Closed Loop Flappy Bird System: Final Report
Matthew Arboleda, Seolbin Hong, Julia Chun

Github Link to the code:

https://github.com/Jchun2/Flappybird

1. Abstract

The aim of this project was to extend the learning goals of the Spring '25 Robotics Course to a final project of the students' choosing. Our group chose to expand upon the mechanical properties of the arduino development board, LCD screen, and a time-of-flight (ToF) distance sensor to simulate a closed-looped robotics interaction with real-time sensor feedback to control a game's logic and respond to a dynamic user environment. We used the arduino kit and LCD instead of a computer screen to distinguish between a robotic system and a general-purpose hardware/software, emphasizing continuous sensing, actuation, and feedback. Key algorithms within this project are feedback control for vertical movement of the character and drawing of dynamic scenes with obstacles and user movement interacting. This game has been implemented and tested within the constraints of an embedded platform concerning computational resources and latency challenges. It is also required for a responsive, intuitive control system based on real-world sensor input. Through this project, we explored the complexities of embedded systems, unpredictability of physical inputs, and future extensions of advanced robotics and computer science.

2. Objectives

The procedural goal of this project was to create a game using an arduino kit, lcd and a time of flight sensor. We hoped to extend our knowledge from our previous projects with arduino, and experiment more heavily with the components of a developmental board to perform certain tasks. We aimed to transform the readings from the sensor into meaningful data so that we could manipulate the information and allow the user to interact with the system and receive feedback from the system. For our purposes, based on the success of the detected movement of the user's hand and the game's mechanisms, the difficulty of the game would change with the speed and pipe gaps.

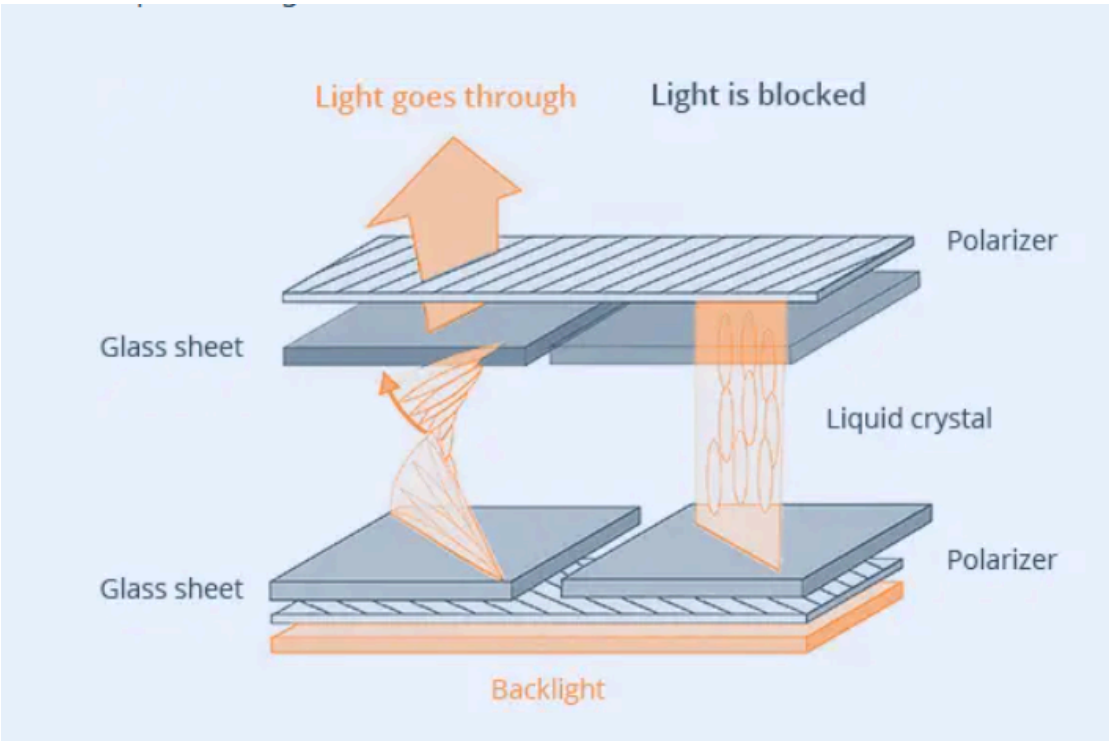
3. Prior Work/Background on LCD

3.1 Liquid Crystal Display (LCD) Exploration and Background

For this project, we wanted to explore and use the Liquid Crystal Display (given from the Arduino kit) as we have not used this device in the previous labs. This device is a flat panel display which uses liquid crystals to operate. For our project, this would be used as the main display for our Flappy Bird game.

The LCD is made of millions of pixels and works by controlling the amount of light that passes through liquid crystal molecules sandwiched between two polarizing filters. When an electric current is applied, the orientation of the crystals changes, altering how much light can pass through. This allows each pixel to appear dark or light depending on the signal it receives, forming characters or graphics on the screen. In our project, the LCD's characters are

composed of 5x8 dot matrices, and we used these to display both static and dynamic elements of the game.



