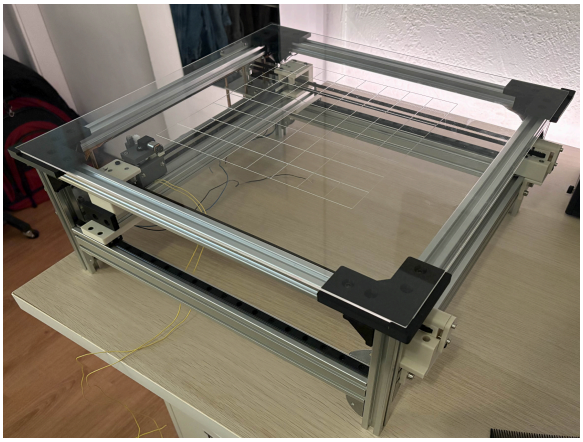
ME 102B Final Report - IntelliChess

Jason Kyle Lopez, Brian Lu, Nelson Sim, David Soto Gonzalez

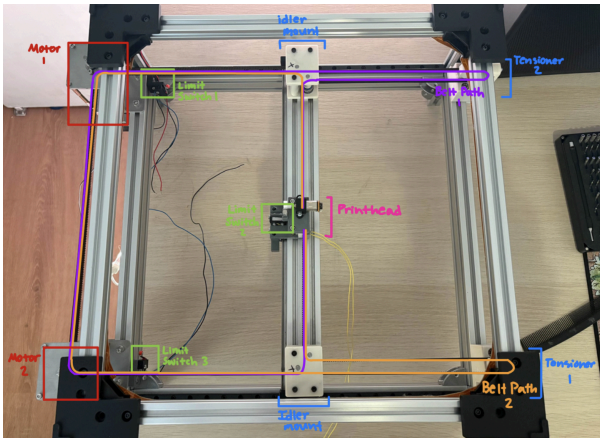
For our final project, we focused on making an automatic chessboard that moves chess pieces quickly and efficiently based on the user's input. We decided to call our system IntelliChess, as it resembles an intelligent chessboard that can move on its own. It can be played with another person and has a clear acrylic board as the chessboard to see the mechanisms inside working. The idea was pursued as the group was interested in chess and wanted to learn how to design a mechanical gantry system, which worked well with this product. We initially aimed to include a computer opponent option if the user wanted to play against the board or have it be an educational tool to learn how to play chess. We had to simplify our goal to ensure that the mechanism and software to move/capture the pieces worked flawlessly.

Our high-level strategy was a 2-axis gantry system that could move a magnet to the desired pieces and positions underneath the acrylic chessboard. We 3D printed the chess pieces at a smaller scale to avoid having the chess pieces bundle up due to each piece having a magnet. Additionally, we included limit switches to zero the 2-axis gantry system after each command and attached LEDs to a potentiometer to light up the board. We initially wanted an electromagnet as the base magnet. The idea was that the electromagnet would turn on when it contacts the acrylic board to move the individual chess pieces and then turn off when the chess piece has reached its new position. However, we changed the idea to having a DC motor move a regular magnet up and down. It will go up when arriving at the desired chess piece and down when it moves positions. At the beginning of the product design, we wanted to include an AI computer opponent so that the user could play against the chessboard. Despite not achieving this idea at present, we are still able to have two-player games and can include the concept at a later time. Additionally, we aimed to have the gantry system move 1500 steps per second, which is slow, and we thought it would be a perfect speed to move the chess pieces efficiently. In the end, the gantry system currently moves 3750 steps per second, more than double what we aimed for and it still moves the chess pieces very efficiently and smoothly.

IntelliChess's frame is mostly made from several 20x20mm aluminum extrusions of different lengths. This main rigid structure is held together by small L brackets, hex nuts, and screws.

