Week 6:

Date: 09/30/2022

Hours: 14

Description of design efforts:

Hardware:

I resolved the piezoelectric sensor this week by swapping both sensors on the table for 2 fresh ones. With our current prototype setup, I found that we run into an issue where the leads can become unsoldered, if not taken care of properly. Unfortunately, the small leads already soldered to the sensors are very fragile, and little solder holds them to the sensors. Yet, too much heat and soldering done by us is likely to cause issues on the sensor, so we must leave those connections as-is. So, we must make sure to cover those well (with masking tape, for now), and not allow the wires to bend in directions that apply forces that may pull on the solder.

I tested our current setup with the new sensors and found a strong correlation between bounces on either side of the table, and which side the code predicted the bounce to be on. More detail about this algorithm is explained in “Microcontroller Programming”. No bounces on the red side of the table resulted in a “blue bounce”, and I found that less than 1% of bounces on the blue side resulted in a “red bounce”. I believed this discrepancy was due to slight variations in the strength of the sensors. To resolve that problem, I added a second sensor to each side of the table to boost each side’s ability to detect a bounce. Now, we are more likely to detect stronger signals on the side the bounce occurred on. A messy, but insightful, image of the circuitry is shown below in Figure 1.

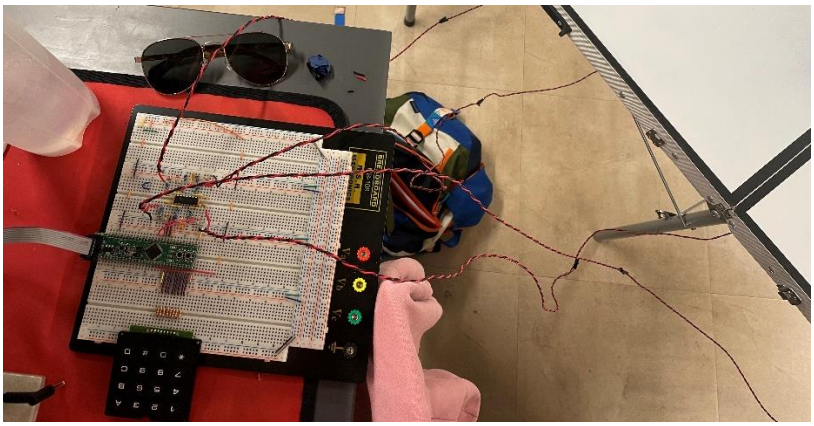

Figure 1. Microcontroller Circuitry with Four Piezoelectrics

With 2 microphones on each side, I have yet to see any wrongly detected bounces. Large signals, such as slamming the paddle on the table, will trigger both sides, however. Of course, this could be an accidental reality, thus, we should be able to handle this situation. If a packet is sent to the laptop with both bounces detected, likely, it should be dropped. If, however, the score is wrongly calculated, the user will be able to modify the score manually.

Microcontroller Programming:

The most extensive work I've done this week incorporates creating the current bounce detection logic. Currently, the system works as follows:

1. Choose a threshold value, either by a calibration sequence (WIP) or by a hard-coded entry. While debugging, I am using a hard-coded threshold, set to 400 (a 12-bit value, in accordance with the ADC). This value was picked by running my calibration sequence a few times and averaging the values.
2. Poll each ADC channel always. Right now, this happens at a rate of 20 kHz. Because this accounts for 4 channels, we read each channel at approximately 5 kHz.
3. If any sensor detects a signal larger than the threshold, begin storing each ADC transfer for the next 40 rounds, for a total of 160 recorded uint16_t's.
4. Sum all the signals from the red side.
5. Sum all the signals from the blue side.
6. If the red sum is larger than the blue sum, record a red bounce. Otherwise, record a blue bounce.

Temporarily, some basic game logic is added, for demonstration purposes:

7. If we record a blue bounce, and the most recent bounce was also blue, add 1 point to red's score, and reset the previous bounce (vice versa for blue).

Note: additions must be made to this code to be well-designed but will not be continued on the microcontroller. All the game logic will be on the laptop. We need, for example, a "timeout" sequence, where if we bounced on the red side and don't receive another bounce within a given time frame, we infer that the red player has missed the ball, and it is now on the floor. This will add a point to the blue payer's side. This is code that will be added in the coming weeks.

I also spent some time removing hard-coded values for enumerations and #define statements. These values include, but are not limited to, the number of ADC channels to support, how many samples of each channel to collect, and each of the bits to set in the UART packet. These changes not only make the code more readable, but help when modifying values later, where changes must only be made in 1 place.

Next Week:

Next week, I intend to pursue completion of the following tasks:

1. Add error reporting logic in packet configuration logic. That is, if we're filling out information in a non-empty packet, we have produced an error.
2. Add WDG timer to protect the code against hard faults.

With these tasks complete, I will begin to assist with other work, relating to either game logic or development of ball tracking algorithms. Of course, much more work can be done now to work on subsystems such as microphone triangulation, but our focus should be on tasks that are timelier. While the ADC system is fresher in my mind right now, it is more helpful to the team that I assist them in software.