The LCD has 2 rows and 16 columns and is capable of displaying alphanumeric characters and custom glyphs made from a 5x8 pixel matrix. It uses multiple wires to send data from the microcontroller to the display simultaneously (this process is also called a parallel interface contrast to the serial interface which sends data one bit at a time). The LCD receives instructions and data via a 4-bit parallel interface from the Arduino. Character codes are sent to the display RAM (DDRAM), and for more advanced rendering, such as the bird and pipe graphics, we define custom characters using the character generator RAM (CGRAM). Up to 8 custom characters can be defined at once.

Potentiometer use:

The contrast of the display is set by applying a reference voltage to the VO pin, controlled through the potentiometer wired as a voltage divider. This allows tuning of the darkness/brightness of the characters to suit various ambient lighting conditions.

4. Process

4.1 Creating a Closed Human in the Loop Robotic System

Closed Loop Flappy Bird System: Final Report
Matthew Arboleda, Seolbin Hong, Julia Chun

We decided to implement a system reflecting a closed-loop robotic system in the form of a Flappy Bird-style game using the Arduino Uno, an LCD screen, a VL53L0X distance sensor, a servo motor, and LED indicators.

This system operates as a closed-loop control system where the user's hand movement changes the distance read by the VL53L0X sensor. That reading is immediately mapped to a vertical position for the bird displayed on the LCD. Based on the bird's position relative to the obstacles, the game updates the servo position and flashes LEDs to indicate success or failure. The user sees these changes and responds by adjusting their hand position again. The continuous feedback cycle between input, output, and user response forms a real-time closed loop, but specifically a d human-in--loop feedback control system as this system relies on human response as the control mechanism. The user perceives the system's feedback on the LCD display, servo motion, and LED indicators to adjust their hand.

4.2 System Overview

The system uses four main hardware components:

LCD (16x2): Used to visually display the bird and obstacles in a 16-pixel-high space using custom character bitmaps. The LCD allows for abstract spatial feedback within a minimal display.

- Potentiometer: The potentiometer connected to the LCD adjusts the display contrast. that it acts as a voltage divider: rotating the knob changes the resistance ratio, thereby adjusting the voltage at the wiper (VO pin), which controls how dark the characters appear.

VL53L0X Distance Sensor: Senses the vertical hand position of the user and serves as real-time input.

Servo Motor: Rotates in steps to represent the user's progress, providing continuous physical feedback and indicating game progression toward a win condition.

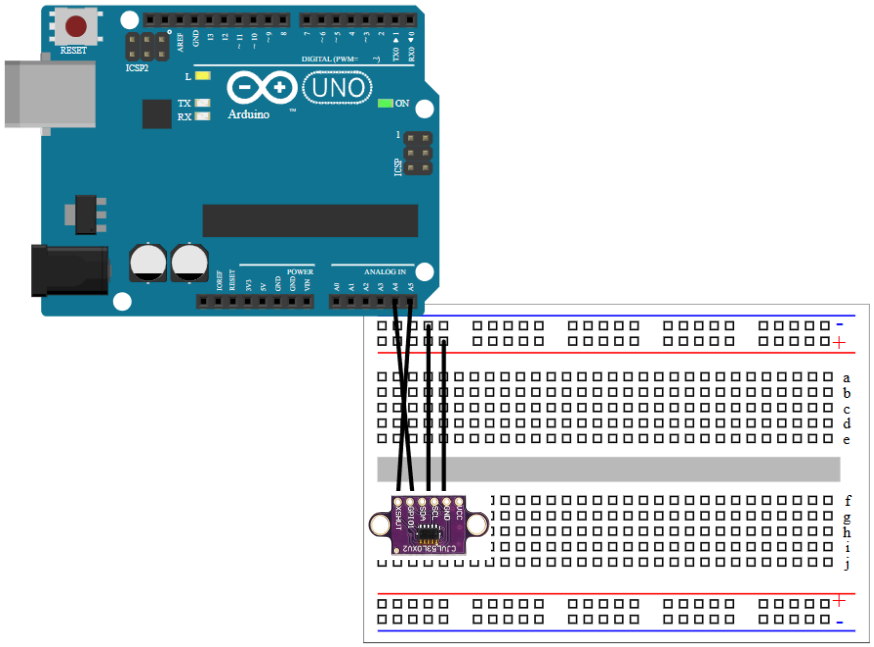
LEDs (Red and Green): Indicate crash or success through brief flashes, providing simple, intuitive binary feedback.

4.2.1 Wiring and Circuit Diagrams

Note: that the colors in the parentheses represent the color of wire on the circuit diagram schematic. This does not represent the color of wires in the actual (real life) pictures.

