Mechanical Pong

Two Paddles, One Ball

Group 23
Brandon Alvarado
Colton Hood
Isaac Morris
Jeremy Tabke

May 10, 2019

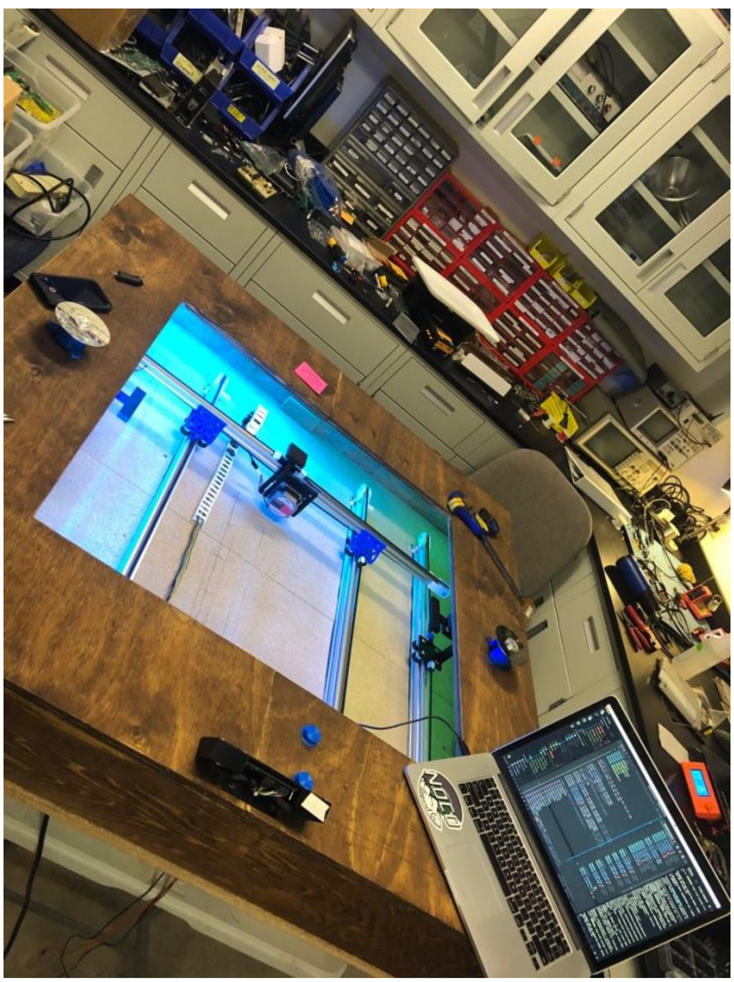


Table of Contents

System Details ***6***

Software Flowcharts **14**

Actuators, Mechanisms, and Hardware **18**

Logic, Processing, and Control **19**

Lessons Learned ***21***

References ***23***

Appendix ***24***

Teensy 3.6 Game Play Code: **24**

PIC16F88 LED Strip Control Code: **34**

Wiring Diagram: **38**

Design Summary

Mechanical Pong is a modern twist on a classic video game, Atari Pong. The project constructed a physical representation of the first video game. The original game can be seen below in figure 1. The project has two game modes, versus and rally. In versus mode, two players face off hitting the ball back and forth with the paddles. If a player fails to return the ball, the opposing player is rewarded a point. Rally mode is single player against the computer. The goal is to return the ball as many times as possible before the player misses.

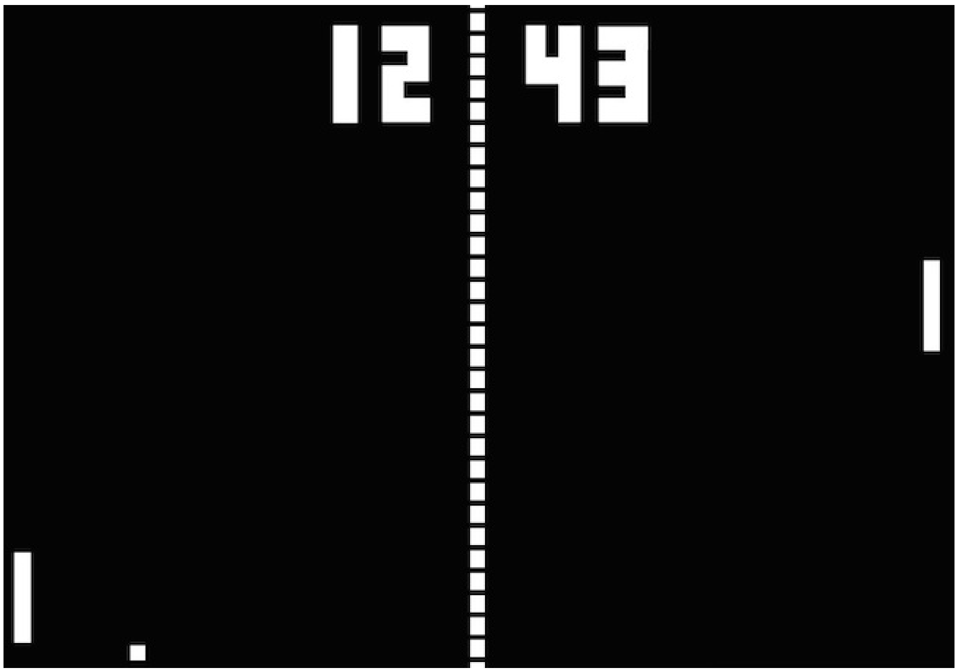


Figure 1: The original pong game layout

Mechanical Pong was constructed with four 20 x 40 aluminum rails, one 20 x 20 aluminum rail, two paddles, a ball, and multiple GT2 pulley and belt systems. The entire system is powered by five stepper motors on a two-dimensional axis system. The ball reacts to the walls and the paddles according to the programmed game dimensions on a Teensy 3.6 board. These dimensions correspond to the boundaries the ball can travel to.

