

# Game Controller for Nintendo Mike Tyson's Punch-Out!!

Jonya Chen, Mike Singer, Michael Wu

CS 294-98 / ME 290U Homework 3

## 1 Introduction

For HW 3 we decided to make a controller to play the classic Nintendo game, Mike Tyson's Punch-Out!! This game appealed to us because of its physical nature. One of the earliest interactive game controllers, the Nintendo PowerGlove, actually used the same flex sensors we were given for this project. That controller could even be used to play this very game. The concept of the game is simple: you play as the boxer Little Mac and compete against a variety of opponents. Your moves are: left punch (high and low), right punch (high and low), defend, duck, dodge left and dodge right. To control these inputs, we created a belt using flex sensors for dodging, and boxing gloves with accelerometers for all punching and defending. The physical belt was designed and constructed by Mike, the circuit and Redbear Duo sensor code was designed and written by Jonya, and the input mapping of the sensor readings to keystroke presses was written by Michael. All three contributed to soldering and wire management.

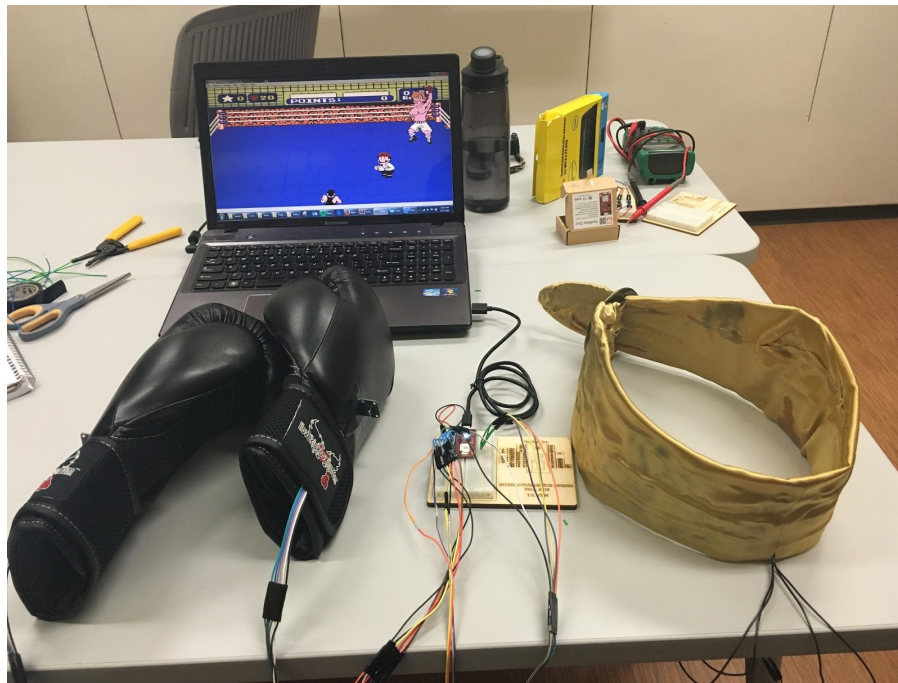


Figure 1 - Finished Controller with Game

## 2 Hardware Design

### 2.1 Circuit (Jonya)

For the gesture detection, we decided to use two Adafruit LIS3DH Triple-Axis Accelerometers to attach onto each boxing glove. Since we wanted to communicate information through the SPI interface, we then connected the accelerometer pins to their respective inputs on the RedBear Duo. Note that since we had 2 accelerometers, we needed to use both SPI and SPI1 hardware interfaces. The connections were made as follows:

Left Glove LIS3DH		Right Glove LIS3DH	
LIS3DH pin	Duo pin	LIS3DH pin	Duo Pin
Vin	VN	Vin	VN
GND	DNG	GND	GND
SCL (SCK)	A3	SCL (SCK)	D4
SDA (MOSI)	A5	SDA (MOSI)	A2
SDO (MISO)	A4	SDO (MISO)	D3
CS	A2	CS	D5

The flex sensors were fairly simple to wire up. Since each flex sensor acted as a resistor, in order to read the change in voltage, we just created a voltage divider circuit. Since the resistance of the flex sensor was around 10 k $\Omega$  when flat and 20 k $\Omega$  when flexed, we decided to go with a pull-down resistor of 15 k $\Omega$ . Therefore, we used the following equation:

$$V_{out} = V_{in} \left( \frac{R_{flex}}{R_{flex} + 15 \text{ k}\Omega} \right)$$