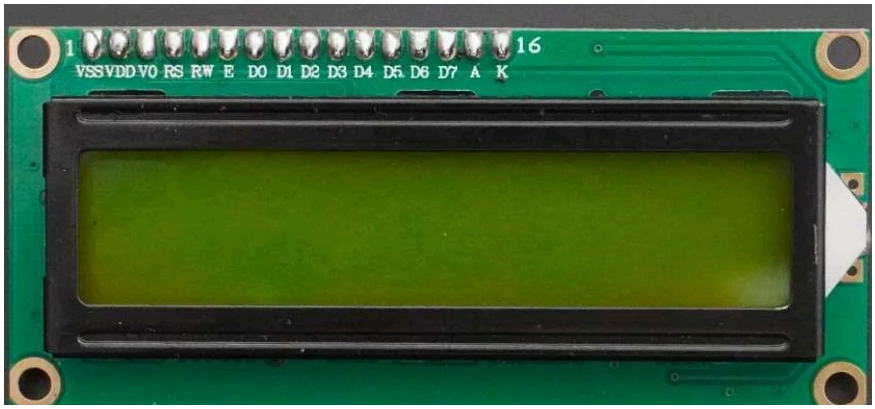
4.2.1a Input Connections
VL53L0X Distance Sensor (black)
We connected VCC to the positive railing and GND to the negative railing of the sensor. SCL is connected to analog pin 4 and SDA is connected to analog pin 5.

Closed Loop Flappy Bird System: Final Report
Matthew Arboleda, Seolbin Hong, Julia Chun



4.1.2b Output Connections

LCD Connections:



Closed Loop Flappy Bird System: Final Report
Matthew Arboleda, Seolbin Hong, Julia Chun

LCD connections:

RS (Register Select) (yellow)– Pin 12: Selects whether data is interpreted as a command or character data.

E (Enable)(brown) – Pin 11: Used to tell the LCD when to read the data lines.

DB4–DB7 (Data lines) (green)– Pins 5, 4, 3, 2: Used to send 4-bit binary data to the LCD.

VSS (grey/VND in schematic)– GND: Connects to ground.

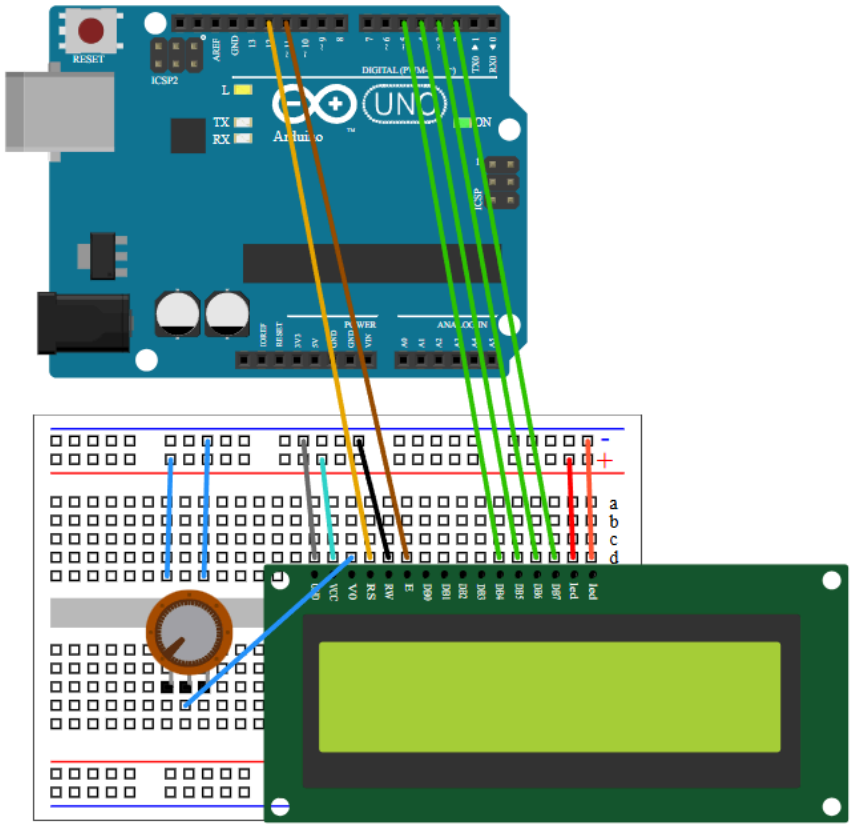
VDD (turquoise/ VCC in schematic)– 5V: Powers the LCD.

VO (Contrast) (blue)– Middle terminal of potentiometer: Adjusts character contrast via analog voltage.

RW (Read/Write) (black)– GND: Tied to ground to set the LCD in write-only mode.

LED+ (Anode) (red)– 5V through a resistor: Powers the backlight.

LED- (Cathode) (orange) – GND: Backlight ground.



4.1.2b Servo Motor Connections (purple)

We connected the servo motor in DB0, DB1, and DB2 in the LCD. DB0 is connected to ground while DB2 is connected to the positive railing. DB1 is connected to Digital Pin 10.