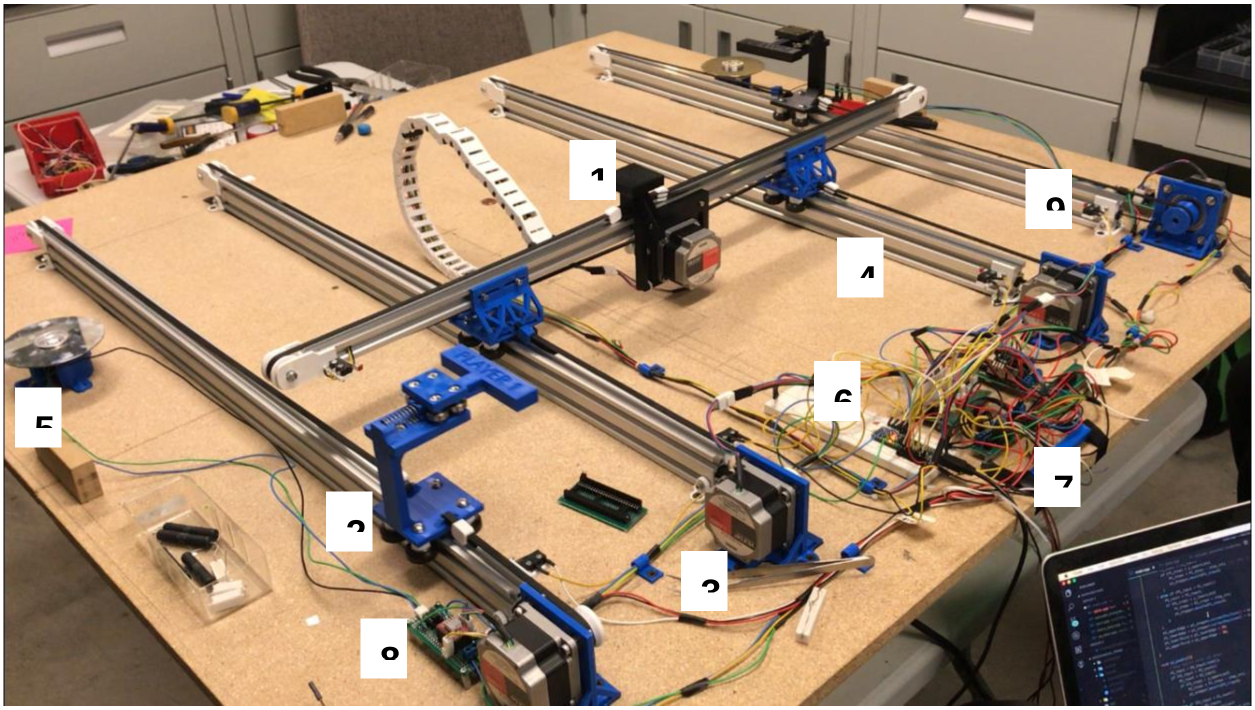


Figure 2: The board shown with the top removed

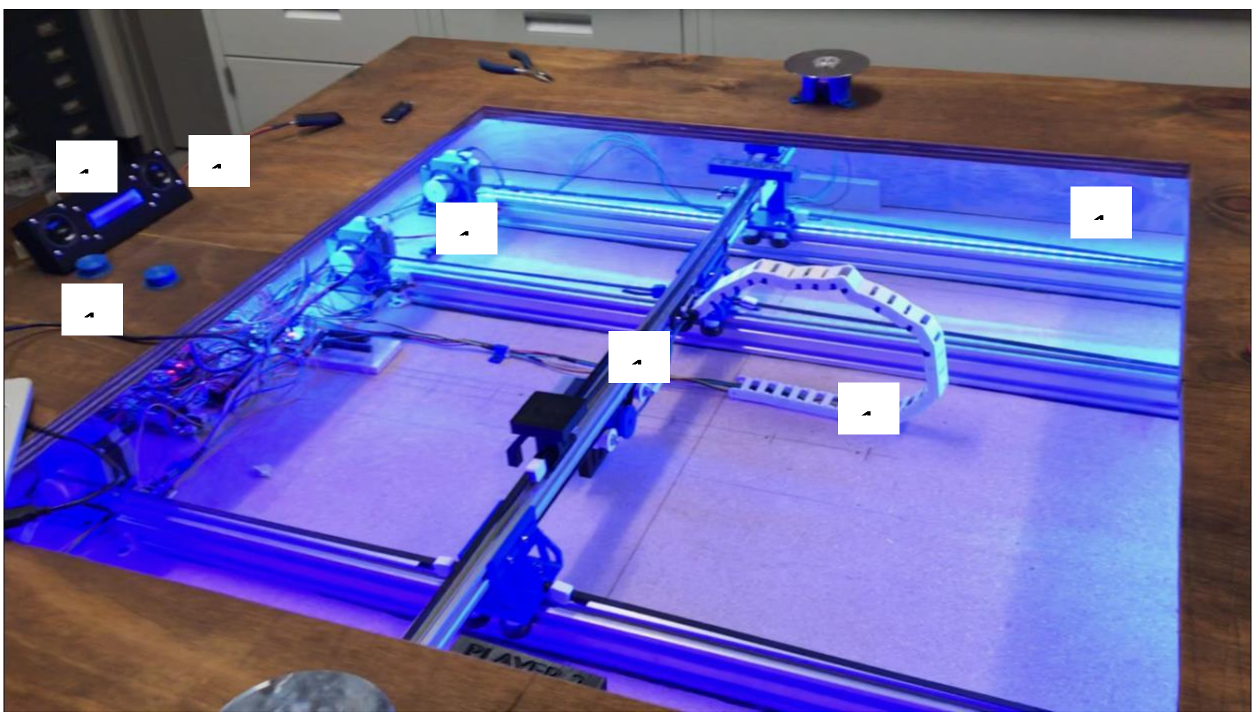


Figure 3: The game board shown with the top attached

Table 1: System Components

1	Ball
2	Paddle
3	Stepper motor
4	20x40 aluminum rail
5	Hard drive motor converted to rotary encoder
6	Teensy 3.6
7	PIC
8	Stepper driver
9	Limit switch
10	1.5W speakers and LM386 amplifier circuit (mounted underneath housing)
11	Buttons to control LCD
12	LCD
13	LED strip
14	GT2 pulley and belt
15	Cable Track
16	20x20 aluminum rail

The paddles are controlled by the user (2). The user input is read from a rotary encoder that the player spins (5). When the encoder is spun clockwise the paddle moves right along the y-axis. When spun counterclockwise the paddle moves left along the y-axis. The ball slides across the 20 x 20 Aluminum rail across the x-axis (16). The rail is simultaneously moving in accordance to the code across the y-axis (4).

The ball motion is controlled by three of the five stepper motors and rails (3). The ball is on two rails instead of one because we felt it was necessary to increase the rigidity of that axis, given the ball would making quick movements, experience force acting on it from the spring in the paddles, and the rail would need to also support the weight of the motor.

The two LED strips were mounted underneath the game board cover, and were controlled directly by the PIC (13, 7). The LEDs were wrapped around the cover of the game board so that they could not be seen, but still would light up the game. Each players side had its own dedicated LED that would flash its color depending on the mechanics of the game. Any ball actions such as scoring, being returned by the player, bouncing off the walls, or being missed would result in its own LED and sound sequence. The speakers used are mounted on top of the board as well (10).

System Details

The system has a simple two-dimensional design. The two paddles are constrained to a one dimensional movement along the y-axis provided by a pulley system along a 20 x 40 Aluminum rail (1a). The ball (1b) follows a more complicated path across two axes of motion. The ball moves along the x-axis and y-axis simultaneously. The x-axis movement is provided by a pulley system along two 20 x 40 Aluminum rails (1c) and the y axis movement is provided by a perpendicular pulley system along a 20 x 20 Aluminum rails (1d) .

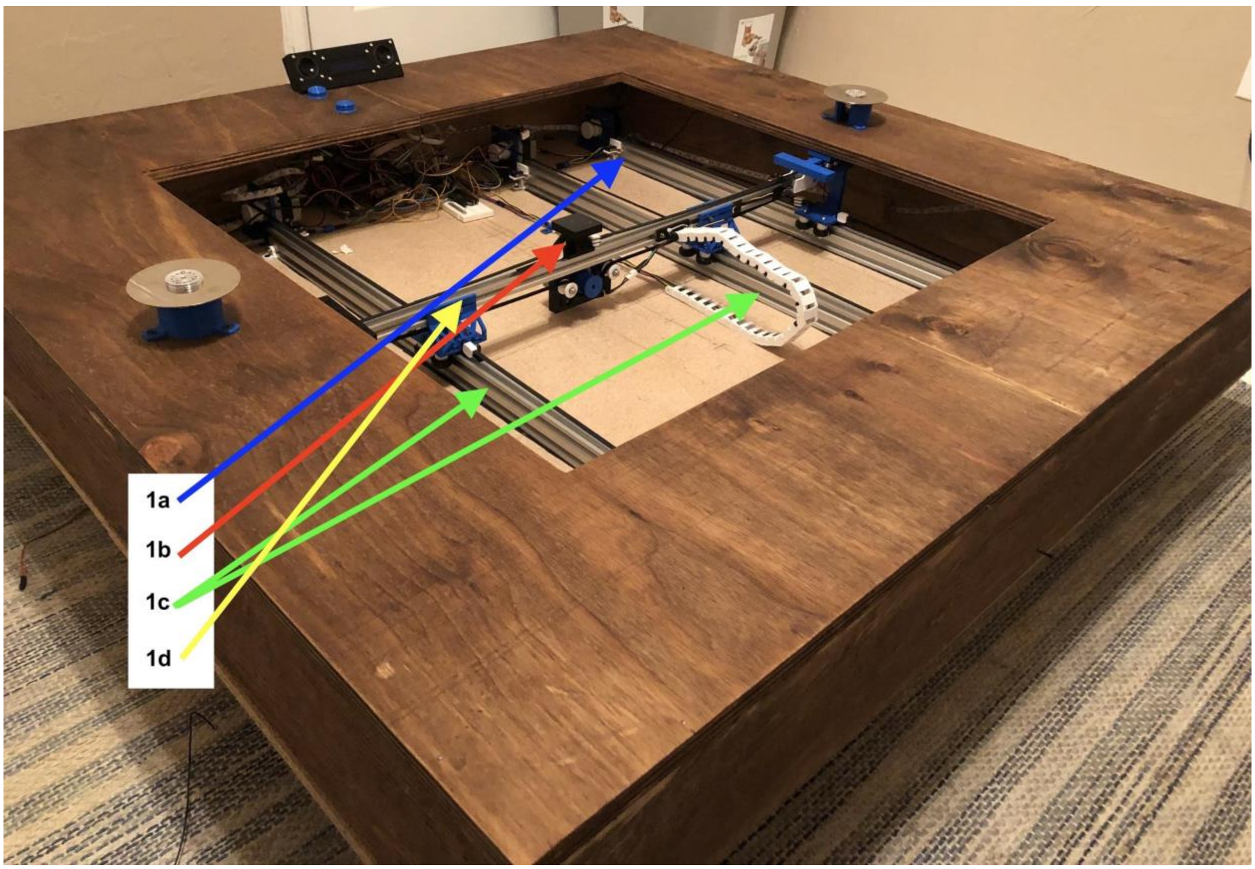


Figure 4: Rail Layout and General Overview

Each pulley system was comprised of a stepper motor (2a), an aluminum rail (2b), a timing belt (2c), a carriage (2d), a limit switch (2e) and two timing pulleys (2f). Each stepper motor provided movement to the carriages across the aluminum rail via the timing belt and timing pulleys and each carriage was different for each rail function. The motor drivers played an important role in handling the distribution of power to the stepper motors by distributing power to each of the four wire coils on each stepper based on the input from the Teensy 3.6 microcontroller.

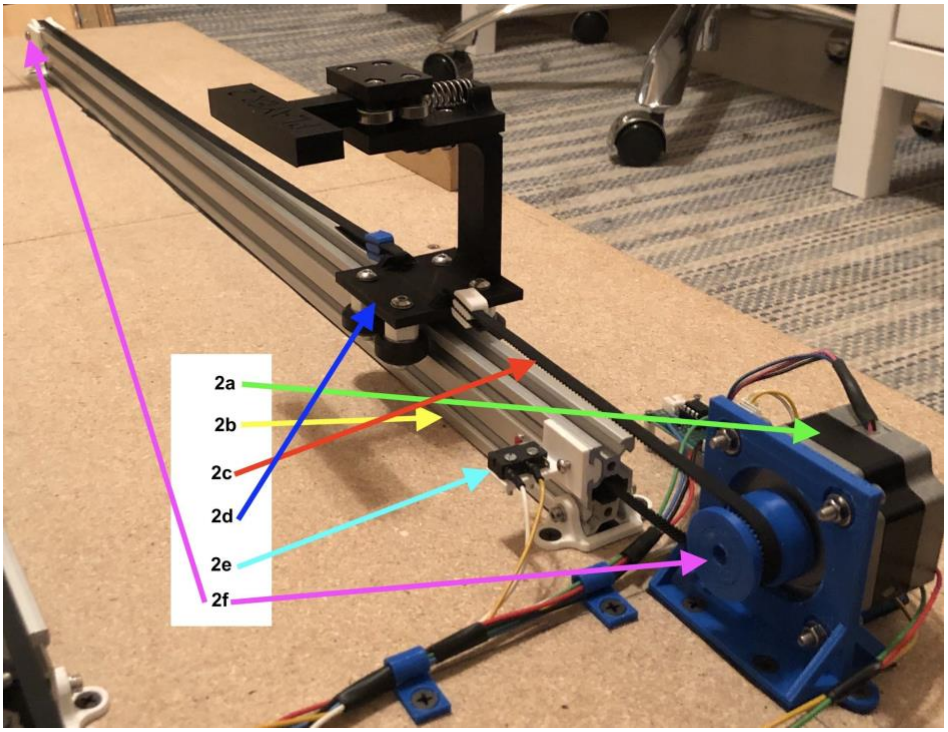


Figure 3: Pulley System

The Teensy 3.6 microcontroller (Figure 4) was the most important component in the project design. The Teensy 3.6 defined the playing field for the game and was constantly reading the positions of each paddle and ball The Teensy communicated with every stepper motor driver, constantly recording the positions of the paddles and ball. Thanks to its 180 MHz ARM Cortex-M4 processor it was able to handle all of the game variables while cycling fast enough to step all 5 stepper motors to maintain mechanical movements.