When testing the circuit and reading the sensors, we just breadboarded all the components. As a result, our circuit followed the schematic below:

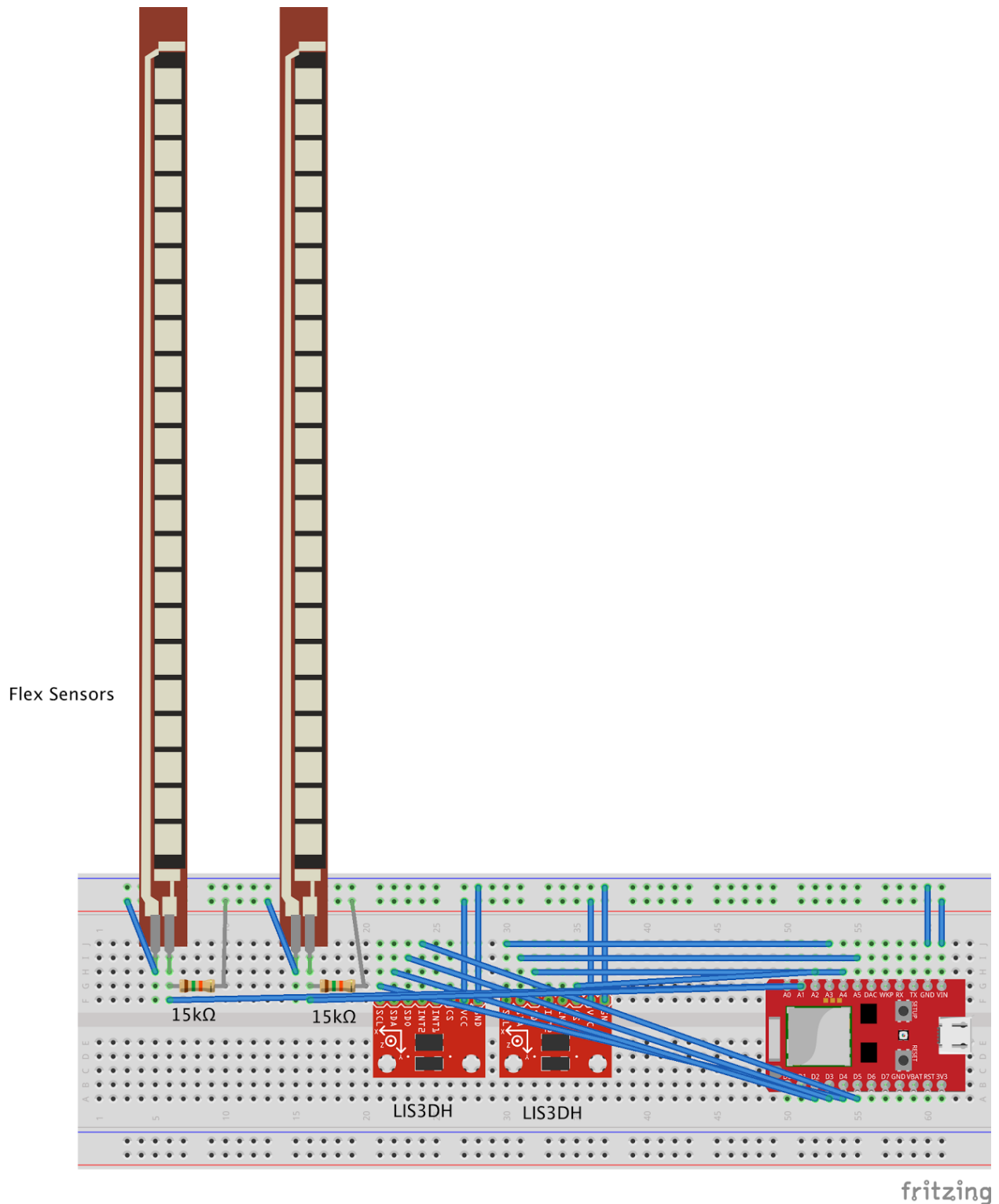


Figure 2 - Complete Circuit Diagram

Once we ensured that all sensor readings worked as planned, we then removed the sensors from the breadboard and designed enclosures for each, as detailed in the following section.