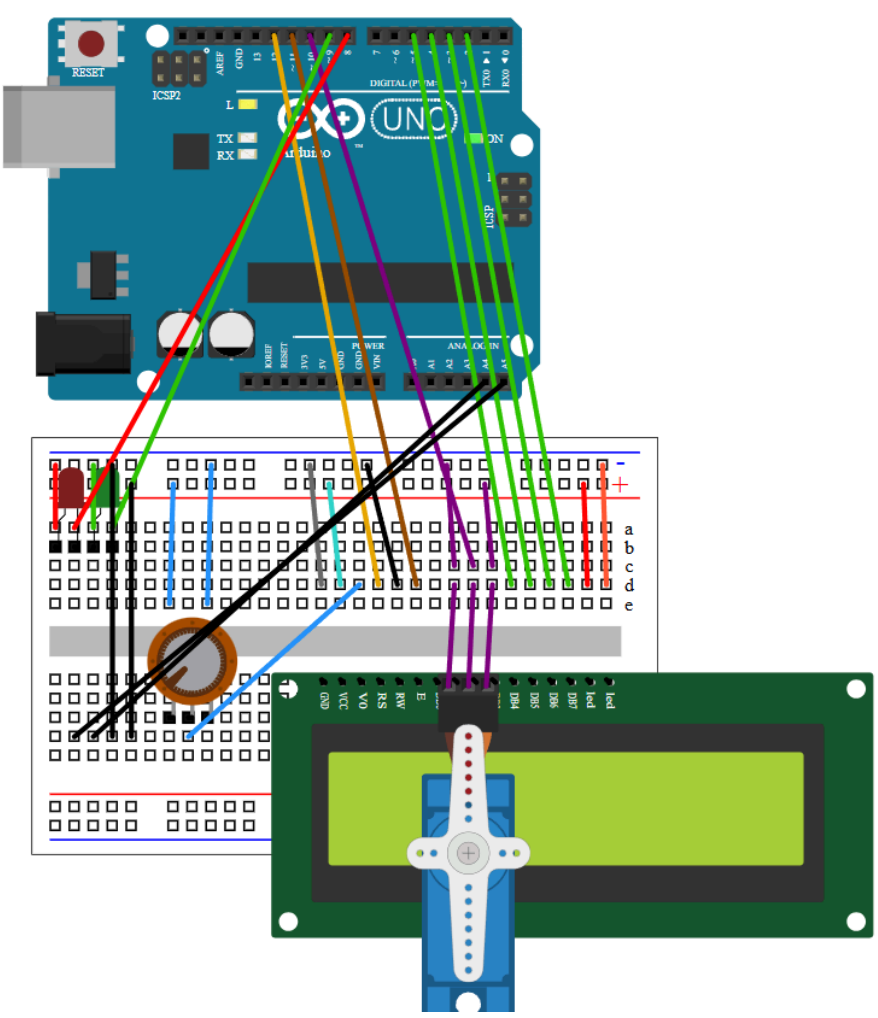
4.1.2c LED connections (red wires for red LED and green wires for green LED)

We connected the red and green diodes (side by side) in the top left corner of the breadboard. We made sure that the shorter leg(cathode) of the diode

connected to ground and the longer leg(anode) connected to the digital pins

(8 for the red LED and 9 for the green LED)

Full Circuit Wiring Schematic (created using “Arduino on Cloud” software)



Closed Loop Flappy Bird System: Final Report
Matthew Arboleda, Seolbin Hong, Julia Chun

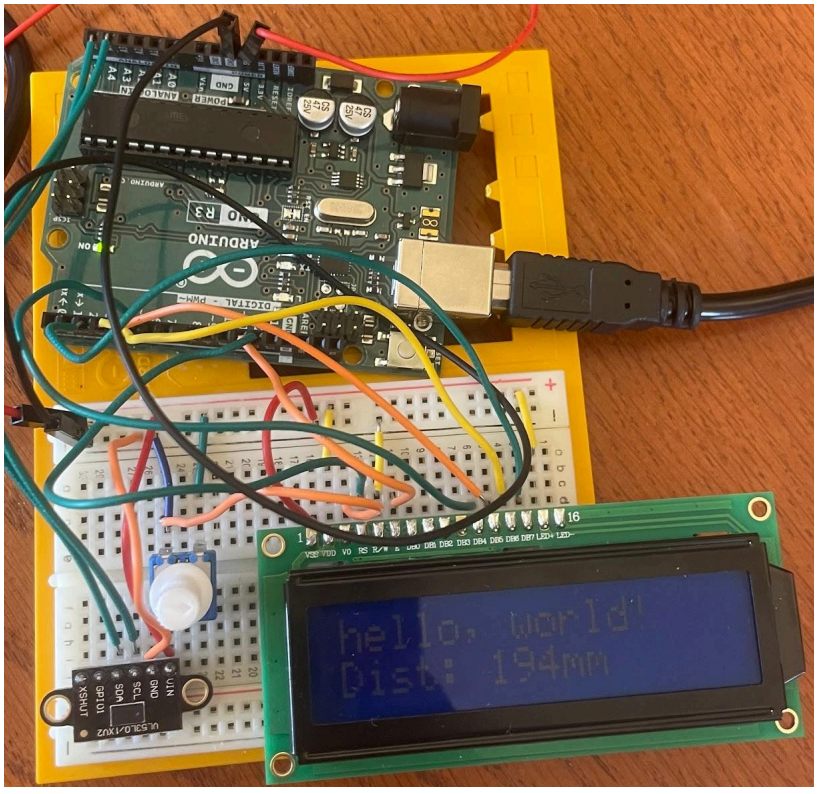
4.3 Testing the Circuits and Building our system