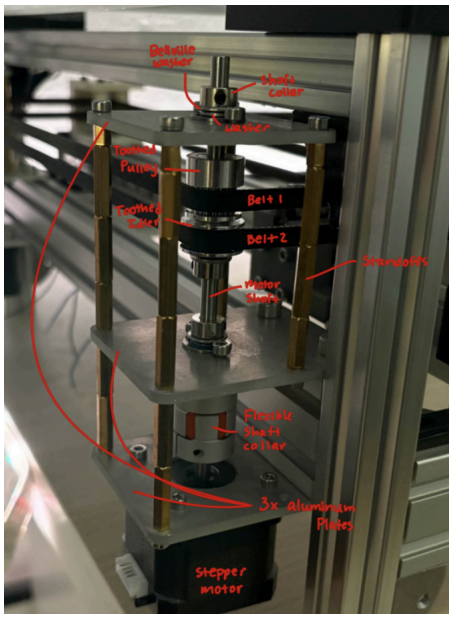


IntelliChess Frame

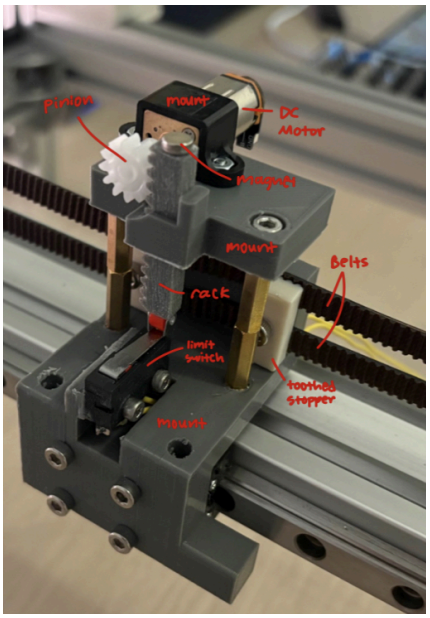


Labeled Components

The gantry moves in the x direction along two rails and rail blocks on both ends of our frame and has another rail and rail block attached to it that moves in the y direction. Our motor housings each include 1/8in aluminum plates, each water jet to their desired dimensions to fit our bearings and screws for mounting. For each motor, there are three aluminum plates spaced out by standoffs of different lengths. We needed to have correct sizing for our motor shaft and standoffs to ensure that our toothed idlers and pulleys for our belts were parallel to our tensioners and idlers on our gantry. Other important components of our motor housing include a flexible shaft collar, shaft collars, bearings, and washers.

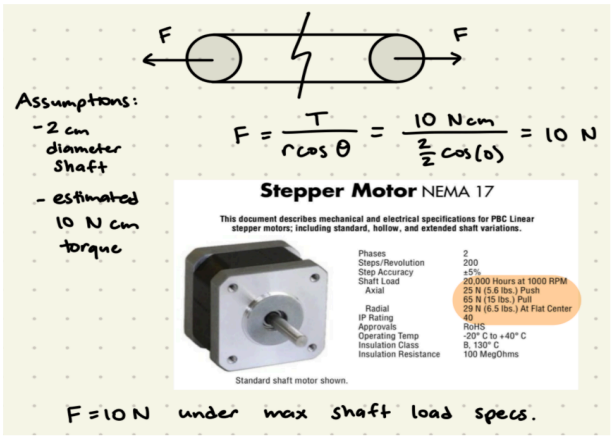


Motor Housing with Labels



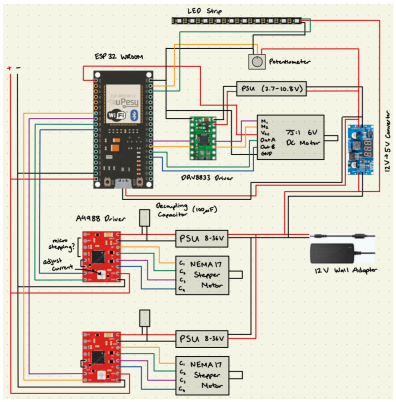
Printhead with Labels

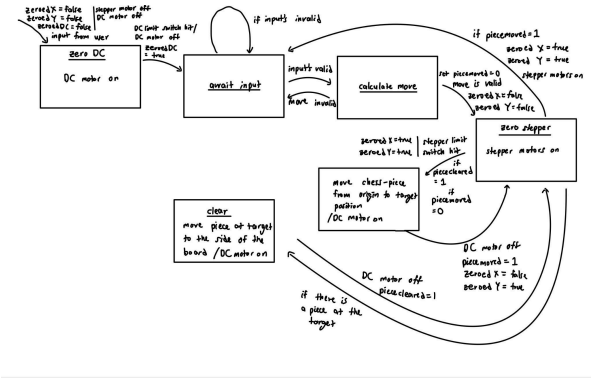
The tensioners were heavily modified from the Voron 3D Printer tensioners to fit and mount to our model's frame. This is important so that we can modulate the tension of our belts to make sure they are not too loose/tight and that they have good contact with our idlers and pulleys. Here are our calculations for the forces that would be acting on our stepper motor shafts:



Stepper Motor Force Diagram

Our tensioners and idlers on our gantry guide our belts with two F695 bearings that are stacked on top of each other. Our belts are held together at the “print head” of our gantry via two 3D-printed toothed stoppers where they are clamped down. Our print head is responsible for moving the main magnet that will move our chess pieces. It also houses our DC motor which controls a pinion that moves our rack up and down. We moved our DC motor mechanism as close to the acrylic board as possible via standoffs. We have a total of three limit switches, two that are used to stop our gantry and printhead from moving in the x-y directions, and the last one is attached to our printhead which is used to reset our rack to a starting position. The last components of our project just include peripherals such as our mounts for our acrylic board and tensioners and of course our chess pieces.





Circuit Diagram State Diagram

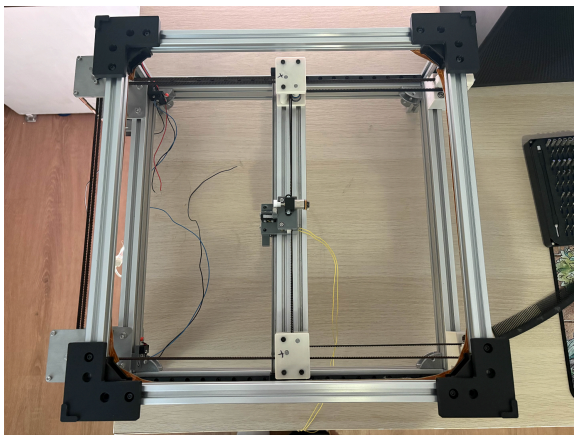
A strategy that we would recommend to future students that worked very well for us is to complete the assigned work, whether for a class assignment or the group project, 2 days before the deadline. That way, there can be enough time for revision, ask for help, and ensure that everything is completed on time. What we would have done differently on this project is order the parts and assemble the mechanical components earlier so that we could have spent more time on the code to include the computer feature and fix up the wiring to make it look neat.