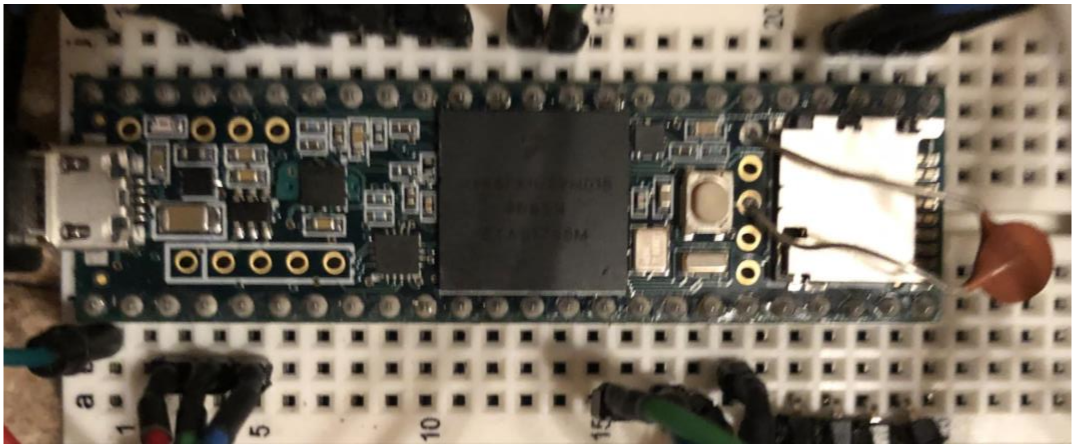


Figure 5: Teensy 3.6 Microcontroller

The ball system is designed with a limit switch contact to home the ball. The attachment to the 20x20 slides just inside the geometry of the 20x20 rail. The mounting points for the belt are kept close to the 20x20 to avoid binding. This is a two part system and both parts were printed without supports to improve print quality.

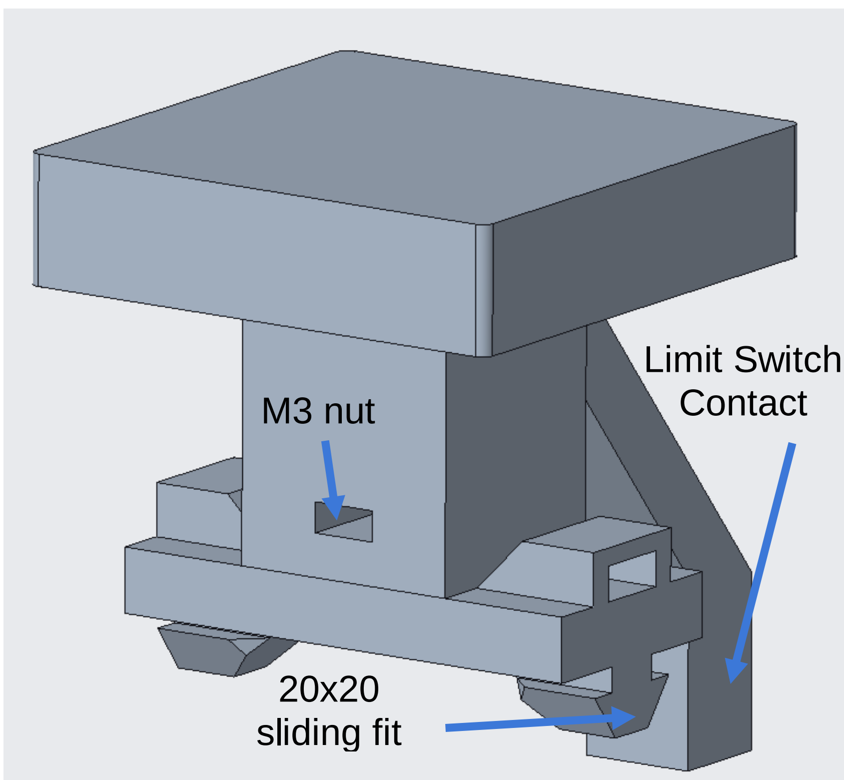


Figure 6: The ball assembly showing the sliding fit portion that locks the ball to the 20x20, the GT2 belt attachment points, the M3 nut slot, and the limit switch contact.

The final paddle design included four bearings that constrained the motion of the translating paddle to one axis. The paddle is attached to a spring along this axis, and the ball contacting the paddle moves the paddle. This is not a vital component of the system, and just adds to the illusion that the ball is being bounced off the paddle. The ball's position is controlled by the code, and does not respond to the paddle being depressed.

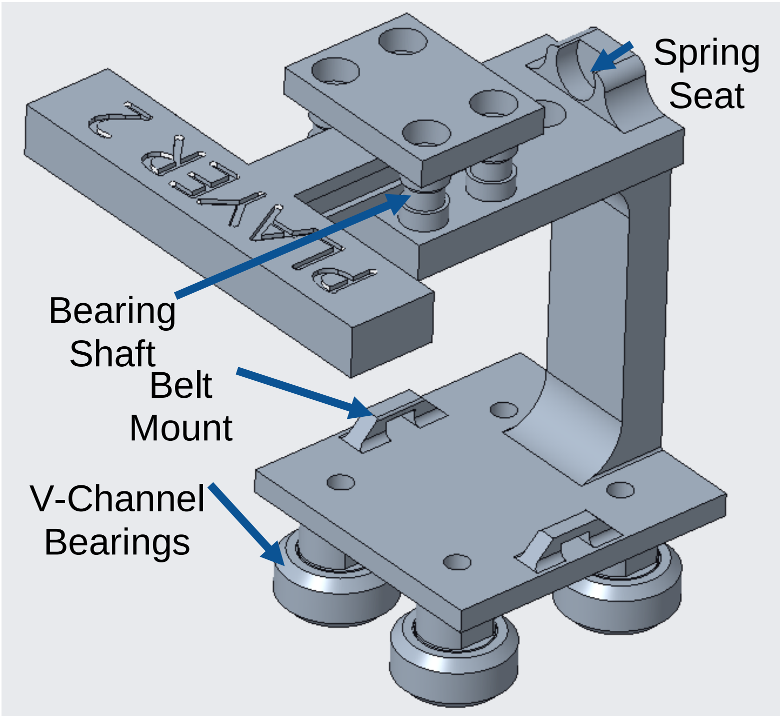


Figure 7: The paddle assembly showing the sliding spring mechanism, the belt attachment points, the v-channel bearings, and the modular design

The assembly is printed in 4 parts, all of which are printed without supports and assembled with M5 nuts and bolts.

The stepper motor mounts were designed with vibration dampening in mind. Bicycle tire inner tubes were used underneath the motor mount to try and isolate each motor from the rest of the system, and the holes used to mount the motor had recessed areas to allow a rubber grommet to sit. The gears were created in CREO, using a GT2 belt profile. This belt is commonly used, cheap, and high functioning. It has an internal fiber layer that resists significant stretching and keeps the belt straight. These features are shown in figures 7 and 8. In figure 8, the dashed line represents the stiff internal fibers in the belt.