## 2.2 Creating the belt and gloves (Mike)

The physical controller consists of two parts: the boxing gloves and the belt. The gloves were used for all punching inputs as well as defending. They consist of off-the-shelf boxing gloves with an Adafruit LIS3DH Triple-Axis Accelerometer taped to each glove.



Figure 3 - Boxing Gloves and Belt

The belt was constructed using two long flex sensors attached to leather pieces, which were then attached to an off-the-shelf leather belt. The entire assembly was then encased in fabric to create the finished controller.



Figure 4 - Flex Sensor with Soldered Wires Positioned on Leather Piece





Figure 5 - Flex Sensor Glued in Place

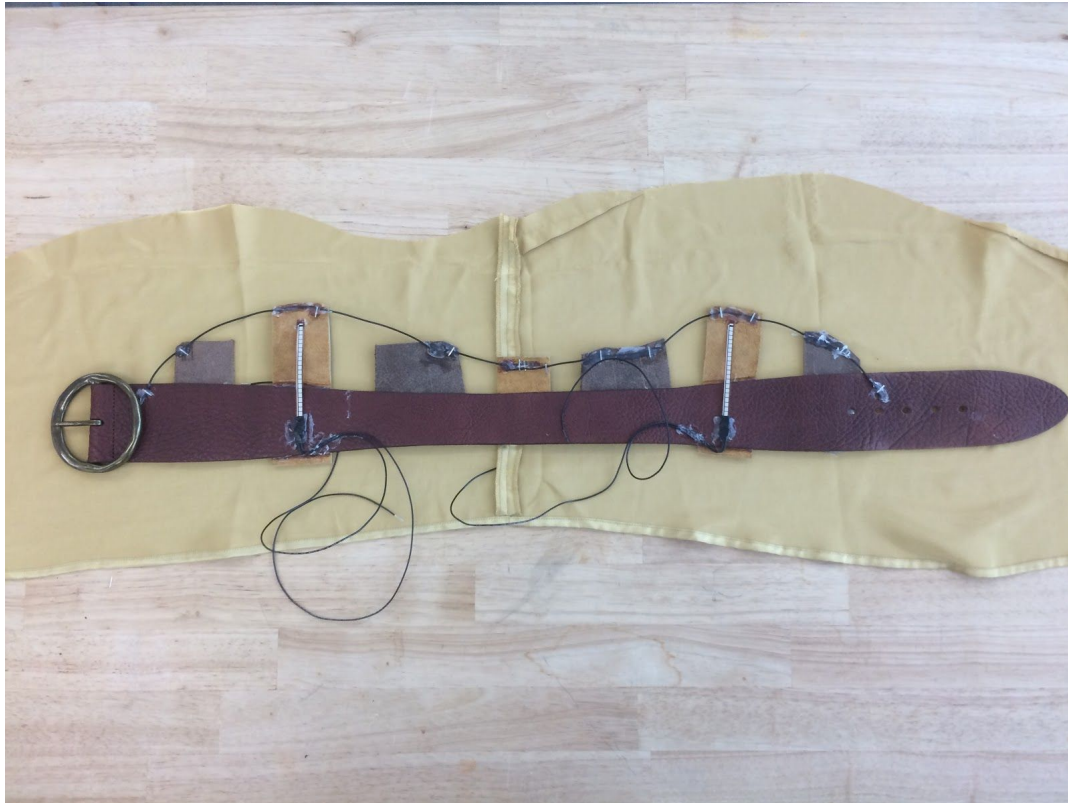


Figure 6 - Leather Shaping Pieces with Guide Wire Attached

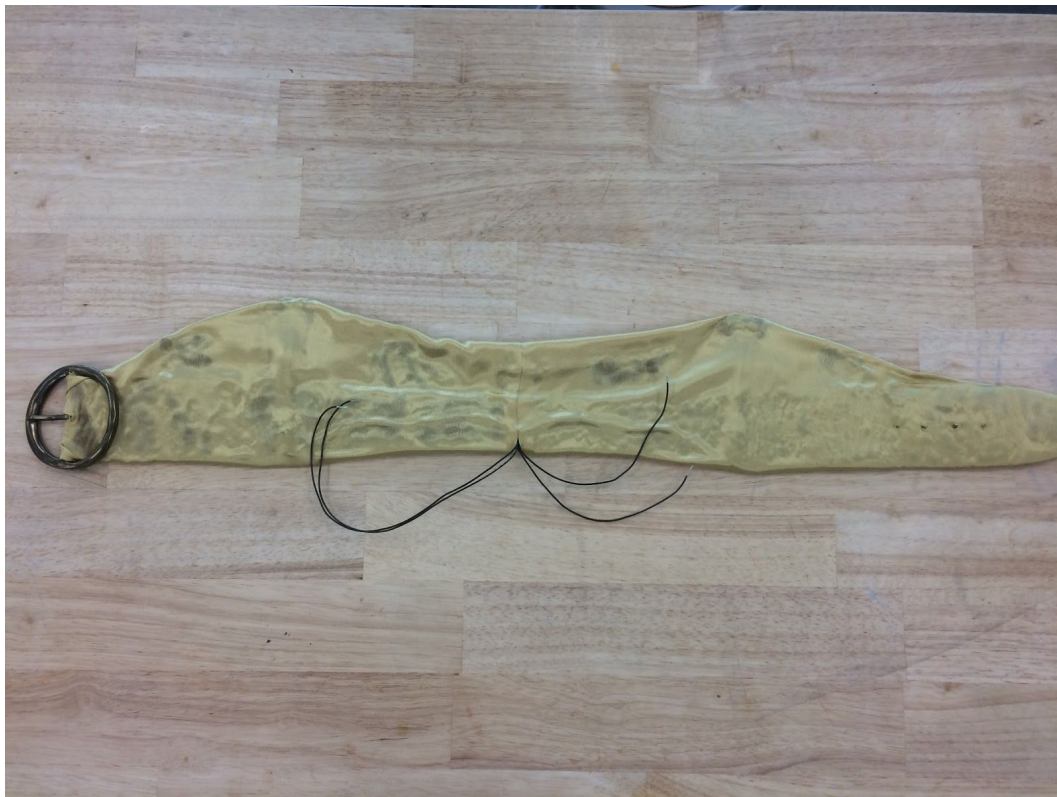


Figure 7 - Completed Belt Encased in Fabric

A major consideration taken into account in constructing the belt was the placement of the sensors on the leather straps. When wearing the completed belt, a player must be able to dodge both left and right by bending their torso in either direction. This motion is identical to what our player Little Mac does in the video game. We experimented with different placements of the flex sensors and with different amounts of existing flexion in the sensors in order to create a reliable amount of flexion each time. The leather straps were then positioned on the belt so that they would align with player's sides. Extra pieces of leather were then attached to the belt to create an aesthetic form. A piece of 3D printer filament was attached to the pieces of leather to serve as a guidewire for the fabric. The fabric was cut and hot-glued to the belt, with the connecting sensor wires protruding from the back.