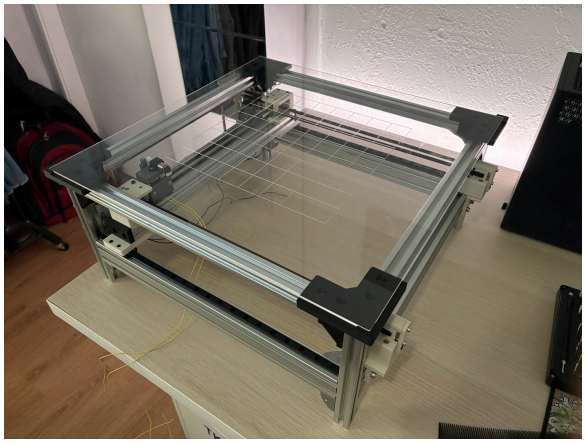
Appendix

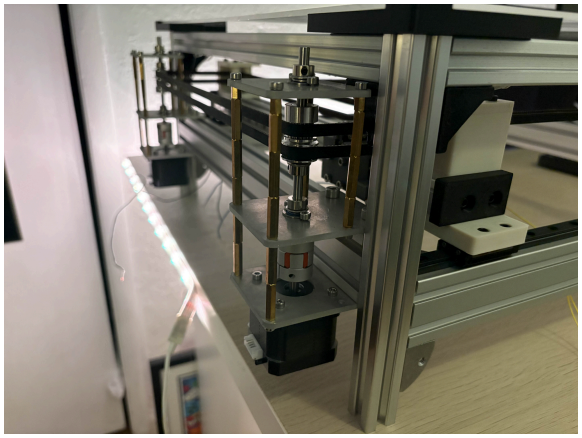
Bill of Materials:

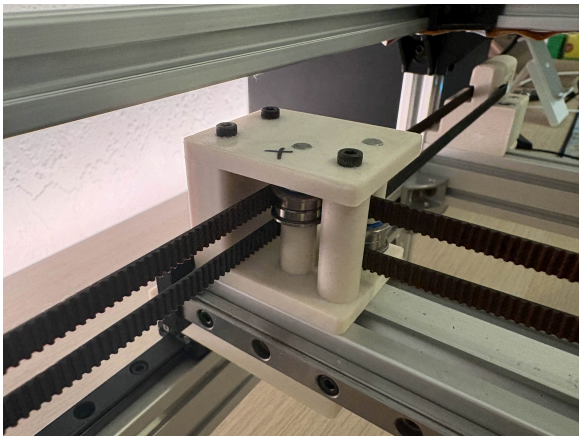
							Total Price for Project: $274.07
Quantity	Price/unit ($)	Vendor	Purchase URL	Description	Multiple	Total Qty	Total Price/part ($)
5	$0.00	Teammate	NA	20x40 400mm Aluminum Extrusion	1	5	$0.00
1	$15.19	Amazon	https://www.amazon.com/Aluminum-Extrusion-Europea	20x20 200mm Aluminum Extrusion - (4-Pack)	1	1	$15.19
12	$0.00	Teammate	NA	20x20 L Brackets	1	12	$0.00
3	$0.00	Teammate	NA	MGN9H Rails (400mm)	1	3	$0.00
3	$0.00	Teammate	NA	MGN9H Blocks	1	3	$0.00
1	$6.97	Amazon	https://www.amazon.com/Fastener-Nickel-Plated-Sliding	M5 L Bracket Nuts	1	1	$6.97
1	$6.99	Amazon	https://www.amazon.com/HELIFOUNER-Pieces-2020-Al	M3 L Bracket Nuts	1	1	$6.99
1	$0.00	Teammate	NA	DC Gear Motor with Encoder	1	1	$0.00
2	$9.99	Amazon	https://www.amazon.com/STEPPERONLINE-Stepper-Bip	NEMA17 Steppers (38mm Base)	1	2	$19.98
1	$7.99	Amazon	https://www.amazon.com/WWZMDiB-Stepstick-Stepper		1	1	$7.99
1	$13.98	Amazon	https://www.amazon.com/Csdtyih-Male-Female-Standof	Brass Stand-offs (F-F 20mm)	1	1	$13.98
		Amazon	Included in kit above	Brass Stand-offs (M-F 20mm)	1	0	$0.00
		Amazon	Included in kit above	Brass Stand-offs (M-F 15mm)	1	0	$0.00
2	$0.00	Teammate	NA	2GT 20T Toothed Pulley	1	2	$0.00
2	$0.00	Teammate	NA	2GT 20T Toothed Idler	1	2	$0.00
4	$0.00	Teammate	NA	M5 Shaft Collar	1	4	$0.00
2	$0.00	Teammate	NA	M5 Flexible Shaft Collar	1	2	$0.00
1	$7.29	Amazon	https://www.amazon.com/uxcell-695-2RS-Bearing-5x13	F695-2RS Bearing (5x13x4mm) - (5-Pack)	1	1	$7.29
20	$0.00	Teammate	NA	M5 Washers	1	20	$0.00
1	$21.99	Amazon	https://www.amazon.com/Assortment-M2-M3-M4-M5/	M3 Screws 10mm	1	1	0
8			Included in assortment above	M5 Screws 10mm	1	8	$0.00
1	$6.79	Amazon	https://www.amazon.com/uxcell-100mm-Stainless-Steel	M5 Shaft 100mm - (Set of 5)	1	1	$6.79
	$0.00	Teammate	NA	M3 Washers	1	0	$0.00
2	$4.33	Misumi	https://us.misumi-ec.com/vona2/detail/110300086920/	M5 18mm Pin (Shaft)	1	2	$8.66
1	$5.74	McMaster	https://www.mcmaster.com/products/screws/socket-he	M5 Screws 30mm - (10-Pack)	1	1	0
4	$0.00	Teammate	NA	M3 Heat Set Inserts	1	4	$0.00
1	$7.17	McMaster	https://www.mcmaster.com/products/screws/socket-he	M3 Screws 30mm - (10-Pack)	1	1	0
4	$4.33	Misumi	https://us.misumi-ec.com/vona2/detail/110300086920/	M5 40mm Pin (Shaft)	1	4	$17.32
6	$4.57	McMaster	https://www.mcmaster.com/products/screws/system-of	M3 Screws 45mm	1	6	0
2	$0.00	Teammate	NA	6mm width Belts (Might need more of)	1	2	$0.00
1	$14.99	Amazon	https://www.amazon.com/Aluminum-Protective-Treatab	1/8" Aluminum Plate - (2-Pack)	1	1	$14.99
2	$5.21	Misumi	https://us.misumi-ec.com/vona2/detail/110300086920/	100mm Rotary Shaft 5m Diameter	1	2	$10.42
1	$9.99	Amazon	https://www.amazon.com/Belleville-Conical-Stainless-Co	M5 Bellville Washers	1	1	$9.99
1	$5.72	McMaster	https://www.mcmaster.com/products/racks-and-pinions	Pinion	1	1	$5.72
1	$3.80	McMaster	https://www.mcmaster.com/products/racks-and-pinions	Rack	1	1	$3.80
1	$2.43	K&J Magnetic	https://www.kjmagnetics.com/proddetail.asp?prod=D68	Permanent magnet	1	1	$2.43
1	$8.99	Amazon	https://www.amazon.com/Magnets-Refrigerator-Neodyi	Smaller magnets (for the base of the smallest c	1	1	$8.99
1	$0.00	Teammate	NA	Roll of LED strips	1	1	$0.00
1	$11.99	Amazon	https://www.amazon.com/ALITOVE-Adapter-Converter-1	12V Power Supply	1	1	$11.99
1	$6.99	Amazon	https://www.amazon.com/ElectroCookie-Solderable-Bre	Breadboard PCB	1	1	$6.99
1	$5.99	Amazon	https://www.amazon.com/HiLetgo-KW12-3-Roller-Switc	Limit switches - (10-Pack)	1	1	$5.99
1	$35.98	Amazon	https://www.amazon.com/Lesnlok-Pieces16x16-Acrylic-I	Acrylic - (2-Pack)	1	1	$35.98
1	$7.99	Amazon	https://www.amazon.com/JTAREA-LM2596-Converter-Re	Step down converter - (2-Pack)	1	1	$7.99
1	~	Bolt Depot	https://boltdepot.com/Catalog	Socket Head Screws (ALL NEEDED FOR PROJECT	1	1	37.63