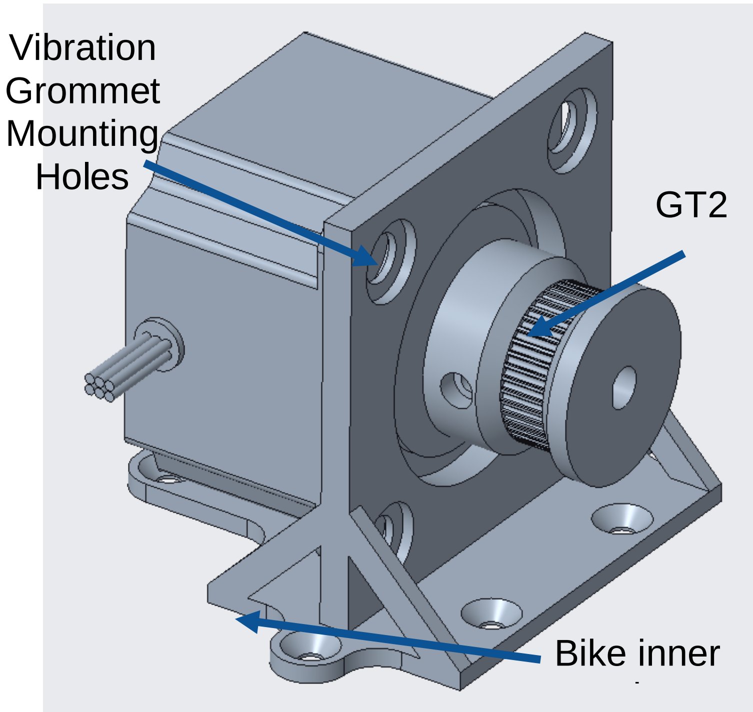


Figure 8: Stepper Motor Mount and Assembly

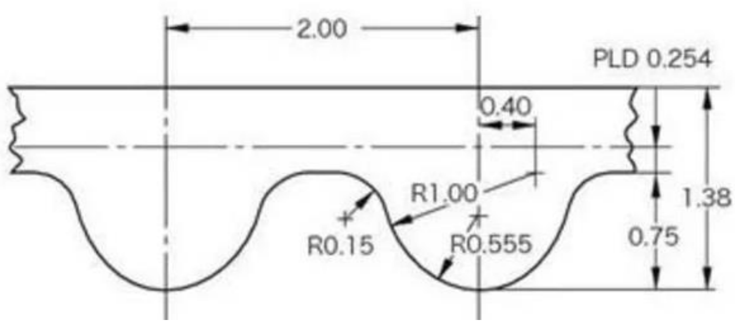


Figure 9: GT2 belt profile

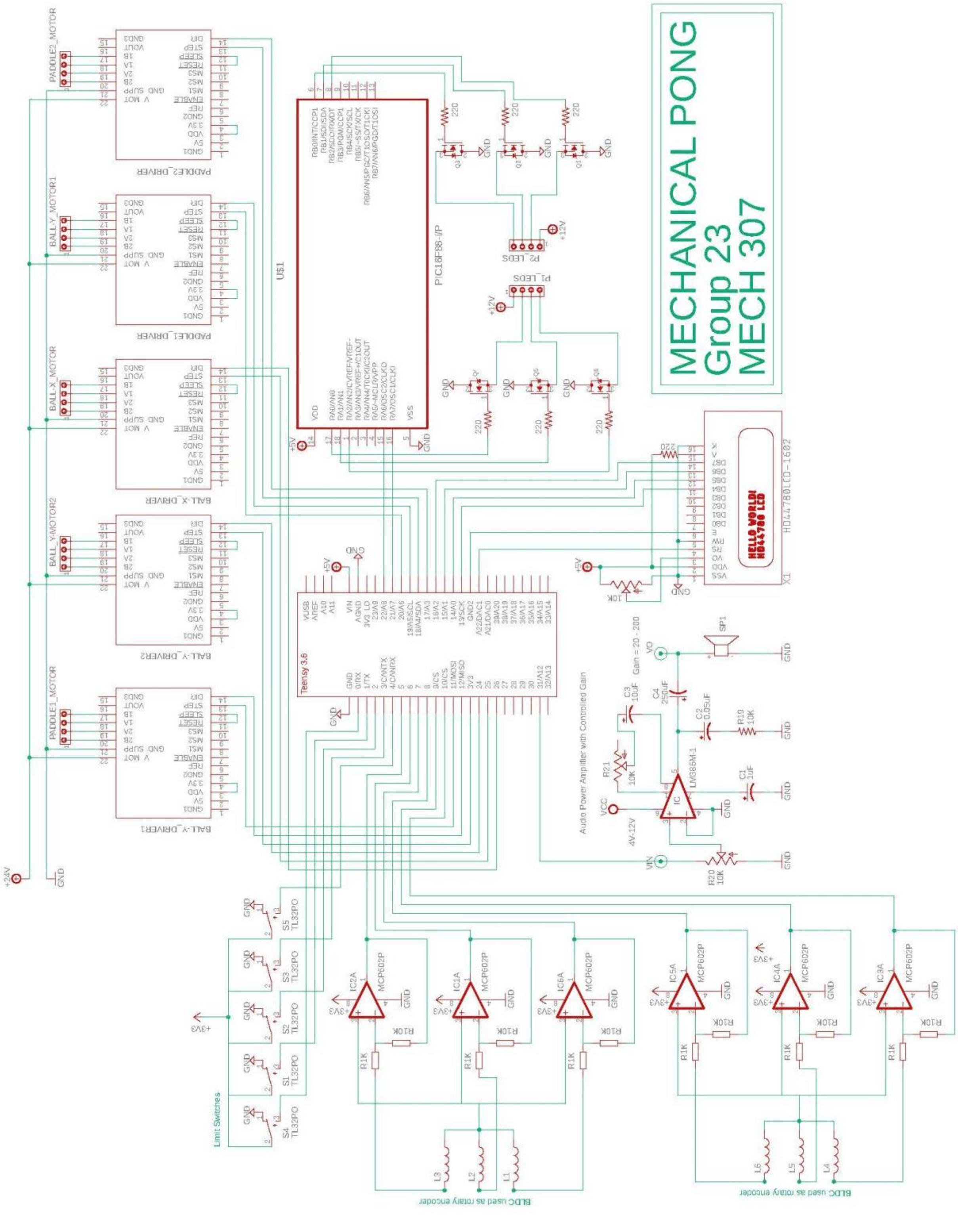
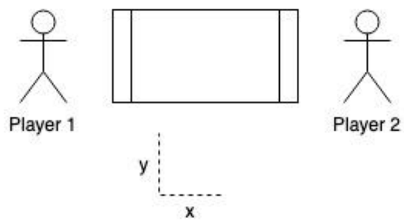
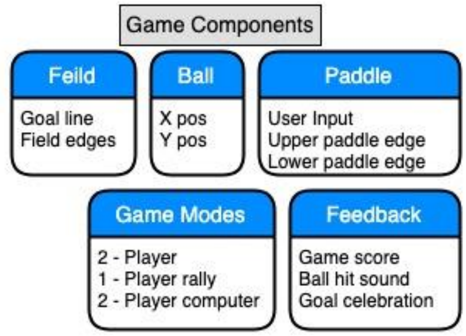
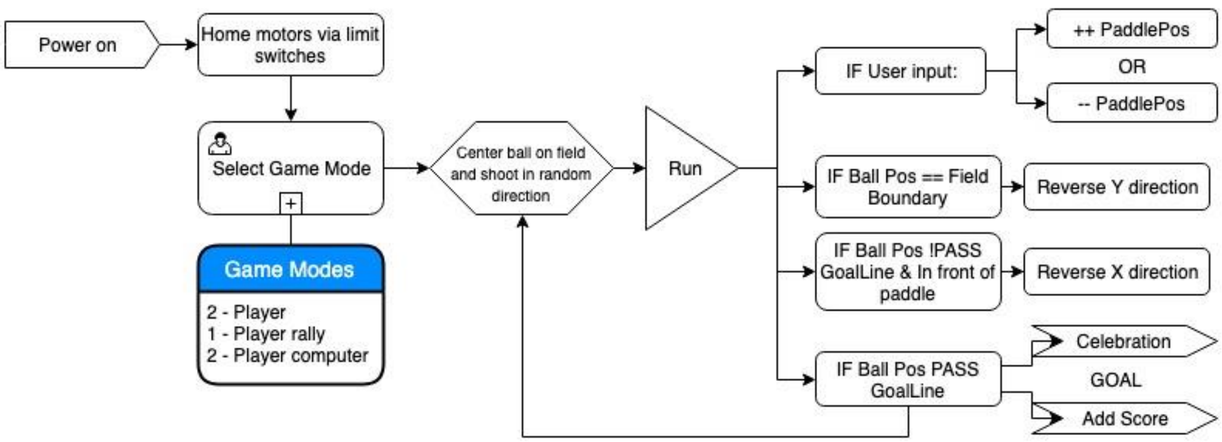


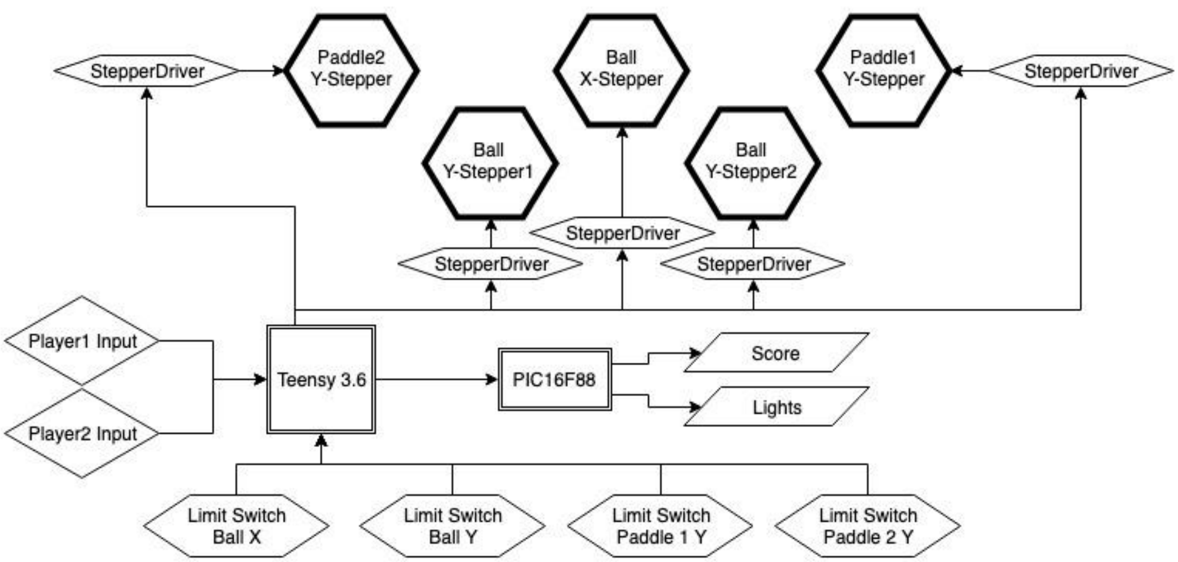
Figure 10: Wiring Diagram of Mechanical Pong System

Above is the full wiring diagram. The large number of inputs into the Teensy can be seen from this diagram, very few of the 60 I/O pins are left over. The circuit details the PIC16F88 LED light control circuit as well. The user paddle inputs are seen as motor coils attached to Schmitt-triggers with hysteresis.


Player 1
Player 2
y
x


Game Components
Feild
Goal line
Field edges
Ball
X pos
Y pos
Paddle
User Input
Upper paddle edge
Lower paddle edge
Game Modes
2 - Player
1 - Player rally
2 - Player computer
Feedback
Game score
Ball hit sound
Goal celebration


Power on
Home motors via limit switches
Select Game Mode
Game Modes
2 - Player
1 - Player rally
2 - Player computer
Center ball on field and shoot in random direction
Run
IF User input:
++ PaddlePos
OR
-- PaddlePos
IF Ball Pos == Field Boundary
Reverse Y direction
IF Ball Pos !PASS GoalLine & In front of paddle
Reverse X direction
IF Ball Pos PASS GoalLine
Celebration
GOAL
Add Score


StepperDriver
Paddle2 Y-Stepper
Ball X-Stepper
Paddle1 Y-Stepper
StepperDriver
Ball Y-Stepper1
Ball Y-Stepper2
StepperDriver
StepperDriver
StepperDriver
Player1 Input
Player2 Input
Teensy 3.6
PIC16F88
Score
Lights
Limit Switch Ball X
Limit Switch Ball Y
Limit Switch Paddle 1 Y
Limit Switch Paddle 2 Y

The functional elements used to produce this project are a microcontroller to handle the game logic and operating software. The PIC16F88 drives the LED strip control programs interfaced to the Teensy microcontroller. Five stepper motors are used and driven by the A4988 motor driver. Attached to each rail the stepper motor operates on is a limit switch to set the initial hard limit for the software and align the system. An LCD and two speakers are used to provide game feedback including celebration sounds and display player scores.