## 3 Software Design

### 3.1 Working with Sensors (Jonya)

In order to code the controller, I took an incremental approach and built upon each phase. To start, I read in the data of one accelerometer. Using the LIS3DH demo code as a base, I was able to map the hardware pins, get SPI running, and read the sensor data. I also made sure to include the Adafruit LIS3DH library and the Adafruit\_Sensor library, while making sure to change some functions and header files so that it was compatible with the RedBear Duo, rather

than Arduino. I could then open up my serial terminal and read out both raw and normalized X, Y, Z-axis information. Note that I initially implemented the I2C interface but then realized that SPI can be run much faster than I2C and the Duo also conveniently accounts for two SPI interfaces which makes connecting two accelerometers much easier.

Once I got one accelerometer to work, then I also attached a second one with the SPI1 interface and added an additional library. In addition, I had to modify the code to be able to go from the sensor data to actually detecting which motion that indicated. I added a large if-else statement to create thresholds such that once a certain value is met on one or some of the accelerometer axes, then I would print out that respective gesture.

Finally, I combined the flex sensor data, which worked similarly to the two accelerometers. Each flex sensor was mapped to an analog pin. I would read in each value, check to see if it's less than a certain threshold value, and if so, I would output that the gesture is a dodge. Once all the sensors were connected, we were then able to test the overall device and of course, tweak some of the thresholds at the end to make sure that each gesture was properly detected. For the complete code, refer to our GitHub repository (link included at the end of this document).

## 3.2 Input Mapping & Emulator (Michael)

Since the game was originally for the NES, we used an emulator to run the game. There were 2 processes of input mapping: taking motion from the user and converting them into states; and taking those states and converting them into button presses.