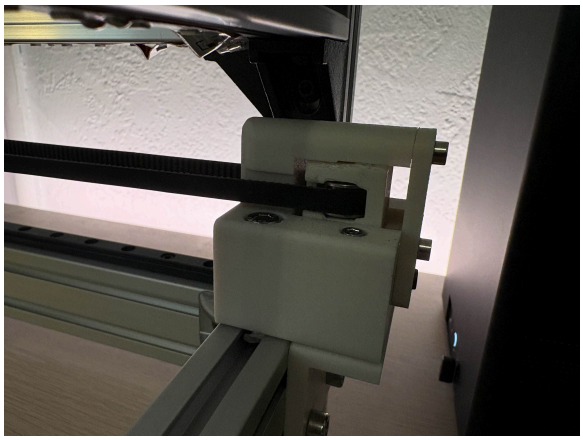
Mechanical Design / CAD

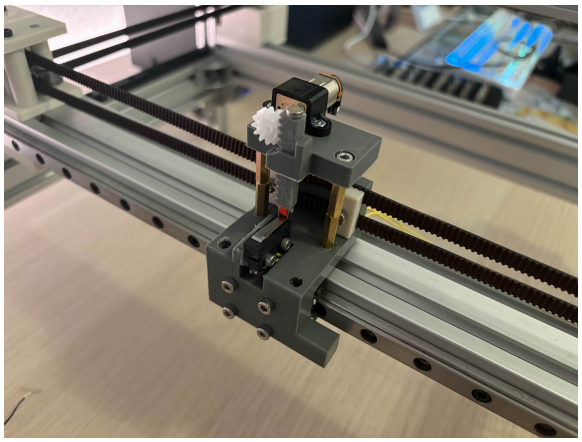


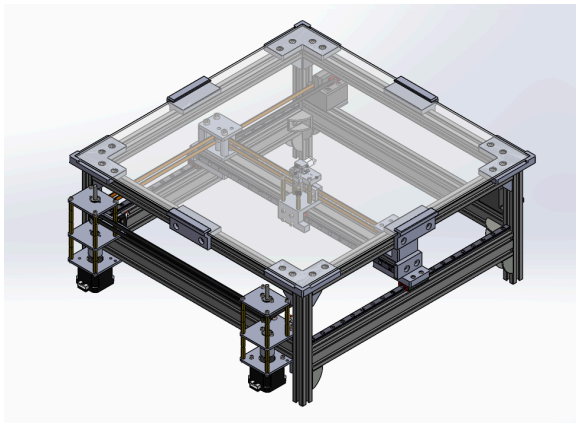


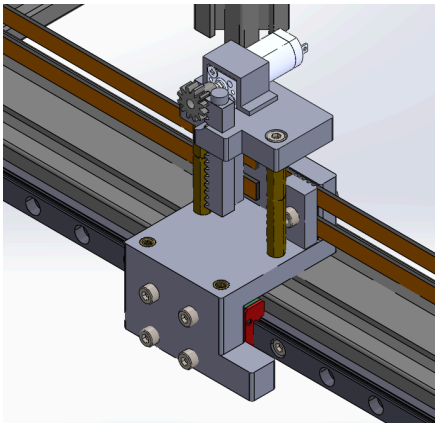


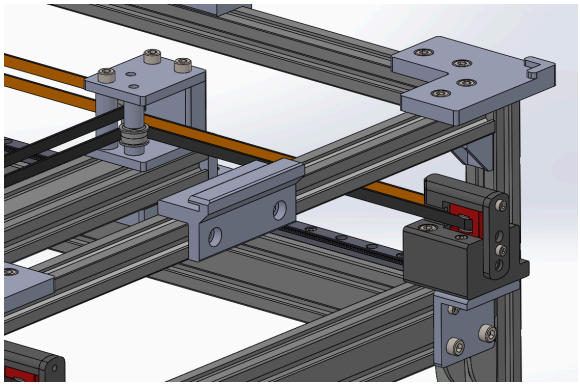


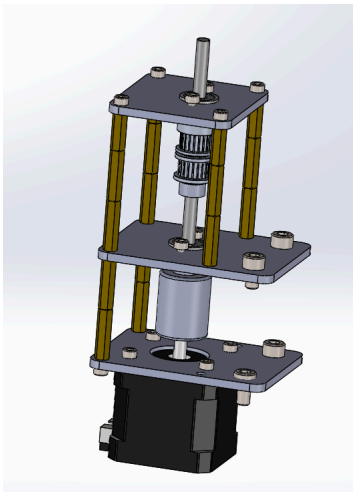












Code:

```
#include <AccelStepper.h>
#include <ezButton.h>
#include <ESP32Encoder.h>
#include <Arduino.h>

struct deltaValues {
  float deltaA;
  float deltaB;
};

#define IN1 32
#define IN2 14
#define DC_ENC1 26
#define DC_ENC2 25

#define STEP_PIN1 16
#define DIR_PIN1 19
#define STEP_PIN2 21
#define DIR_PIN2 17

#define LIM_SWITCH_X 33
#define LIM_SWITCH_Y 27
#define LIM_SWITCH_DC 15

const int STEP_MAX_SPEED_TYP = 3750;
const int STEP_ACCEL_TYP = 20000000;
```

```
const int STEP_MAX_SPEED_ZERO = 2750;
const int STEP_ACCEL_ZERO = 20000000;
const int SCALE = 1232.86; //ADJUST AS NECESSARY (ROTATIONAL TO LINEAR)
const float OFFSET = 0.5; // ADJUST AS NECESSARY (SQUARE LENGTH/2)

AccelStepper stepper1(AccelStepper::DRIVER, STEP_PIN1, DIR_PIN1);
AccelStepper stepper2(AccelStepper::DRIVER, STEP_PIN2, DIR_PIN2);

//PWM properties
const int freq = 5000;
const int resolution = 8;
int MAX_PWM_VOLTAGE = 120;
volatile bool destinationReached = false;

int targetX1, targetY1, targetX2, targetY2;
float pathMove[5][2];
float pathClear[5][2];
int currentStep;
int mode = 0;
int val = 0;

String layout[8][8];
int blackCaptured = 0;
int whiteCaptured = 0;
float capturedX, capturedY;

ezButton limitSwitchX(LIM_SWITCH_X);
ezButton limitSwitchY(LIM_SWITCH_Y);
ezButton limitSwitchDC(LIM_SWITCH_DC);

bool zeroedX = false;
bool zeroedY = false;
bool zeroedDC = false;
bool arrived = true;
bool validMove = true;

void setup() {
  Serial.begin(115200);

  // configure PWM functionalities
```

```
  ledcAttach(IN1, freq, resolution);
  ledcAttach(IN2, freq, resolution);

  // Initalize DC motor
  pinMode(DC_ENC1, INPUT);
  pinMode(DC_ENC2, INPUT);

  // Initialize stepper motor parameters (3200 CTS/REV)
  stepper1.setMaxSpeed(STEP_MAX_SPEED_ZERO);
  stepper1.setAcceleration(STEP_ACCEL_ZERO);
  stepper2.setMaxSpeed(STEP_MAX_SPEED_ZERO);
  stepper2.setAcceleration(STEP_ACCEL_ZERO);

  // Initialize debounce time for limit switches
  limitSwitchX.setDebounceTime(50);
  limitSwitchY.setDebounceTime(50);
  limitSwitchDC.setDebounceTime(50);

  // Interrupts for limit switch zeroing
                      attachInterrupt(digitalPinToInterrupt(LIM_SWITCH_X),
ISR_horizontalLimit, FALLING);
   attachInterrupt(digitalPinToInterrupt(LIM_SWITCH_Y), ISR_verticalLimit,
FALLING);
      attachInterrupt(digitalPinToInterrupt(LIM_SWITCH_DC),   ISR_DCLimit,
FALLING);

  // Initlize board layout
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      if (j < 2) {
        layout[i][j] = "[W]";
      } else if (j > 5) {
        layout[i][j] = "[B]";
      } else {
        layout[i][j] = "[ ]";
      }
    }
  }
}
```

```
void loop() {
  switch (mode) {
    case 0: // Zero DC
      if (Serial.available() > 0) {
        Serial.read();
        zeroDC();
        mode = 1;
      }
      break;
    case 1: // Received input
      Serial.print(zeroedDC);
      if (Serial.available() > 0) {
        readInput();
        if (!validMove) {
          break;
        }
        stepper1.move(3200);
        while (stepper1.distanceToGo() != 0) {
          stepper1.run();
        }
        stepper1.stop();

        zeroedX = false;
        zeroedY = false;
        mode = 2;

        Serial.println();Serial.println();
        Serial.println("STARTING LAYOUT");
        for (int i = 0; i < 8; i++) {
          for (int j = 0; j < 8; j++) {
            Serial.print(layout[i][j]);
            Serial.print("");
          }
          Serial.println();
        }
        Serial.println();Serial.println();
      }

      break;
