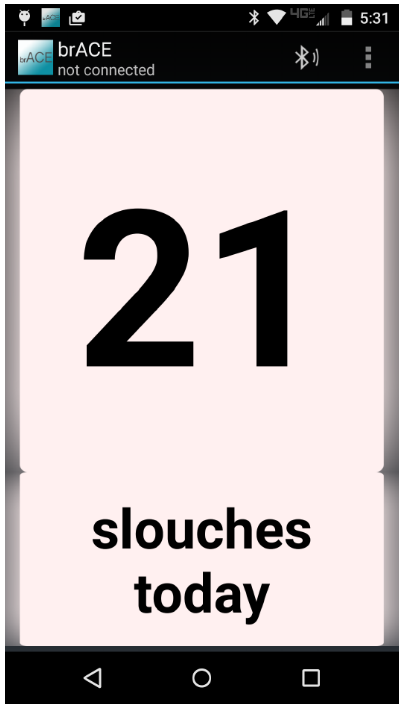


Figure 6: brACE Android App

The Android App implemented Bluetooth, saving state, sharing, and UI design. We were able to successfully connect and send messages to the Android device from the Gumstix Boards via Bluetooth. The message coming in to the Android App let the Android App know whether to increment the slouch counter and turn on the beeping noises or shut off the beeping noises (aka, if they slouch, mark it as a slouch, or if they have movement, shut off the alarm to tell them to move). From there, we saved the slouch counter value and the current date in the app, so that even if the app was opened again later, the user could still see the number of times they slouched that day. Sharing was implemented so that the user could share their slouch progress with their friends.

[13] https://code.google.com/p/btstack/

The app was created entirely in Java and XML. We used the Android Studio Beta as our IDE for the app, which came with its own set of challenges, as we had to download many libraries and get familiar with a foreign IDE.

The bluetooth connection was created through the phone via the Android Bluetooth API[14]. It checks to make sure the phone has bluetooth, then opens a RFCOMM socket to connect to the gumstix boards via a custom thread. Once we have established a connection, we remove the thread and begin anew with a thread dedicated to the connection. We wait for messages to come from the gumstix, and when we receive them we send them to a custom class that handles the UI. From there, we parse the message for the first character, which differentiates between a slouch (‘s’) and a reset (‘r’). If we are sent a slouch, we increment the slouch counter, update the UI, and set a notification to go off with an alarm. If we are sent a reset, we shut off the alarm. The slouch marks when they slouched, and the reset marks when they stand up to shut off the alarm. We also save state on close of the Android App, so that when the user closes the app they can still keep track of the number of slouches they have had that day. To do this, we save the number of slouches and the current date on close. When the user opens the app, the system checks to see if it is the same day, and if so sets the slouch counter to the stored value. If not, it sets the slouch counter back to zero. The UI was made to highlight the users current number of slouches. If the user slouches, the UI updates to a darker red color which is proportional to the slouch counter. If the screen is dark red, the user knows they have been slouching too much.

f. Mobile Power Management

The Gumstix Verdex runs on a 3.3V logic level. What we did was split a mini-B usb cable and connected only the power lines to 4.5V. The Gumstix’s onboard voltage regulator then drops the 4.5V down to the 3.3V voltage level required for the Gumstix. The 4.5V is provided with three alkaline AAA batteries (1.5V * 3 = 4.5V). The data lines on the mini-B usb are not connected.