Software Flowcharts

Pictures:

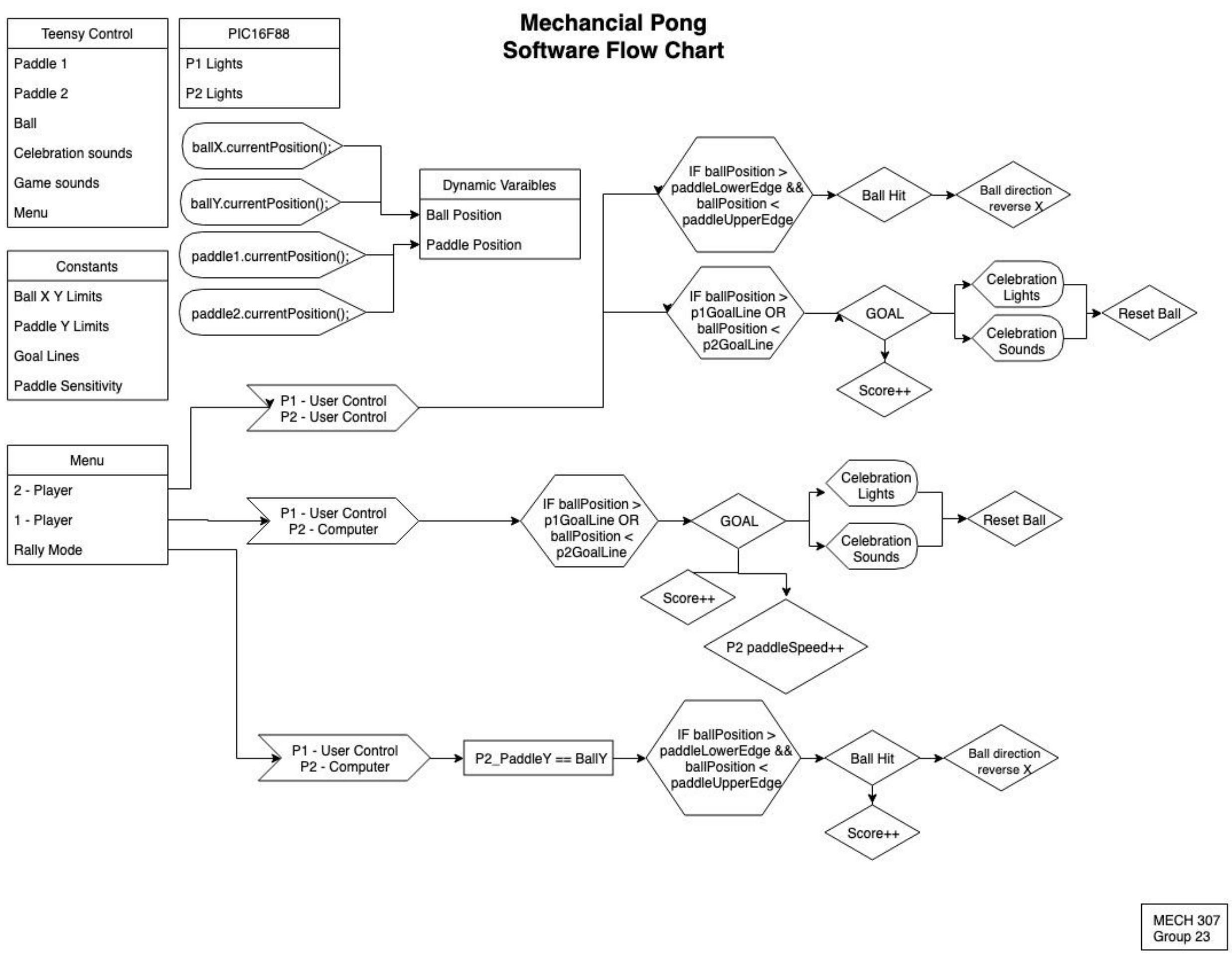


Figure 11: Software Flow Chart

Explanations:

The software handles the majority of the control for this project. The initial homing is set by the limit switches during the initial homing routine. This is a hard limit to the end of the rail. After this absolute zero has been set on all of the rails the game field is defined in the software and relies on an open-loop control system for tracking of the stepper motor positions. The game piece edges are then calculated in the software to determine where on the paddle the ball hits and redirect the ball in the y-direction if the ball hits the paddle on the upper or lower third.

In the 2-player game mode both of the paddles are controlled via the user input connected to each paddle. 1-player mode takes user input to control the paddle and the player 2 paddle is controlled via the software. In this mode the max speed of paddle 2 is set significantly lower than the ball y motor speeds. The software attempts to track the paddle to the ball movements. Each goal scored by player 1 increases the max speed of the player 2 paddle and gets successively harder until the unbeatable round when the paddle speed reaches the max speed of the ball. The final mode is rally mode in which the player 2 paddle tracks the ball perfectly and returns all of the shots by player 1. At each return the score for player 1 increases.

Design Evaluation

Mechanical Pong has successfully achieved the requirements stated in the functional element categories. In some areas the system has gone above expectations and anticipate grade adjustments. This is explained and stated in the following paragraphs.

Output Display

The output display used in Mechanical Pong was an LCD display (Figure 12a). This was the center for selecting game mode and displaying score.

LED strips were placed under the case. The LED lights were a backlight and reacted to a player scoring during the game and player paddle hits.

Audio Output Device

The speakers (Figure 12b) used for Mechanical Pong were extracted from a broken UE Boom Bluetooth Speaker. Pre-recorded voices and Pong sounds were downloaded from the internet and saved on to an SD card. The audio files were powered through a Teensy 3.6 board. An LM386 Audio Amplifier was used to produce higher volume.

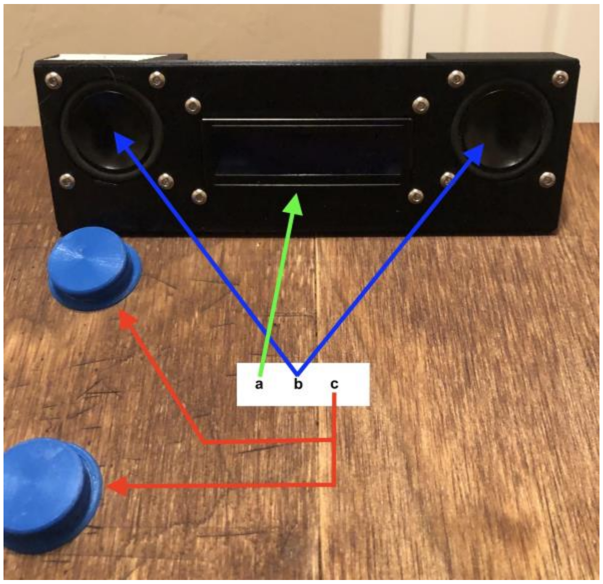


Figure 12: LCD screen, speakers, and game buttons

Manual User Input

Two forms of user input were incorporated in this project. The first are two buttons to control the LCD Display. One button (Figure 12c) is used to toggle through the options provided and the second to select the desired option. The second user input was rotary encoders. A brushless DC motor was attached to a Schmitt-trigger with hysteresis and repurposed to be used as a rotary encoder (Figure 13). The encoders are used to control the direction of the paddles. When the encoder is spun clockwise by the user, the paddle moves right along the rails. When spun counterclockwise the paddle slides left.

Figure 13: Manual User Input - BLDC Motor Used as Encoder

Automatic Sensor

Limit switches (Figure 14)were placed at the end of each rail. The switches determined the maximum amount of steps the paddles, ball, and rails were allowed to travel. The switches were also used to home each component of the system before each game play.

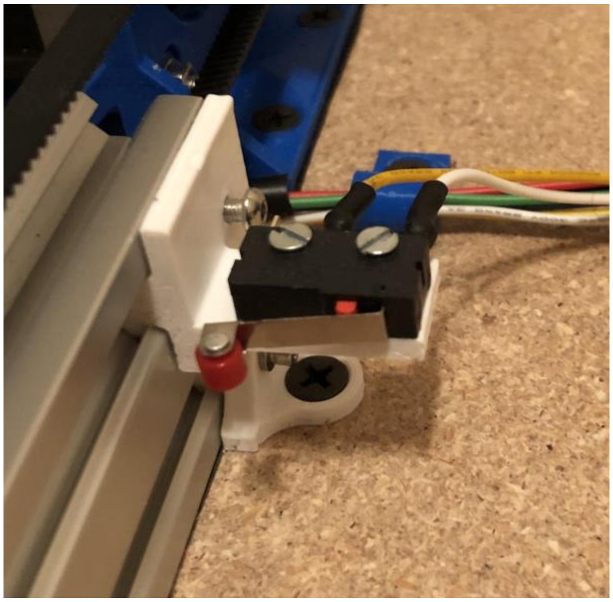


Figure 14: Limit Switch Mounted on Rail

Actuators, Mechanisms, and Hardware

A Vexta 2-Phase Bipolar Stepper Motor (Figure 14)was used to run each pulley system. A total of five were used to complete the project, one for each paddle, two to pull the ball rail across the y-axis, and one to slide the ball across the x-axis rail.

Modular perf boards were soldered together with the components for each player system, a motor controller, and Schmitt-trigger circuit (Figure.

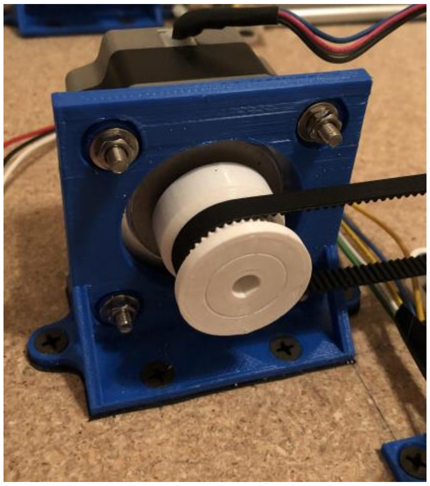


Figure 15: Vexta stepper mounted in the system

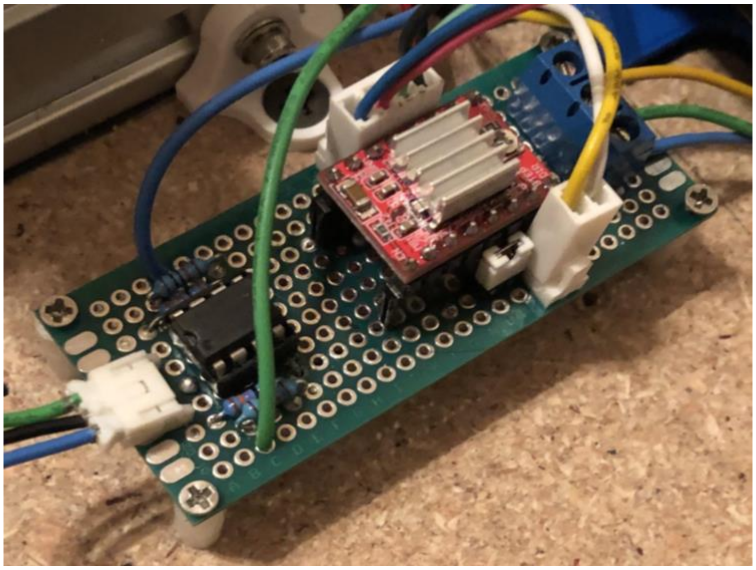


Figure 16: Stepper Motor and Motor Driver Circuit

Logic, Processing, and Control

The main microcontroller in the system was the Teeny 3.6, (Figure 3) it handled all of the motor executions and game play controls. It communicated with the PIC 16488, which was in charge of reading the inputs from the Teensy 3.6 microcontroller and displaying the correct LED color output for each player for each game play scenario (Figure 17).