    case 2: // Calculating moves
```

```
      calculateMove(targetX1, targetY1, targetX2, targetY2);
      if (layout[targetX2][targetY2] != "[ ]") {

Serial.println();Serial.println();Serial.println("CLEARING");Serial.println();Serial.println();
        currentStep = -5;
        if (layout[targetX2][targetY2] == "[W]") {
          capture("White");
        } else {
          capture("Black");
        }
        calculateClear(targetX2, targetY2, capturedX, capturedY);
      }
      layout[targetX2][targetY2] = layout[targetX1][targetY1];
      layout[targetX1][targetY1] = "[ ]";
      mode = 3;
      break;
    case 3: // Zeroing
      zero();
      if (currentStep < 0) {
        mode = 4;
      } else if (currentStep < 5) {
        mode = 5;
      } else {
        mode = 6;
      }
      break;
    case 4: // Clearing current piece
      while (currentStep < 0) {
        moveClear();
        if (currentStep == -4) {
          ascendState();
        } else if (currentStep == 0) {
          descendState();
        }
        delay(500);
      }
      mode = 3;
      break;
    case 5: // Move piece to target
```

```
      while (currentStep < 5) {
        movePiece();
        if (currentStep == 1) {
          ascendState();
        } else if (currentStep == 5) {
          descendState();
        }
        delay(500);
      }
      mode = 3;
      break;
    case 6: // Reset and return home
      arrived = true;
      currentStep = 0;

      Serial.println();Serial.println();
      Serial.println("ENDING LAYOUT");
      for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
          Serial.print(layout[i][j]);
          Serial.print("");
        }
        Serial.println();
      }
      Serial.println();Serial.println();

      mode = 1;
      break;
  }
}

void moveClear() {
  // Resets speed/accel to normal
  stepper1.setMaxSpeed(STEP_MAX_SPEED_TYP);
  stepper1.setAcceleration(STEP_ACCEL_TYP);
  stepper2.setMaxSpeed(STEP_MAX_SPEED_TYP);
  stepper2.setAcceleration(STEP_ACCEL_TYP);

  int deltaA, deltaB;
  Serial.print("Clearing, step ");
```

```
  Serial.println(currentStep);
  deltaA = pathClear[currentStep + 5][0];
  deltaB = pathClear[currentStep + 5][1];

  stepper1.move(deltaA);
  stepper2.move(deltaB);
  zeroedX = false;
  zeroedY = false;

  // Move the steppers to the target
  while (stepper1.distanceToGo() != 0 || stepper2.distanceToGo() != 0) {
    stepper1.run();
    stepper2.run();
  }

  // Print once arrived at each step
  Serial.print("Arrived at step ");
  Serial.print(currentStep);
  Serial.print(": Stepper Position = (");
  Serial.print(stepper1.currentPosition());
  Serial.print(", ");
  Serial.print(stepper2.currentPosition());
  Serial.println(")");

  // Move to the next step
  currentStep++;
}

void movePiece() {
  // Resets speed/accel to normal
  stepper1.setMaxSpeed(STEP_MAX_SPEED_TYP);
  stepper1.setAcceleration(STEP_ACCEL_TYP);
  stepper2.setMaxSpeed(STEP_MAX_SPEED_TYP);
  stepper2.setAcceleration(STEP_ACCEL_TYP);

  int deltaA, deltaB;
  Serial.print("Moving, step ");
  Serial.println(currentStep);
  // Set motor targets based on the current step's delta values
  deltaA = pathMove[currentStep][0];
```

```
  deltaB = pathMove[currentStep][1];

  stepper1.move(deltaA);
  stepper2.move(deltaB);
  zeroedX = false;
  zeroedY = false;

  // Move the steppers to the target
  while (stepper1.distanceToGo() != 0 || stepper2.distanceToGo() != 0) {
    stepper1.run();
    stepper2.run();
  }

  // Print once arrived at each step
  Serial.print("Arrived at step ");
  Serial.print(currentStep);
  Serial.print(", Deltas = (");
  Serial.print(deltaA);
  Serial.print(", ");
  Serial.print(deltaB);
  Serial.println(")");
  Serial.print(", Stepper Position = (");
  Serial.print(stepper1.currentPosition());
  Serial.print(", ");