We first needed to figure out how to turn states into button presses. Since our game was closed-source, we did not have the option to integrate the commands into the game (approach 3). Since USB HID mode is not supported on the RedBear Duo, we could not use that method either (approach 1). Therefore, we used an injection app to translate the commands from the microcontroller to the game emulator (approach 2).

We found a free RS-232 to keyboard software (keyboard wedge) that converts outputs from a serial device and inputs them into any application (in this case, the game emulator). However, the keyboard wedge only converts single digit numeric values. Therefore, the microcontroller had to output numbers and game emulator could only accept numbers as inputs. This limited our possible button presses to 10.

The next step was deciding which buttons were required to play the game. We considered the original game controls using an Nintendo Entertainment System (NES) controller. We then decided which controls we were going to implement. We decided that the start and select button were not used often during gameplay, so they did not need to be triggered with motion. They are simply pressed onto the computer's keyboard.

















Move	Instruction
<b>Dodge Left</b>	Tap  on the direction pad
<b>Dodge Right</b>	Tap  on the direction pad
<b>Quick Dodge</b>	Tap  when dodging right, or  when dodging left, to quickly return to the center.
<b>Defend</b>	Hold  on the direction pad to block punches
<b>Duck</b>	Tap  on the direction pad to duck underneath a punch
<b>Left Jab</b>	Hold  and tap  .
<b>Right Jab</b>	Hold  and tap  .
<b>Left Body Blow</b>	Tap  without touching the direction pad.
<b>Right Body Blow</b>	Tap  without touching the direction pad.
<b>Uppercut</b>	Press  to exchange one star for an uppercut. If you have no stars stored up, you cannot perform this move.
<b>Recover</b>	Rapidly tap the  and  buttons to get back up on your feet after getting knocked down.
<b>Health Boost</b>	Between rounds, press  to instruct Doc Louis to loosen you up and increase your stamina. This may only be performed once per match.

Figure 8 - Original Game Controls Using an NES Controller

Next, we had to take motion from the user and convert them into states. We used a series of gestures similar to the motions of the in-game character. By looking at the data from the accelerometers on the boxing gloves, we could infer the user's gesture.

This is the neutral stance. User's back is straight so neither flex sensor is triggered. Both gloves are held with the thumbs facing up and level with the body. No inputs are sent to the game.



Figure 9 - Neutral Stance



This is the defend stance. User's back is straight so neither flex sensor is triggered. Both gloves are held up, with the thumbs facing the face. The microcontroller sends "2" to the serial output, which is converted to the down button on the d-pad.



Figure 10 - Defend Stance

This is the duck stance. The user must bend waist until it is near horizontal and must move his/her hands with the body. The user's back is bent forward, so neither flex sensor is triggered. Both gloves are held down, with the thumbs facing the face. The microcontroller sends double "2" to the serial output, which is converted to pressing the down button on the d-pad twice.

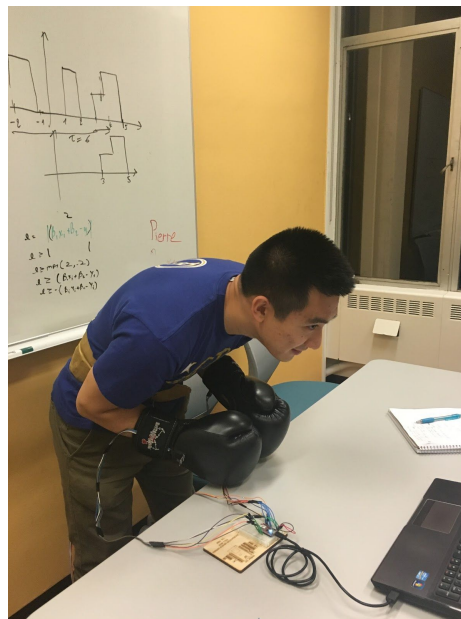


Figure 11 - Duck Stance

This is the dodge left stance. The user must bend waist to the left. Both gloves are held similar to the neutral stance. The microcontroller sends “3” to the serial output, which is converted to pressing the left button on the d-pad.



Figure 12 - Dodge Left Stance

This is the dodge right stance. Similar to the dodge left stance, except the user must bend waist to the right. The microcontroller sends “4” to the serial output, which is converted to pressing the right button on the d-pad.