4.3.1 Testing the LCD

To test just the LCD display, we ran the tester code “HelloWord.ino" from the Arduino library (should be included in Github link). This program prints “hello, world!” and then also prints the time elapsed on the LCD. We decided to test this first to make sure our LCD was working properly.

(we lost out picture of our LCD display for this step so found a picture of we saw online)

4.3.2 Adding out input device, the Time of Flight Sensor
We then connected our time of flight sensor to the system and outputted out Time of Flight sensor value on the LCD screen.



4.3.3 Adding LEDs and our Servo Motor
Code:

Since our game is using the LCD screen, we were limited in space had we decided to use full characters (one character per character panel). We wanted our game to be dynamic and have multiple positions in its vertical movement, but because it was a 2x16 character panel, with up to 8 unique characters stored at a time, we thought that we could, at first, only move our character two positions, the upper row and the bottom row. However, after research, we learned that each character was 5 x 8 bits, which allowed us to have 16 positions for the character (as we made the character 1 pixel).

```
void makeBirdChar(int pixelY, bool mergeWithPipe, byte pipeData[8]) {
  for (int i = 0; i < 8; i++) {
    byte base = mergeWithPipe ? pipeData[i] : B00000;

    // If this row matches the bird's vertical position, add the bird pixel
    if (i == pixelY) {
      base |= B00100;
    }

    birdChar[i] = base;
  }
}
```

We created simple bitmaps for the character and pipes to help us work with the LCD. Though we now had more positions with the character, we still ran into issues with creating and storing unique characters properly. When the character and pipe overlapped, the pipe would disappear momentarily and the bird character would be the only thing showing up in that character space until the pipe moved left and appeared again as they were not overlapping anymore. We had some difficulty with merging the character and the pipe bits together as one character, but we were able to do it by creating a unique character with the combined bird pixel and pipe pixels.

To make the scene more dynamic, we created a helper function to create different heights for the pipes. This function is able to control the display through a byte array. Once the values in the pipe byte array are changed they are then displayed as a “character” on the LCD display which are just filled in rectangles. We also attempted to make the pipes more dynamic by having them flow into the opposite row (so that the bottom row could occasionally be taller than 8 bits and be in the bottom and top character/vice versa). We succeeded in creating the scenes, but it created complications with the collision/score system and would not work well together as the frames were being displayed quickly and had to merge with the character. Due to time constraints and the late arrival of this idea, we did not fulfill it by this deadline but hope to do so by the presentation time. We created the bird/character like how we created the pipes by using a byte array. This is all done in order to allow us more freedom with the limited space we had on the LCD screen.

In terms of the game itself, the pipes start being displayed on the rightmost column of cells. The code loops itself to where each iteration of the loop the location of the pipes gets

shifted one column to the left until it reaches the leftmost column. Once this has happened the next set of pipes are rendered on the rightmost column with new random values for their heights. The character's position on the LCD doesn't change on the x-axis as the pipes come towards it, but the value on the y-axis is controlled by the player/user. The player will control this through the VL53L0X (time of flight) sensor. We use the input of the sensor in order to have the position of the bird move up or down according to the distance of the player's hand in relation to the sensor. At first, we had the range of the sensor at some arbitrary range like (0-200), but then we were able to configure it so that the initial position/distance of the user's hand mapped onto the middle of the game and create a different range (same difference in distance, but different distances away from the sensor). We coded a beginning where the distance of the user's hand will be averaged over three measurements over three seconds to code the middle of the range, so the interaction and sensor values could be more dynamic. We used the `map()` function to translate the sensor's distance readings into a pixel value (`birdPixel`) for custom rendering. This setup let us maintain smooth vertical animation and helped calibrate the experience to each individual user's hand position. The map()function linearly scales a value from one range to another and performs the formula scaledValue = (value - fromLow) * (toHigh - toLow) / (fromHigh - fromLow) + toLow. In our code below, delta is the hand's offset fro mthe center in mm, the input range is from +60 to -60 (hand moves +- 60 mm from neutral, and the output range is 0 to 15 representing the bird;s pixel position on the LCD.

```
27
28      waitForHandToStart();
29      threshold = countdownAndGetDistance();  // Continue setup
30
31      lcd.clear(); // Clear the LCD after countdown
32    }
33
    // Apply dead zone
    if (abs(delta) < deadZone) {
      delta = 0;
    }

    // Map to screen pixel range
    birdPixel = map(delta, movementRange / 2, -movementRange / 2, 0, 15);

    lastDist = dist;
  }
```