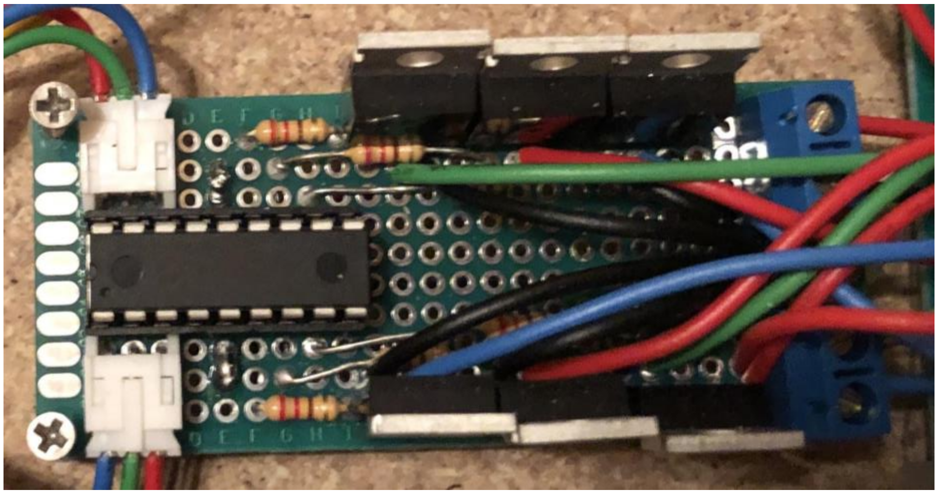


Figure 17: PIC16F88 LED Strip Circuit

Table 2: Partial Parts List

Part Name	Model Number	Vendor	Price
Dell Power Supply	M375P-00	Donation*	$0.00
Vexta 2-Phase Bipolar Stepper Motors	PX243-02AA	Donation*	$0.00
Teensy 3.6 Microcontroller	3266	Amazon	$38.00
V-Slot 20 x 40 and 20 x 20 Linear Rails	215-LP 280-LP	OpenBuilds Part Store	$93.39
200 Times Gain 5V-12V LM386 Audio Amplifier Module	B01FDD3FYQ	Amazon	$7.98
12V RGB EconoLED Flex Strips	SYNCHKG02335 5	Amazon	$9.99
GT2 Timing Belt	073	Amazon	$9.99
Stepper Motor Driver with Heat Sink	A4988-5P	Amazon	$14.98
Double Sided Proto Boards	AMA-18-581	Amazon	$15.99
V-Slot Ball Bearing	CR-10S	Amazon	$23.98
12V to 24V Step Up Converter Regulator	INT-12T24-10A	Amazon	$19.99
Brushless DC Motors	Unavailable	CSU Surplus	$3.60

**Parts were salvaged from broken computers and previous senior design projects.*

Lessons Learned

The first design for our springing action on our paddle involved three aluminum rod which slid in holes incorporated in our paddle print. These slid when pushed on from directly in front, but any time there was a moment applied the rods bound up. In the future we would never rely on sliding fit mechanical components when there is any chance an applied moment/ torque will be applied. This can be avoided by paying attention to the 2:1 bearing ratio in design, but this also constrains geometry significantly. When possible, use bearings for linear motion to prevent binding

There were many prints that had issues getting the hardware to fit correctly. We learned that there could be some general tolerances that worked well for the printed components on our printers. We modeled to allow tolerances in to create different fit types. If geometry was seen parallel to the print bed, oversizing geometry was necessary. The values below are for geometry parallel with the print bed.

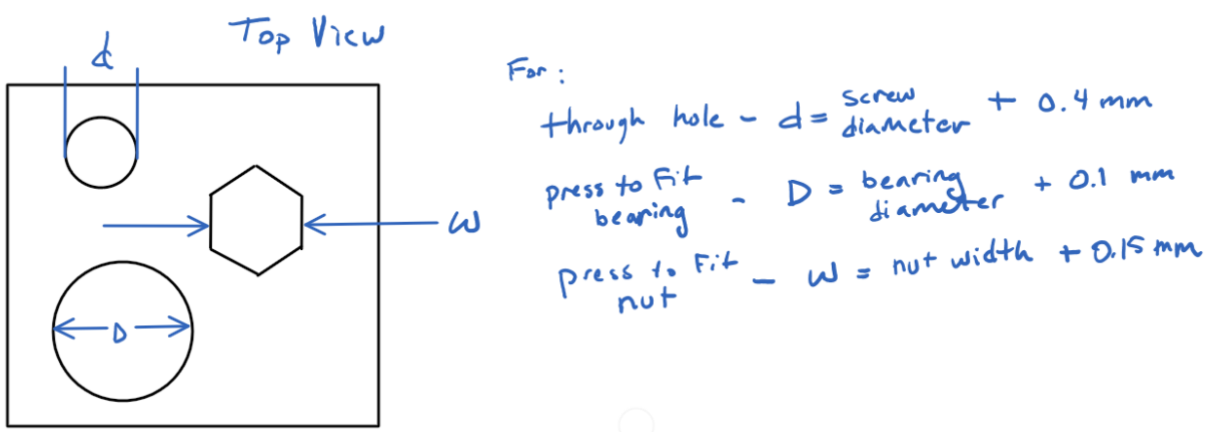


Figure 18: Printing Tolerance

We spent time modeling some brackets and small parts that we could have found online that were already created. Take advantage of open source CAD websites. Thingiverse had models for the cable chain, belt clips, and other forms of cable management that were used in this design. These would have been very time consuming to recreate in a CAD software, and instead could be downloaded for free.

The Dell Power Supply determined the fate of this project. When the power supply is not connected to a computer's motherboard, the power supply has an under-over voltage protection that automatically shorts the system and shuts down. We discovered this issue late into the testing period of the project and did not have enough time to fix the problem. With research we discovered how to overcome the under-over voltage protection. In order to fix the problem specific pins on the power supply supervisor integrated circuit must be grounded. This would disable the undervoltage protection and give the system uninterrupted power.

References

"1 MHz, Low-Power Op Amp." Microchip. 2009. https://ww1.microchip.com/downloads/en/DeviceDoc/21733j.pdf.

"18/20/28-Pin Enhanced Flash MCUs with NanoWatt Technology." Microchip. 2013. http://ww1.microchip.com/downloads/en/DeviceDoc/30487D.pdf.

"AccelStepper Library for Arduino." AccelStepper. https://www.airspayce.com/mikem/arduino/AccelStepper/index.html.

Allegro. "DMOS Microstepping Driver with Translator And Overcurrent Protection." Microsystems. 2014. https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf.

"Disabling Under/over Voltage Protection on ATX Power Supplies." Hackaday. August 30, 2013. https://hackaday.com/2013/08/30/disabling-underover-voltage-protection-on-atx-power-supplies.

"PICBASIC PRO Compiler Reference Manual." Micro Engineering Labs Inc. 2013. http://pbp3.com/downloads/PBP_Reference_Manual.pdf.

Schroeder, Jonathan R. "Linear Bearings: Understanding the 2:1 Ratio and How to Overcome the Stick-Slip Phenomenon." Machine Design. September 29, 2017.https://www.machinedesign.com/motion-control/linear-bearings-understanding-21-ratio-and-how-overcome-stick-slip-phenomenon.

Appendix

Teensy 3.6 Game Play Code:

---START CODE---

```
//library import
#include <Arduino.h>
#include <LiquidCrystal.h>
#include <AccelStepper.h>
#include <Encoder.h>
#include <pitches.h>
#include <GameConstants.h>
#include <pitches.h>
#include <Audio.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <SerialFlash.h>

//connect lcd
const int rs = 7, en = 8, d4 = 9, d5 = 10, d6 = 11, d7 = 12;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// GUItool: begin automatically generated code
AudioPlaySdWav           playSdWav1;     //xy=163,80
AudioOutputAnalogStereo  dacs1;          //xy=405,61
AudioConnection          patchCord1(playSdWav1, 0, dacs1, 0);
AudioConnection          patchCord2(playSdWav1, 1, dacs1, 1);
// GUItool: end automatically generated code
#define SDCARD_CS_PIN    BUILTIN_SDCARD

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {
 4, 8, 8, 4, 4, 4, 4, 4
};

//connect encoder
Encoder P1_Input(P1_CW_lead, P1_CCW_lead);
Encoder P2_Input(P2_CW_lead, P2_CCW_lead);
int step_int = 30; //encoder sensitivity
//connect steppers
```

```
//1/4 stepping w/ 40 tooth pulleys == 10 step/mm
int step_mm = 10;
//paddle motors
AccelStepper p1_stepper(1, p1_STEP_PIN, p1_DIR_PIN); //set mode to DRIVER, STEP pin 6,
DIR pin 7
AccelStepper p2_stepper(1, p2_STEP_PIN, p2_DIR_PIN); //set mode to DRIVER, STEP pin 6,
DIR pin 7
//ball motors
AccelStepper ball_y1_stepper(1, y1_STEP_PIN, y1_DIR_PIN); //set mode to DRIVER, STEP
pin 6, DIR pin 7
AccelStepper ball_y2_stepper(1, y2_STEP_PIN, y2_DIR_PIN); //set mode to DRIVER, STEP
pin 6, DIR pin 7
AccelStepper ball_x_stepper(1, x_STEP_PIN, x_DIR_PIN); //set mode to DRIVER, STEP pin
6, DIR pin 7

//CORRECT THESE VALUES (just placeholders)*****!!!!
int p1_upperEdge;
int p1_middle;
int p1_lowerEdge;
int p1_lowerThird;
int p1_upperThird;
int p2_upperEdge;
int p2_middle;
int p2_lowerEdge;
int p2_lowerThird;
int p2_upperThird;
int ball_p2_edge;
int ball_p1_edge;
int ball_upperEdge;
int ball_lowerEdge;
int const pxEdge_toZero = step_mm * 32; //measurment for setting goallines
int const y_upperLim = 8000; //8000 // 8500(ylen) - 300(limswitch to edge) -
200(paddle lower edge) steps
int const y_lowerLim = 200; //200
int const y_center = y_upperLim / 2;
int const x_upperLim = 8180; //8180
int const x_lowerLim = 220; //220
int const x_center = x_upperLim / 2;
int const x_p2Goal = x_upperLim - pxEdge_toZero;
int const x_p1Goal = x_upperLim - pxEdge_toZero;
```

```
//for edge tracking of paddle and ball
int paddle_len = step_mm * 95; //step/mm * mm == steps
int pyEdge_toZero = step_mm * 19; //step/mm * mm == steps
int ball_width = step_mm * 45; //step/mm * mm == steps
int bxEdge_toZero = step_mm * 18;
int byEdge_toZero = step_mm * 7;

//these need to be updated********!!!!!
//homing step counter
int p1_homing;
int p2_homing;
int y1_homing;
int y2_homing;
int x_homing;
//encoder input
int P1_input;
int P2_input;
int P1_count = 0;
int P1_steps = 4000; //set to start position
int P2_count = 0;
int P2_steps = 4000; //set to start position

//score variables
int p1_score = 0;
int p2_score = 0;

void reset_ball(){
   ball_x_stepper.moveTo(x_center);
   while (ball_x_stepper.distanceToGo() != 0){
       ball_x_stepper.run();
   }
   ball_y1_stepper.moveTo(y_center);
   ball_y2_stepper.moveTo(y_center);
   while (ball_y1_stepper.distanceToGo() != 0 && ball_y2_stepper.distanceToGo() != 0){
       ball_y1_stepper.run();
       ball_y2_stepper.run();
   }
}

void update_score(){
   lcd.setCursor(0,1);
   lcd.print("P1:");
```

```
    lcd.setCursor(4,1);
    lcd.print(p1_score);
    lcd.setCursor(11,1);
    lcd.print("P2:");
    lcd.setCursor(14,1);
    lcd.print(p2_score);
}

void p1_goal(){
    //what to do when p1 scores
    //have switch cases for game modes
    digitalWrite(P1_goalPin, HIGH);
    //delay(100); //add if necessary
    digitalWrite(P1_goalPin, LOW);
    p1_score++;
    update_score();
    reset_ball();
}

void p2_goal(){
    //what to do when p2 scores
    digitalWrite(P2_goalPin, HIGH);
    digitalWrite(P2_goalPin, LOW);
    p2_score++;
    update_score();
    reset_ball();
}

void ball_y_movement(){
    if (ball_y1_stepper.currentPosition() >= y_upperLim &&
ball_y2_stepper.currentPosition() >= y_upperLim){
        ball_y1_stepper.moveTo(y_lowerLim);
        ball_y2_stepper.moveTo(y_lowerLim);
        playSdWav1.play("BOUNCE.wav");
    } else if (ball_y1_stepper.currentPosition() <= y_lowerLim &&
ball_y2_stepper.currentPosition() <= y_lowerLim){
        ball_y1_stepper.moveTo(y_upperLim);
        ball_y2_stepper.moveTo(y_upperLim);
    }
    ball_y1_stepper.run();
    ball_y2_stepper.run();
    ball_upperEdge = ball_y2_stepper.currentPosition() + byEdge_toZero + ball_width;
```

```
    ball_lowerEdge = ball_y2_stepper.currentPosition() + byEdge_toZero;
}

void ball_x_movement(){
    //detect paddle hits or goal
    //p1
    if (ball_p1_edge == x_p1Goal && (ball_lowerEdge < p1_upperEdge || ball_upperEdge >
p1_lowerEdge)){
        ball_x_stepper.moveTo(x_upperLim);
        //redirect hits
        if (ball_lowerEdge > p1_upperThird){
            ball_y1_stepper.moveTo(y_upperLim);
            ball_y2_stepper.moveTo(y_upperLim);
        } else if (ball_upperEdge < p1_lowerThird){
            ball_y1_stepper.moveTo(y_lowerLim);
            ball_y2_stepper.moveTo(y_lowerLim);
        }
        playSdWav1.play("BALLHIT1.wav");
    } else if (ball_p1_edge > x_p1Goal){
        p1_stepper.stop();
        p2_goal();
    }
    //p2
    if (ball_p2_edge == x_p2Goal && (ball_lowerEdge < p2_upperEdge || ball_upperEdge >
p2_lowerEdge)){
        ball_x_stepper.moveTo(x_lowerLim);
        //redirect hits
        if (ball_lowerEdge > p2_upperThird){
            ball_y1_stepper.moveTo(y_upperLim);
            ball_y2_stepper.moveTo(y_upperLim);
        } else if (ball_upperEdge < p2_lowerThird){
            ball_y1_stepper.moveTo(y_lowerLim);
            ball_y2_stepper.moveTo(y_lowerLim);
        }
        playSdWav1.play("BALLHIT.wav");
    } else if (ball_p2_edge > x_p2Goal){
        p2_stepper.stop();
        p1_goal();
    }
    ball_p1_edge = ball_x_stepper.currentPosition() + bxEdge_toZero;
    ball_p2_edge = ball_x_stepper.currentPosition() + bxEdge_toZero + ball_width;
}
```

```
void ball(){
   //call ball x movement
   ball_x_movement();
   //call ball y movement
   ball_y_movement();
}

void p1_paddle(){
   //have switch cases for modes
   P1_input = P1_Input.read();
   if (P1_input > P1_count){
       P1_count = P1_input;
       if (P1_steps > y_upperLim){
           P1_steps = P1_steps - step_int; //if this is too slow increase by multiple
P1_steps like 5 or 10
           p1_stepper.moveTo(P1_steps);
       }
   } else if (P1_input < P1_count){
       P1_count = P1_input;
       if (P1_steps < y_lowerLim){
           P1_steps = P1_steps + step_int;
           p1_stepper.moveTo(P1_steps);
       }
   }
   p1_upperEdge = p1_stepper.currentPosition() + pyEdge_toZero + paddle_len;
   p1_lowerEdge = p1_stepper.currentPosition() + pyEdge_toZero;
   p1_lowerThird = p1_lowerEdge + 30;
   p1_upperThird = p1_upperEdge - 30;
}

void p2_paddle(){
   //have switch cases for modes
   P2_input = P2_Input.read();
   if (P2_input > P2_count){
       P2_count = P2_input;
       if (P2_steps > y_upperLim){
           P2_steps = P2_steps - step_int; //if this is too slow increase by multiple
P2_steps like 5 or 10
           p2_stepper.moveTo(P1_steps);
       }
   } else if (P2_input < P2_count){
```

```
        P2_count = P2_input;
        if (P2_steps < y_lowerLim){
            P2_steps = P2_steps + step_int;
            p2_stepper.moveTo(P2_steps);
        }
    }
    p2_upperEdge = p2_stepper.currentPosition() + pyEdge_toZero + paddle_len;
    p2_lowerEdge = p2_stepper.currentPosition() + pyEdge_toZero;
    p2_lowerThird = p2_lowerEdge + 30;
    p2_upperThird = p2_upperEdge - 30;
}

void homing(){
    pinMode(p1_homePin, INPUT_PULLUP);
    pinMode(p2_homePin, INPUT_PULLUP);
    pinMode(y1_homePin, INPUT_PULLUP);
    pinMode(y2_homePin, INPUT_PULLUP);
    pinMode(x_homePin, INPUT_PULLUP);
    //home paddles first
    //set stepper homing speed
    //p1
    p1_stepper.setCurrentPosition(0);
    p1_stepper.setPinsInverted(true);
    p1_stepper.setMaxSpeed(2000);
    p1_stepper.setAcceleration(2000);
    p1_homing = 1;
    while (digitalRead(p1_homePin)){
        p1_stepper.moveTo(p1_homing);
        p1_stepper.run();
        p1_homing--; //depends on motor direction
        delay(5);
    }
    p1_stepper.setCurrentPosition(0);
    p1_stepper.moveTo(y_center); //send paddles to center
    while (p1_stepper.distanceToGo() != 0){
        p1_stepper.run();
    }
    //p2
    p2_stepper.setCurrentPosition(0);
    p2_stepper.setMaxSpeed(2000);
    p2_stepper.setAcceleration(2000);
    p2_homing = 1;
```

```
while (digitalRead(p2_homePin)){
    p2_stepper.moveTo(p2_homing);
    p2_stepper.run();
    p2_homing--; //depends on motor direction
    delay(5);
}
p2_stepper.setCurrentPosition(0);
p2_stepper.moveTo(y_center); //send paddles to center
while (p2_stepper.distanceToGo() != 0){
    p2_stepper.run();
}

//home ball y
ball_y1_stepper.setPinsInverted(true);
ball_y1_stepper.setCurrentPosition(0);
ball_y1_stepper.setMaxSpeed(2000);
ball_y1_stepper.setAcceleration(2000);
ball_y2_stepper.setPinsInverted(true);
ball_y2_stepper.setCurrentPosition(0);
ball_y2_stepper.setMaxSpeed(2000);
ball_y2_stepper.setAcceleration(2000);
y1_homing = 1;
y2_homing = 1;
while (digitalRead(y1_homePin) || digitalRead(y2_homePin)){
    if (digitalRead(y1_homePin) == HIGH){
        ball_y1_stepper.moveTo(y1_homing);
        ball_y1_stepper.run();
        y1_homing--;
        delay(5);
    }
    if (digitalRead(y2_homePin) == HIGH){
        ball_y2_stepper.moveTo(y2_homing);
        ball_y2_stepper.run();
        y2_homing--;
        delay(5);
    }
}
ball_y1_stepper.setCurrentPosition(0);
ball_y2_stepper.setCurrentPosition(0);

ball_y1_stepper.moveTo(y_lowerLim);
ball_y2_stepper.moveTo(y_lowerLim);
```

```
  while ((ball_y1_stepper.distanceToGo() != 0) || (ball_y2_stepper.distanceToGo() !=
0)){
      ball_y1_stepper.run();
      ball_y2_stepper.run();
  }
  //home ball x
  ball_x_stepper.setCurrentPosition(0);
  ball_x_stepper.setMaxSpeed(2000);
  ball_x_stepper.setAcceleration(2000);
  x_homing = 1;
  while (digitalRead(x_homePin)){
      ball_x_stepper.moveTo(x_homing);
      ball_x_stepper.run();
      x_homing--; //depends on motor direction
      delay(5);
  }
  ball_x_stepper.setCurrentPosition(0);
  //send ball to center
  ball_x_stepper.moveTo(x_center); //send ball x to center
  while (ball_x_stepper.distanceToGo() != 0){
      ball_x_stepper.run();
  }
  ball_y1_stepper.moveTo(y_center); //send call to y center
  ball_y2_stepper.moveTo(y_center);
  while (ball_y1_stepper.distanceToGo() != 0 && ball_y2_stepper.distanceToGo() != 0){
      ball_y1_stepper.run();
      ball_y2_stepper.run();
  }
  playSdWav1.play("INITIALIZATION.wav");
}

void setup(){
  //pic communication
  pinMode(ledPin, OUTPUT);
  pinMode(P1_goalPin, OUTPUT);
  pinMode(P2_goalPin, OUTPUT);
  pinMode(p1_hit, OUTPUT);
  pinMode(p2_hit, OUTPUT);
  pinMode(Blue_ON, OUTPUT);
  pinMode(Green_ON, OUTPUT);
  digitalWrite(ledPin, HIGH);
```

```
  AudioMemory(8); //set memory for sd card access

  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    // to calculate the note duration, take one second divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(35, melody[thisNote], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(8);
}

  digitalWrite(Blue_ON, HIGH);
  lcd.begin(16,2);
  lcd.print("Mechanical Pong");
  lcd.setCursor(0,1);
  lcd.print("Initializing...");
  digitalWrite(Blue_ON, LOW);

  //home steppers
  homing();
  lcd.print(">>>Completed<<<");
  digitalWrite(Green_ON, HIGH);
  delay(2000);
  digitalWrite(Green_ON, LOW);

  lcd.clear();

  //print player scores
  update_score();

  //Game speeds
  ball_y1_stepper.setMaxSpeed(5000);
  ball_y1_stepper.setAcceleration(46000);
  ball_y2_stepper.setMaxSpeed(5000);
  ball_y2_stepper.setAcceleration(46000);
  ball_x_stepper.setMaxSpeed(5000);
```

```
    ball_x_stepper.setAcceleration(46000);
    p1_stepper.setMaxSpeed(5000);
    p1_stepper.setAcceleration(16000);
    p2_stepper.setMaxSpeed(5000);
    p2_stepper.setAcceleration(16000);

}

void loop(){
    if (digitalRead(selectButton) == HIGH && digitalRead(nextButton) == HIGH){
        p1_score = 0;
        p2_score = 0;
        update_score();
        reset_ball();
    }
    //call ball()
    ball();
    //call p1_paddle()
    p1_paddle();
    //call p2_paddle()
    p2_paddle();
}
```