  Serial.print(stepper2.currentPosition());
  Serial.println(")");

  // Move to the next step
  currentStep++;
}

void zero() {
  // Decrease speed
  stepper1.setMaxSpeed(STEP_MAX_SPEED_ZERO);
  stepper1.setAcceleration(STEP_ACCEL_ZERO);
  stepper2.setMaxSpeed(STEP_MAX_SPEED_ZERO);
  stepper2.setAcceleration(STEP_ACCEL_ZERO);

  Serial.println();Serial.println();
  Serial.println("Zeroing X Axis...");
```

```
  stepper1.move(-1000000);
  stepper2.move(1000000);
  while (!zeroedX) {
    stepper1.run();
    stepper2.run();
  }
  stopMotors();
  zeroedX = true;

  Serial.println("X Axis Zeroed. Zeroing Y Axis...");
  stepper1.move(-1000000);
  stepper2.move(-1000000);
  while (!zeroedY) {  // Event checker for limit switch
    stepper1.run();
    stepper2.run();
  }
  stopMotors();
  zeroedY = true;

  // Both axes are zeroed; reset positions and wait for new input
  stepper1.setCurrentPosition(0);
  stepper2.setCurrentPosition(0);
  Serial.println("Zeroing complete. Current position reset to origin.");
  Serial.println();Serial.println();
}

void zeroDC() {
  ledcWrite(IN1, LOW);
  ledcWrite(IN2, MAX_PWM_VOLTAGE);
  delay(100);
  descendState();
}

void ascendState() {
    Serial.println("Transitioned to ascend state. Motor running...");
    ledcWrite(IN1, LOW);
    ledcWrite(IN2, MAX_PWM_VOLTAGE);
    delay(180);
    ledcWrite(IN1, 255);
    ledcWrite(IN2, 255);
```

```
}

void descendState() {
    Serial.println("Transitioned to descend state. Motor running...");
    zeroedDC = false;
    while (!zeroedDC) {
      Serial.print(zeroedDC);
      ledcWrite(IN1, MAX_PWM_VOLTAGE);
      ledcWrite(IN2, LOW);
    }
    ledcWrite(IN1, 255);
    ledcWrite(IN2, 255);
}

// ISR for horizontal limit switch
void ISR_horizontalLimit() {
  zeroedX = true;  // Set flag to indicate horizontal zeroing is complete
}

// ISR for vertical limit switch
void ISR_verticalLimit() {
  zeroedY = true;  // Set flag to indicate vertical zeroing is complete
}

// ISR for DC motor limit switch
void ISR_DCLimit() {
  zeroedDC = true;  // Set flag to indicate DC zeroing is complete
}

// Stop motors after zeroing
void stopMotors() {
  stepper1.stop();
  stepper2.stop();
}

// Reads serial input from user in the form (X1, Y1, X2, Y2) FORMAT IMPORTANT
void readInput() {
   String input = Serial.readStringUntil('\n');  // Read the input until newline
```

```
  // Find the comma separators
  int firstComma = input.indexOf(',');
  int secondComma = input.indexOf(',', firstComma + 1);
  int thirdComma = input.indexOf(',', secondComma + 1);

  if (firstComma != -1 && secondComma != -1 && thirdComma != -1) {
    // Parse the first X coordinate (as a letter)
    char x1Char = input.charAt(0);
    targetX1 = toupper(x1Char) - 'A';  // Convert 'A'-'H' to 1-8

    // Parse the first Y coordinate (as a number)
    targetY1 = input.substring(firstComma + 2, secondComma).toInt() - 1;

    // Parse the second X coordinate (as a letter)
    char x2Char = input.charAt(secondComma + 2);
    targetX2 = toupper(x2Char) - 'A';  // Convert 'A'-'H' to 1-8

    // Parse the second Y coordinate (as a number)
    targetY2 = input.substring(thirdComma + 2).toInt() - 1;

    // Print the parsed coordinates
    Serial.print("Entered coordinates: (");
    Serial.print(targetX1);
    Serial.print(", ");
    Serial.print(targetY1);
    Serial.print(") to (");
    Serial.print(targetX2);
    Serial.print(", ");
    Serial.print(targetY2);
    Serial.println(")");
    validMove = true;

    if (layout[targetX1][targetY1] == "[ ]") {
      Serial.println("Error, There is no piece at the starting square");
      validMove = false;
    }
     if (targetX1 > 7 || targetY1 > 7 || targetX2 > 7 || targetY2 > 7 ||
targetX1 < 0 || targetY1 < 0 || targetX2 < 0 || targetY2 < 0) {
      Serial.println("Error, Input is out of bounds");
