Week 9:

Date: 10/20/2022

Hours: 14

Description of design efforts:

Game Logic:

This week, game logic was the focus of my work. That is, finalizing the basic scoring logic for our base ping pong game. Jack and I planned out an OOP system (using C++ code) to handle playing different minigames, and the relevant objects and inheritance paths; these additions will be made later, as for now, we do not yet have other minigames prepared to be played.

The laptop correctly receives all UART packets from the microcontroller and parses them into the relevant bounce and key press data. It processes this data to handle any necessary game changes. Right now, nothing is yet projected on the table, nor is any ball tracking system yet active. The game loop currently follows the below loop:

```
1. (Re)start the poll timer (will be used later) [To request data every 30 ms].
2. Send 0x1 to the microcontroller via UART, indicating a data request.
3. Wait until we receive a data packet back from the microcontroller.
4. Parse the data packet into its button press data and the bounce data.
5. Handle scoring logic:
    a. If current bounce is not NO_BOUNCE:
        i. If previous bounce is NO_BOUNCE, serve logic:
            1. Store current bounce as previous bounce.
            2. Start the countdown timer (will be used later) [To count a score
               if no ball bounce occurs after 3000 ms].
        ii. If previous bounce is current bounce, score logic:
            1. If current bounce is red bounce:
                a. Award blue 1 point, if possible.
            2. Else:
                a. Award red 1 point, if possible.
            3. Set previous bounce as NO_BOUNCE.
            4. Stop the countdown timer.
        iii. Else:
            1. Set previous bounce as current bounce.
            2. Restart the countdown timer.
    b. If countdown timer is on:
        i. If countdown timer > 3000 ms:
            1. If previous bounce is red bounce:
                a. Award blue 1 point, if possible.
            2. Else if previous bounce is blue bounce:
                a. Award red 1 point, if possible.
6. Handle button press logic:
    a. If no button press, return.
    b. If 1, increment red score by 1, if possible.
    c. If 4, decrement red score by 1, if possible.
    d. If A, increment blue score by 1, if possible.
    e. If B, decrement blue score by 1, if possible.
    f. If 2, decrement the maximum score by 1, if possible.
    g. If 3, increment the maximum score by 1, if possible.
    h. If D, and the last press was D, shutdown the game.
    i. Store the current button as previous button.
7. Check to see if either side has reached the maximum score.
    a. If so, print a message and shutdown the game.
8. Ensure the poll timer has been waiting at least 30 ms, then return to 1.
```

Microphone Array Trilateration:

I began prototyping a trilateration system for our contact microphones. First, I began by ensuring that this may be feasible. To do so, I followed Dr. Walter's advice and focused on the *phase* and not the *strength* of the signals. Of course, we must detect location of different strengths of bounces, thus, that is not a good point to test. The time at which each microphone detects each bounce at the same location, however, will be the same, as the speed of sound will be (approximately, probably; I'm not a materials engineer) static through our table, regardless of the strength of the signal (i.e., loudness of the contact).

The output of the piezoelectric sensors will be a fluctuating signal, oscillating (approximately) sinusoidally, centered on a 0 V bias. Through our filter and protection circuit, we roughly ignore the negatively biased signal and only read the positive signal. This signal will be read at each microphone, but will begin at different times, due to the speed of sound. The closest microphone, of course, will see the signal first. Then, through some math, we can determine the position of the ball based on the time delays.

Figure 1 depicts a prototypal example of the software to handle this. Here, we have 4 microphones arranged in a square, with known positions. Each square in the grid represents 1 unit, which for simplicity, can be defined as 1 ms of time, as in it would take 10 ms for a signal at microphone 1 (10, 10) to reach microphone 2 (0, 10), using the typical axis for computer graphics, where (0, 0) is at the top left corner.

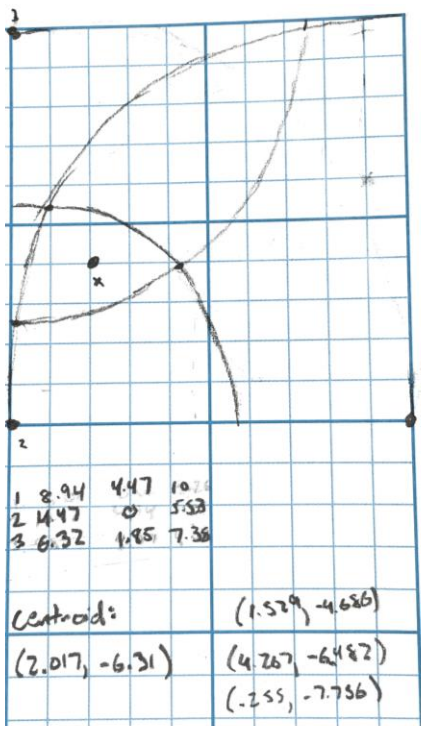


Figure 1. Depiction of Microphone Setup and Bounce Detection

Say we record a bounce at (2, 6). Using the Pythagorean Theorem, we find that microphone 1 (M1) receives the signal at 8.94 ms, microphone 2 (M2) at 4.47 ms, and microphone 3 (M3) at 6.32 ms. But the first microphone to receive the signal, M2, will interpret it's time as 0 ms; thus, M1 and M3 store times of 4.47 ms and 1.95 ms, respectively, if we write their values as the delay from 0 ms.