We also made it so that over the course of the game, based on the user's success, the "speed" at which the pipes move toward the player's character changes, as well as the gap between the pipes. We made this linked towards the score of the match which is based on how

many pipes the player has managed to avoid. The score subtracts from itself each time the player hits a pipe/obstacle. We were able to change the speed of the game by having a variable (frameDelay) which is used at the start of the loop to dictate how fast each iteration happens This means that if a player is struggling and hitting the pipes/obstacles, the speed will slow down and the pipes will be wider if their score drops. After the player reaches the target score, the game is finished and is detected by a conditional in the loop which will display a win message informing the player the game is over. They are also notified that they can play the game again if they hold their hand over the sensor (within a certain distance) for three seconds. The game restarts, but the initial distance threshold set by the user remains the same.

The full code is in our Github link named “Flappybird.ino” with commented descriptions

6. Quantitative Observations and Parameterized Testing

We conducted several tests to evaluate performance and repeatability:

Quantitative Observations:.

- Servo motor reached 150° reliably after 4–6 successful pipe passes.
- A consistent delay of ~400 ms per frame was reduced to ~200 ms after a score of 5, and ~100 ms by score 10.
- Pipe gap decreased linearly with score from 6 to 2 rows.

Parameterized Tests:

- Sensor-to-screen mapping: Tested hand placement at known distances (e.g., ±60 mm, ±30 mm from threshold) and verified correct pixel output using Serial print and visual LCD confirmation.
- Servo angle vs. score: Repeated passes confirmed +15° per success and –5° per failure, with a hard limit at 0° and 180°.
- Reset logic: Player could consistently restart the game by holding their hand within 500 mm for 3 seconds, as verified over 5 repeated trials.
- Pipe rendering reliability: Byte array–driven pipes displayed correctly at all randomized gap heights without graphical glitches.

These tests allowed us to confirm the reliability of our closed-loop system, validate our calibration logic, and identify areas (like display lag at high frame rates) for future refinement.

Closed Loop Flappy Bird System: Final Report
Matthew Arboleda, Seolbin Hong, Julia Chun

Extra test:

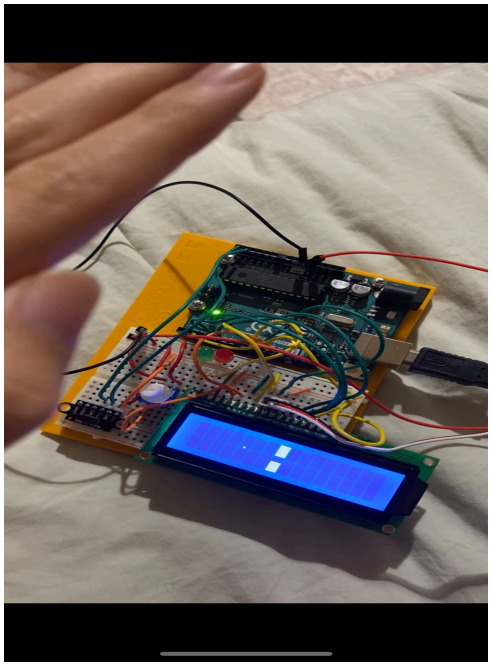
Servo Angle Progression Each success increased servo angle by +15°, and each failure decreased it by –5°.

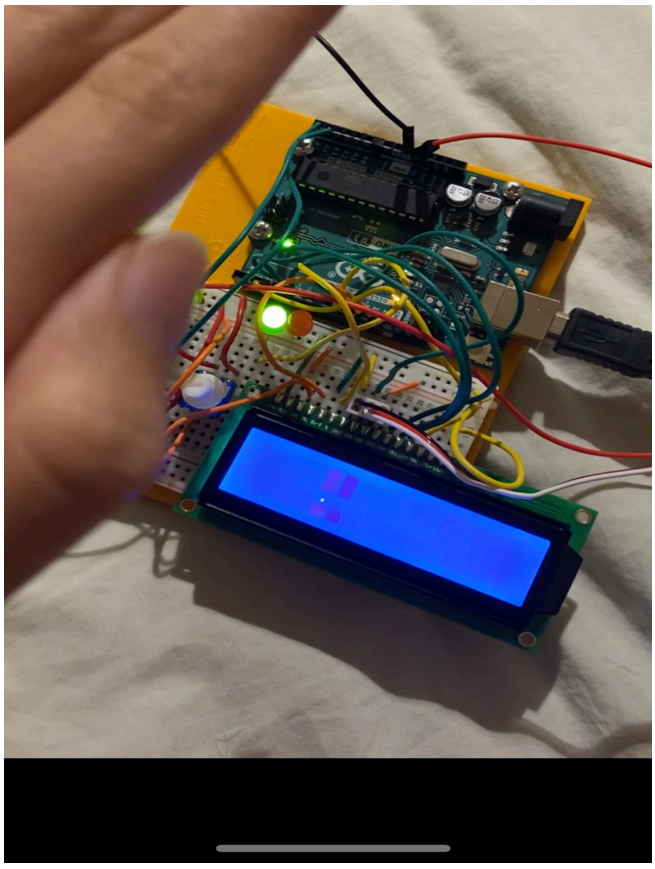
Trial	Successes	Failures	Final Servo Angle
1	5	2	145°
2	6	1	155° (Win)
3	4	3	135°

Players generally won after 5–7 successful passes.