---END CODE---

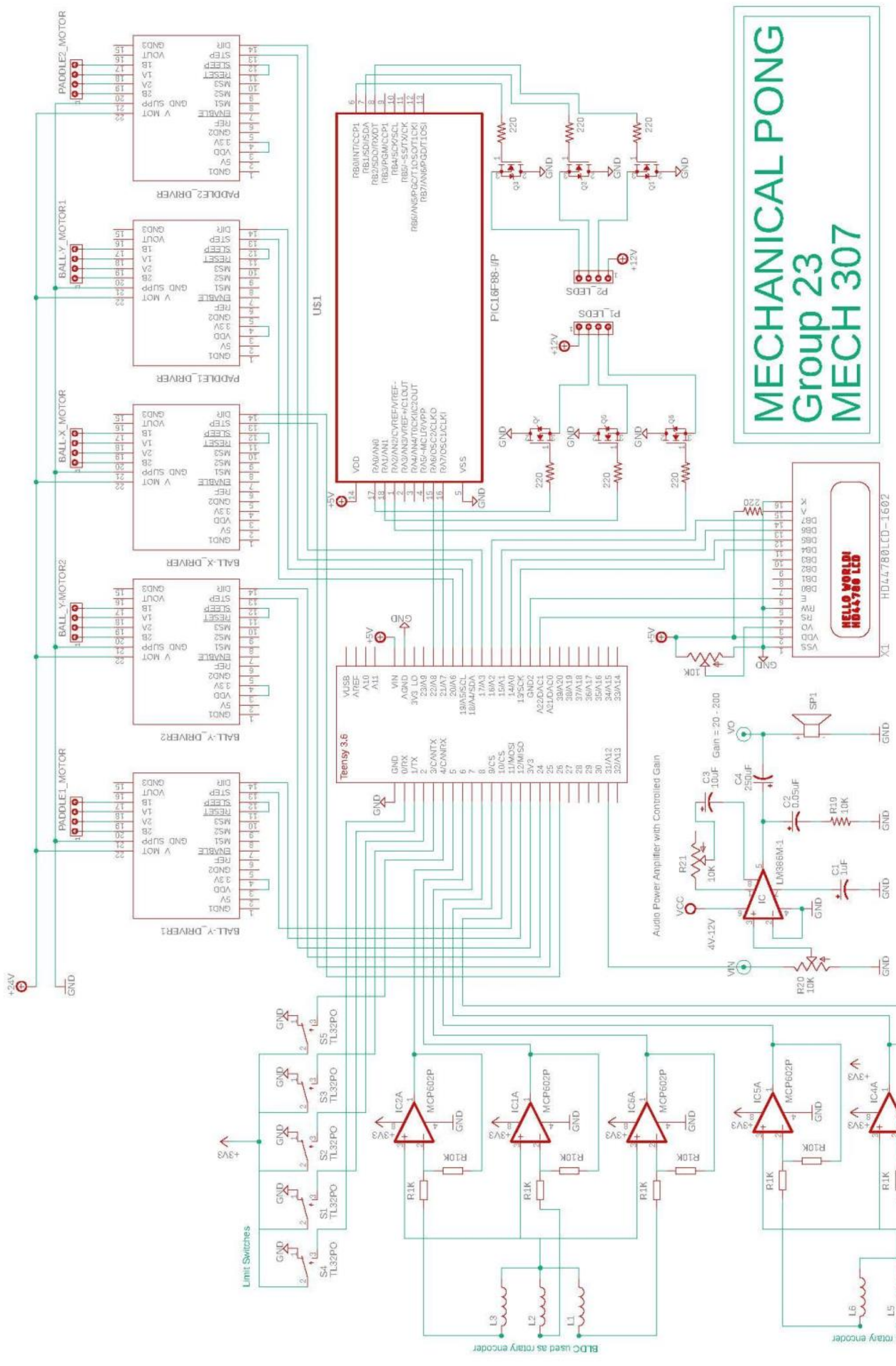
PIC16F88 LED Strip Control Code:

```
'****************************************************************
'*  Name    : pic_pong.BAS                                      *
'*  Author  : MECH 307 Group 23                                 *
'*  Notice  : Copyright (c) 2019 [select VIEW...EDITOR OPTIONS] *
'*          : All Rights Reserved                               *
'*  Date    : 5/8/2019                                          *
'*  Version : 1.0                                               *
'*  Notes   :                                                   *
'*          :                                                   *
'****************************************************************
'Identify and set the internal oscillator clock speed (required for the PIC16F88)
DEFINE OSC 8
OSCCON.4 = 1
OSCCON.5 = 1
OSCCON.6 = 1
```

```
' Turn off the A/D converter (required FOR the PIC16F88)
ANSEL = 0

P1_goal var PORTA.0
P2_goal var PORTA.1
P1_paddle var PORTA.2
P2_paddle var PORTA.3
Blue_ON var PORTA.4
Green_ON var PORTA.7


TRISA = %11111111 'set RA0-RA7 TO inputs

i var BYTE 'counter variable

P1_green var PORTB.0
P1_red var PORTB.1
P1_blue var PORTB.2

P2_green var PORTB.7
P2_red var PORTB.6
P2_blue var PORTB.5

TRISB = %00000000 'set RB0-RB7 TO outputs


main:
        IF P1_goal = 1 THEN
  GOTO P1
  ENDIF
  IF P2_goal = 1 THEN
  GOTO P2
  ENDIF
        IF P1_paddle = 1 THEN
        GOTO Paddle_1
        ENDIF
        IF P2_paddle = 1 THEN
        GOTO Paddle_2
        ENDIF
        WHILE Blue_ON = 1
        HIGH P1_blue
        HIGH P2_blue
        WEND
```

```
  WHILE Green_ON = 1
        HIGH P1_green
        HIGH P2_green
        WEND
  GOTO main
  END

P1:
  HIGH P2_red
  FOR i = 1 TO 3
     HIGH P1_blue
     PAUSE 200
     LOW P1_blue
     PAUSE 200
     NEXT i
  LOW P2_red
  GOTO main
  END

P2:
  HIGH P1_red
  FOR i = 1 TO 3
     HIGH P2_blue
     PAUSE 200
     LOW P2_blue
     PAUSE 200
     NEXT i
  LOW P1_red
  GOTO main
  END

Paddle_1:
        HIGH P1_green
        PAUSE 200
        LOW P1_green
        GOTO main
        END

Paddle_2:
        HIGH P2_green
        PAUSE 200
        LOW P2_green
        GOTO main
        END
```

Wiring Diagram:



List of Figures and Tables

Figure 1: The original pong game layout 2
Figure 2: The board shown with the top removed 3
Figure 3: The game board shown with the top attached 3
Figure 4: Rail Layout and General Overview 6
Figure 5: Teensy 3.6 Microcontroller 8
Figure 6: The ball assembly showing the sliding fit portion that locks the ball to the 20x20, the GT2 belt attachment points, the M3 nut slot, and the limit switch contact. 8
Figure 7: The paddle assembly showing the sliding spring mechanism, the belt attachment points, the v-channel bearings, and the modular design 9
Figure 8: Stepper Motor Mount and Assembly 10
Figure 9: GT2 belt profile 10
Figure 10: Wiring Diagram of Mechanical Pong System 11
Figure 11: Software Flow Chart 14
Figure 12: LCD screen, speakers, and game buttons 16
Figure 13: Manual User Input - BLDC Motor Used as Encoder 17
Figure 14: Limit Switch Mounted on Rail 18
Figure 15: Vexta stepper mounted in the system 18
Figure 16: Stepper Motor and Motor Driver Circuit 19
Figure 17: PIC16F88 LED Strip Circuit 19
Figure 18: Printing Tolerance 21

Table 1: System Components 4
Table 2: Partial Parts List 20