Figure 13 - Dodge Right Stance

This is the punch left stance. User's back is straight so neither flex sensor is triggered. The left arm is fully extended, with the thumb facing up. The right glove is held similar to the neutral stance. The microcontroller sends "8" to the serial output, which is converted to pressing the B button.



Figure 14 - Punch Left Stance

This is the punch right stance. Similar to the punch left stance, except the user must punch with the right glove. The microcontroller sends "7" to the serial output, which is converted to pressing the A button.



Figure 15 - Punch Right Stance

Here is a summary of the sensor trigger values, the microcontroller output, the emulator input, and the game action. For the sensor values, the first letter corresponds to the X-axis of the accelerometer, the Z-axis of the accelerometer, or the flex sensor. The second letter corresponds to the left or right sensor. The accelerometer units are  $\text{m/s}^2$  and the flex sensor units are voltage from the ADC pins on the microcontroller.

Sensor Values	Microcontroller Output	Emulator Input	Game Action
ZL > 8, ZR > 8	2	Down on D-Pad	Defend
ZL < -2, ZR < -2	2, 2	2x Down on D-Pad	Duck
FL < 3000	3	Left on D-Pad	Dodge Left
FR < 2900	4	Right on D-Pad	Dodge Right
ZR > 6, XR < -6	7	A	Punch Right
ZL > 6, XL > 6	8	B	Punch Left

Figure 16 - Input Mapping Summary

## 4 Conclusion

Overall, we were satisfied with our end product. The gloves and belt look good to wear, and the controller responds to the user's inputs well enough to play the game. Implementing the flex sensors were the easiest part of the project. Since they are part of a simple voltage divider circuit, we did not run into any complications.

We faced plenty of obstacles implementing the accelerometers. They worked perfectly fine when we were testing them on the breadboard. However, they were very finicky when we connected them to the long ribbon wire. Sometimes the microcontroller would not get a connection from the accelerometers at all and nothing would print into the terminal. Sometimes the accelerometers worked when we first uploaded the code, but would show all 0s after a couple of minutes.

After much effort debugging every conceivable failure mode, we found out that the SPI protocol has a max wire length. Due to the master-slave relation of SPI protocol, the data must transfer from the microcontroller to the accelerometers and back within the clock frequency. With a long wire, the data is not transferred fast enough so a connection cannot be made. A possible solution is lowering the SPI clock frequency to prevent the propagation delay, but this might be

too slow for our controller needs. Therefore, we decided to use shorter wires but made sure that they were long enough for the user to have full range of motion.

Another obstacle was that the keyboard wedge could not output two buttons at once. This prevented us from implementing the high punches because they require the user to press up and A/B at the same time. The keyboard wedge was a free software that had a 20ms delay for free users. A delay that was smaller than the game's framerate (60 frames per second equates to 16.66 ms) could have worked. We attempted to run the game at a slower rate using the emulator, but this negatively affected game play. We ultimately decided that only implementing the body punch was sufficient to play the game.

There are plenty of improvements that we can make for future iterations. As for hardware, the belt was very small and can only fit very slim people, so the next iteration will utilize an elastic belt that is one size fits all. In addition, the wires connecting the gloves to the breadboard are also relatively short, so we would like to make them longer or possibly made the gloves wireless.

As for software, we can definitely improve the gesture recognition. With more time and user testing, we can calibrate better voltage values. In addition, we want to implement the high punches and start and select buttons. Using another keyboard wedge or implementing a macro can simulate simultaneous button presses. We could implement unique gestures for the start and select buttons as well.

## 5 Links

GitHub: <https://github.com/idd-fall16/PunchOut>

Demonstration Video: <https://youtu.be/9-gMRVcJlko>

Nintendo Power Glove being used to play Mike Tyson's Punch-Out!!:

<https://youtu.be/93iDhnBcMGo?t=21>

Original Game Controls:

[http://strategywiki.org/wiki/Mike\\_Tyson%27s\\_Punch-Out!!/Getting\\_Started](http://strategywiki.org/wiki/Mike_Tyson%27s_Punch-Out!!/Getting_Started)

Keyboard Wedge: <https://www.232key.com/>

NES Emulator: <http://nestopia.sourceforge.net/>

Discussion about Max Wire Length for SPI Connection:

<http://www.avrfreaks.net/forum/max-spi-distance>