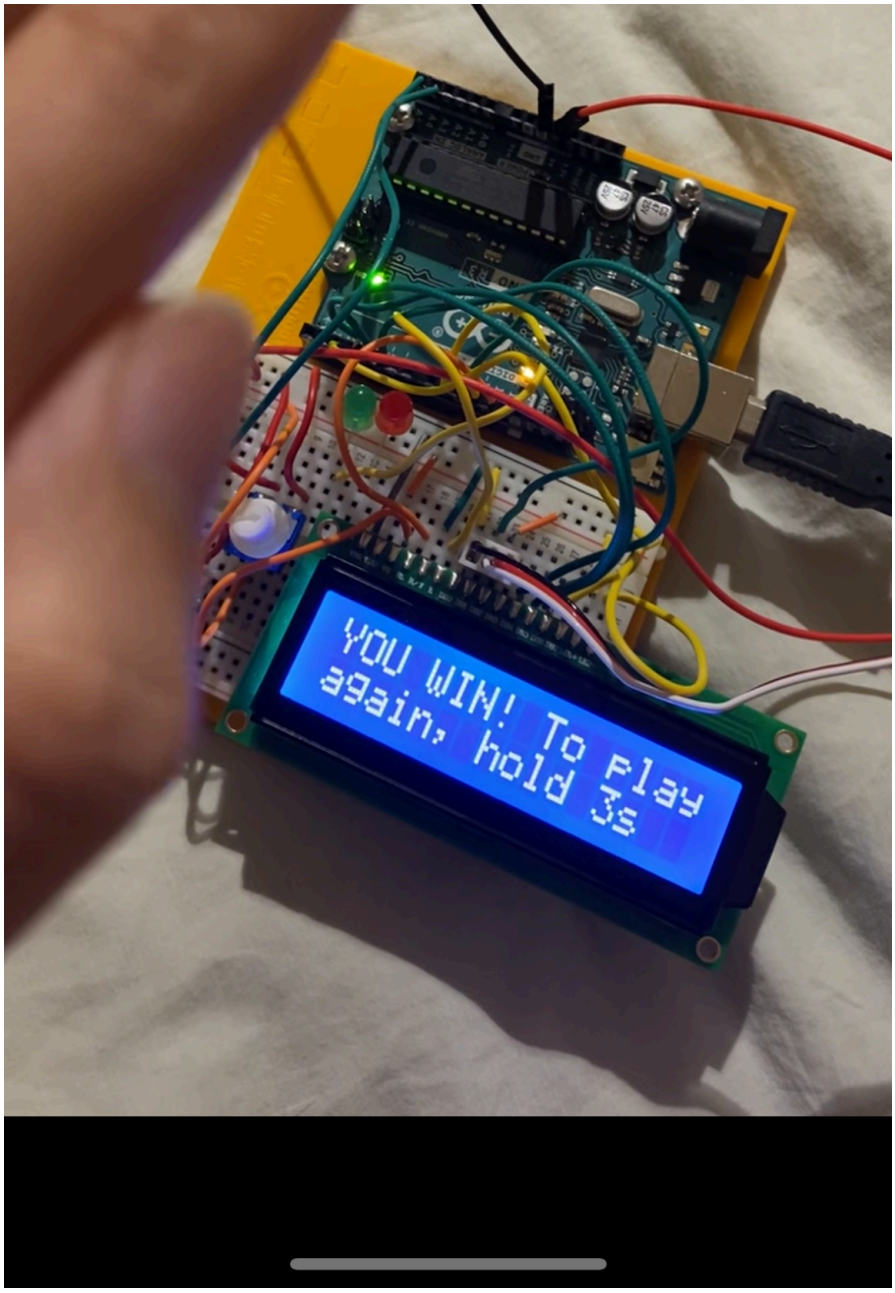
Closed Loop Flappy Bird System: Final Report

Matthew Arboleda, Seolbin Hong, Julia Chun





Closed Loop Flappy Bird System: Final Report
Matthew Arboleda, Seolbin Hong, Julia Chun



5. Results The final Flappy Bird system successfully implemented a real-time closed-loop game using Arduino-controlled hardware. Key gameplay behaviors—including bird motion, collision detection, dynamic difficulty, and game reset—functioned reliably under various conditions. Our testing confirmed that the LCD accurately rendered characters, the VL53L0X sensor provided stable distance readings, and the servo motor's position could be used as both a visual indicator of success and a game progression tracker.

User testing showed that gameplay became increasingly challenging with faster frame rates, and the servo consistently advanced or retreated based on in-game performance. LED

feedback clearly communicated successes and failures, while the win condition (servo reaching ≥150°) provided a tangible end goal. These behaviors confirm that the project meets the objective of creating an interactive, sensor-driven, closed-loop system that delivers both visual and physical feedback based on user input.