[14] https://github.com/Sash0k/bluetooth-spp-terminal

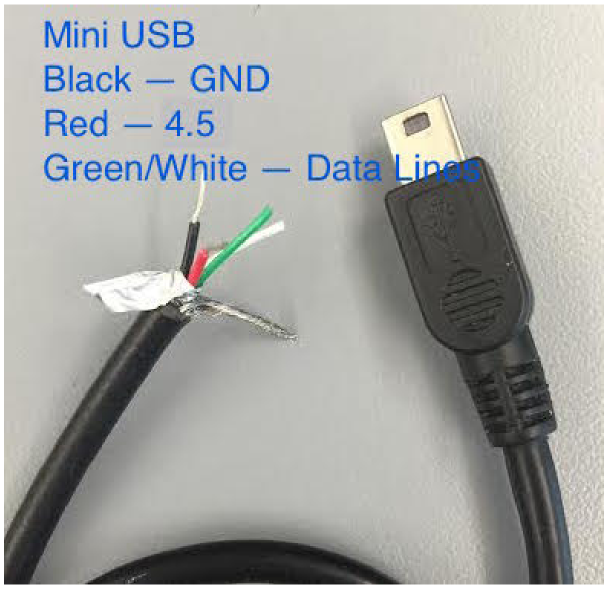


Figure 7: Wiring for a mini-B USB cable

g. Device Case

In order to make a case for the device, we made a 3D model using SolidWorks 2011. This model was sized to fit every component of the device within the case. We used a MakerBot printer in EPIC to 3D print the model. Unfortunately, the model did not take into account shrinkage from the 3D printer, so it was too small to fit all of the components. In order to resolve this issue, we considered multiple alternatives. We tried cutting large holes in the sides of the box to make more room for some of the pieces, and also considered just leaving out the battery pack or bluetooth antenna entirely. When we attempted this, we found that, even then, the system would not fit in the box. It stuck out the side, preventing us from securing the lid, and making it resemble a walkie-talkie. We were unable to print a new box, so we went a different route. Instead, we bought a project enclosure from Radioshack. We cut holes in the sides for the wire necklace, and attached the lid of the old case to the front of the project enclosure (so we maintained some elements of the past design). The case now fit the entire device assembly effectively.

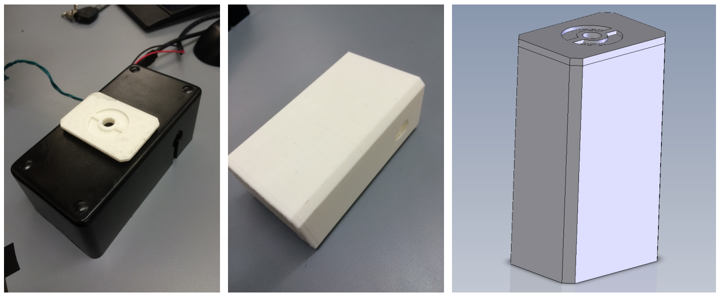


Figure 7: To the left is the final project enclosure. The middle is the 3D printed case. The right is the 3D Model of the 3D printed case.

4. Summary

Our slouch detection system was able to effectively identify when the user was slouching and when they stood up through our movement algorithm. It successfully connected via Bluetooth to an Android device and performed one way communication which alerted the user to when they were slouching. The Android app was able to notify the user via a notification and alarm system to when they were slouching, and keep track of the number of slouches per day the user had.

As far as the market is concerned, we have a ways to go. We would need to successfully implement the autorun feature and include a longer battery life in order to make this a viable product. It would also be difficult to sell at a good price point given the cost of the hardware components, although mass production could drive costs down. Though our final product comes in a finished black case, we would need to fine tune a case around the buttons and something to encompass the wire system that leads to the buttons. Our top competitor in the market, the Lumo Lift[15], is much smaller and less intrusive than our system. In order to compete with them, we would need to downgrade the size of the device and find a more user friendly way to wearing it.

One remaining challenge we would have liked to implement is to be able to keep track of several weeks of data within the Android app. It would be very nice for the user if we could have a chart system and keep track of weeks of their slouch data to show their progress over time. Another idea we had considered was to optimize the slouching algorithm by making use of all three degrees of motion and consulting a health professional. Our slouches are based on what we deem improper posture. However, a chiropractor may have other opinions. We would also ensure the system is capable of running from bootup. Currently, we need to establish a connection with minicom through the serial port with the system powered on to run the program. It would be much improved if we could autorun our code on boot of the Gumstix.

[15] http://www.lumobodytech.com