We can normalize these values to make the calculations easier. To do so, we can normalize the longest delay to 10 ms. The logic behind this is simple: first, 10 is an easy number to normalize to. Second, we know that we should never see a delay of greater than 10 seconds. This is because we will have a microphone at each corner of the square, and the 3 closest ones will be chosen. The most extreme case would be if the bounce occurred directly on a sensor, where the delay would be (correctly) 0 ms. Then, the 2 closer sensors will record exactly 10 ms. If the sensor would have recorded more than 10 ms, a different sensor would have been picked. This logic, of course, relies on the sensors being distributed in a square pattern, which will be more difficult to coordinate on the rectangular ping pong table (but, we plan to have 4 sensors on each side of the table, where each half is approximately square).

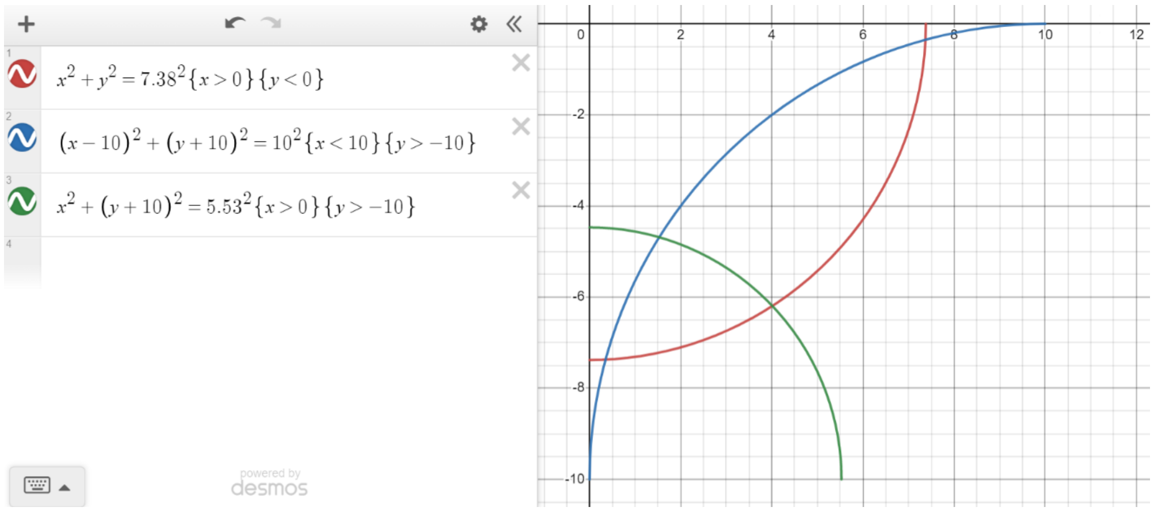
To normalize the signals, we set the largest delay to 10 ms, and the difference between the largest delay and 10 ms will be added to the other 2 signals. To recap on our values, refer to Table 1 below:

	Target (True Delay), TD	Detected Delay, DD	Normalized Delay, ND
M1	8.94 ms	4.47 ms	10 ms
M2	4.47 ms	0 ms	5.53 ms
M3	6.32 ms	1.85 ms	7.38 ms

Table 1. Microphone Sensor Delays

If we were to "draw" the circle with center Mx and radius NDx, we will almost always create a set of graphs with 3 intersection points, 1 between each pair of circles. The corner cases for this algorithm occur when a bounce is perfectly on a corner, where the point will be correctly determined (with a circle of radius 0, and 2 of radius 10), or perfectly in the center (where 3 of the 4 microphones will be chosen at "random", as each theoretically has exactly 0 detected delay), where we also have 1 intersection point with all 3 circles. Note that these cases regard only the intersections of the circles within the 10-by-10 grid that we care about.

With the 2 "base cases", where no additional logic is needed, we have every other case, where there are 3 intersections. If we use each intersection point as points of a triangle, we can estimate the position of the bounce to be at the centroid of the triangle. The equations and graphs are shown in Figure 2, below.



$x^2 + y^2 = 7.38^2 \{x > 0\} \{y < 0\}$

$(x - 10)^2 + (y + 10)^2 = 10^2 \{x < 10\} \{y > -10\}$

$x^2 + (y + 10)^2 = 5.53^2 \{x > 0\} \{y > -10\}$

Figure 2. Plotted Bounce Detection Points

Here, we find the intersection points to be (1.529, -4.686), (4.267, -6.482), and (0.255, -7.756). To find the centroid of these three points, we simply average the x's and average the y's, and get (2.01, -6.31). Remember, we flipped the Y value in our handwritten interpretation, making our centroid point really (2.01, 6.31), which is reasonably close to our bounce at (2, 6).

I know this part of my report is rather informal, but if you have any comments, concerns, or notice any flaws in any of the thinking of this logic, please let me know. I do intend to meet with Dr. Walter soon about this algorithm and what he may suggest as well.

Next Week:

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

1. Debug the game logic to properly score a game of ping pong, beyond any table-moving or intricate rules, such as a let.
2. Continue work on trilateration algorithm.
3. Work as a “freelancer”, filling in man hours where needed to assist my teammates.