Results from parameterized tests and quantitative observations are summarized in the following section.

6. Challenges

When doing this project we encountered some difficulties around certain parts of the game. This included how to display on the LCD screen and make sure that the logic of the game was accurate. When doing collision specifically, we were having trouble with checking to see if the bird/character was “colliding” with the pipe. In order to accomplish this we used a byte array in order to create the pipes and then checked to see if the value was filled. This means that we checked to see if the value in the specific index was “B11111”. If the bird was within the space/y-level that the pipe was filled at then we were able to determine a collision had occurred so that we could adjust the score accordingly. We also had an issue where the bird was originally two pixels, and would split between the two rows of the LCD screen, choosing one row above the other, leading to the collision not always working. We decided to make the bird one pixel instead to help with the calibration of the collision due to the constraints of the display’s speed and increase the possible positions of the user’s movements.

We also initially had trouble with seeing what was happening on the LCD screen. The screen was really dim and it made it difficult to see unless we were directly in front of the screen. In order to fix this problem we were able to adjust the contrast of the screen in order to make the screen bright enough to see from a bit further away and we noticed that we did not plug in the LCD- pin to ground.

Another issue we encountered was trying to adjust the speed of the game. When we were looking at examples, they were all using really slow speeds. We then were able to play around with the concept of a frame limiter so that we could freely adjust the speed of the game/how often the code looped so we could either slow or speed up the game depending on specific conditions (like the score).

7. Reflection/Future Work

This project ended up taking us around 150 total man hours across the 3 of us. Collectively, the research of how to implement a game of this style took us a total of around 20 man hours. We looked at past projects, as well as academic sources to help guide our process and to see how we could extend a game. Finding the proper setup of our circuit board and wiring it up took around 20 hours in total, as it took a while to find a setup that worked with the LCD, arduino kit, and ToF sensor. We had to take multiple sessions rewiring our board to best fit our needs and materials. We also took more time working with the board because some of the wires would

pluck off as we worked on it, and then we had to rearrange it (which we learned by the second time to record our layout in a neat, concise format for repetition). In terms of looking through documentation on how to use the pieces in our project (like the LCD screen and servo motor), it took us around 40 man hours because the LCD had more limitations and properties than we expected. We attempted to create a bitmap of the entire screen for a long time, and then decided to make simpler ones of the characters. And we tried to examine how to manipulate the pixels to get the more dynamic pipe sizes, but it was difficult to achieve even after our efforts. The coding and testing of the game took around 70 man hours due to troubles with collision and trying to ensure better game conditions. The greatest issue we ran into was trying to fix the collision with the pipes and creating dynamic scenes with the character limits and pixel sizes. We also ran into issues with using the LCD because it provided some errors due to its limitations of memory and frame delays. When it would seemingly not turn on, we thought that the issue was with the code, but it turned out that one of our wires was unplugged.

We all took part in research, where Matthew and Seolbin mainly focused on existing projects and extensions of the arduino kit and LCD, and Julia focused on academic sources regarding microcontrollers and games. Matthew created the initial outline of the game with the sensor, bird character and pipe movements. Julia created the servo motor extension and Seolbin created the code for the scoring (and implementation of a high score which was later not used). Julia and Seolbin worked on the circuit board, while Matthew took this time to develop the code with creating dynamic pipes. Seolbin helped debug the code by allowing the character and pipe to merge and implemented a threshold initialized by the user's hand at the beginning of the game. Matthew and Julia also helped to debug and run tests for the collision at this time. Seolbin also attempted to create a bitmap and the pipes that extended into the other row of LCD but kept running into issues regarding collison and scoring. Julia created a circuit visual to help with replication and researched more about the underlying principles and materials of the project.

For future works, we would recommend students to have more practice on working with LCDs before jumping into this project. We felt pressured by the time constraint and thought that we had to jump into coding, but it would have been beneficial for us to take our time and prep with the exercises in the book and with online resources. We would also recommend sharing the arduino kit often with the group members because a lot of our experience with this project was with debugging and testing. Because of this project, we have more experience with working with sensors, sensor data, the arduino kit and manipulating it to create and use meaningful data within our system. We also now have better documentation skills and a greater understanding to be proficient at developmental boards and circuits. To further extend this project for other students, we would ask the students to see if they can create a more dynamic way to use sensors. We tried to create the interactions of the system to be more dynamic, but we would like to see more dynamic ways that sensors can be used.

Closed Loop Flappy Bird System: Final Report
Matthew Arboleda, Seolbin Hong, Julia Chun

Sources:
https://docs.arduino.cc/learn/electronics/lcd-displays/
https://www.scribd.com/document/408455408/FLAPPY-BIRD-GAME-FULL-REPORT-pdf
https://docs.arduino.cc/learn/electronics/lcd-displays/
https://electronics.howstuffworks.com/lcd.htm