```

```
      validMove = false;
    }
  } else {
    Serial.println("Error, Please enter the coordinates in the form: A1, B1, C2, D2");
    validMove = false;
  }
}

/* Consolidates matrix of dA and dB paths using calculateSteps()
    OFFSET = 0 means it travels along the edges
    Adding OFFSET means it moves to the center
    Calculates path to target square
 */
void calculateMove(float startingEdgeX, float startingEdgeY, float targetEdgeX, float targetEdgeY) {
  Serial.println("MOVE STEPS");
  // Step 1: Move to center of the piece
  deltaValues step1 = calculateSteps(startingEdgeX + OFFSET, startingEdgeY + OFFSET, 0, 0);
  pathMove[0][0] = step1.deltaA + 2000; // + 18000; // + 27466.67 Extra constant to account for grid offset
  pathMove[0][1] = step1.deltaB - 1900; // + 8666.667;

  // Step 2: Move to nearest corner of the piece
    deltaValues step2 = calculateSteps(startingEdgeX, startingEdgeY, startingEdgeX + OFFSET, startingEdgeY + OFFSET);
  pathMove[1][0] = step2.deltaA;
  pathMove[1][1] = step2.deltaB;

  // Step 3: Move along edges to the desired corner
    deltaValues step3 = calculateSteps(targetEdgeX, startingEdgeY, startingEdgeX, startingEdgeY);
  pathMove[2][0] = step3.deltaA;
  pathMove[2][1] = step3.deltaB;

  // Step 4: Move to target corner
    deltaValues step4 = calculateSteps(targetEdgeX, targetEdgeY, targetEdgeX, startingEdgeY);
  pathMove[3][0] = step4.deltaA;
```

```
  pathMove[3][1] = step4.deltaB;

  // Step 5: Move to center of the target position
   deltaValues step5 = calculateSteps(targetEdgeX + OFFSET, targetEdgeY +
OFFSET, targetEdgeX, targetEdgeY);
  pathMove[4][0] = step5.deltaA;
  pathMove[4][1] = step5.deltaB;

  for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 2; j++) {
          Serial.print(pathMove[i][j]);
          Serial.print(" ");
        }
        Serial.println();
      }
}

/* Consolidates matrix of dA and dB paths using calculateSteps()
    OFFSET = 0 means it travels along the edges
    Adding OFFSET means it moves to the center
    Calculates path to "graveyard"
 */
void  calculateClear(float  startingEdgeX,  float  startingEdgeY,  float
targetEdgeX, float targetEdgeY) {
  Serial.println("CLEAR STEPS");
  // Step 1: Move to center of the piece
  deltaValues step1 = calculateSteps(startingEdgeX + OFFSET, startingEdgeY
+ OFFSET, 0, 0);
   pathClear[0][0] = step1.deltaA + 2000; // + 18000; // + 27466.67 Extra
constant to account for grid offset
  pathClear[0][1] = step1.deltaB - 1900; // + 8666.667;

  // Step 2: Move to nearest corner of the piece
     deltaValues  step2  =  calculateSteps(startingEdgeX,  startingEdgeY,
startingEdgeX + OFFSET, startingEdgeY + OFFSET);
  pathClear[1][0] = step2.deltaA;
  pathClear[1][1] = step2.deltaB;

  // Step 3: Move along edges to the desired corner
```

```
    deltaValues step3 = calculateSteps(startingEdgeX, 0, startingEdgeX,
startingEdgeY);
  pathClear[2][0] = step3.deltaA;
  pathClear[2][1] = step3.deltaB;

  // Step 4: Move to targetX, 0
  deltaValues step4 = calculateSteps(targetEdgeX, 0, startingEdgeX, 0);
  pathClear[3][0] = step4.deltaA;
  pathClear[3][1] = step4.deltaB;

  // Step 5: Move to center of the target position
      deltaValues   step5   =   calculateSteps(targetEdgeX,   targetEdgeY,
targetEdgeX, 0);
  pathClear[4][0] = step5.deltaA;
  pathClear[4][1] = step5.deltaB;
}

/* Calculates belt deltas given target X, Y coordinates in mm (ASSUMES 1
REV = 10 MM DELTA)
    Solves for deltaA and deltaB using Crammer's rule
 */
deltaValues calculateSteps(float targetX, float targetY, float originX,
float originY) {
  deltaValues results;
  results.deltaA = SCALE * ((targetX - originX) + (targetY - originY));
  results.deltaB = SCALE * ((targetY - originY) - (targetX - originX));

  // Print the deltaA and deltaB
  Serial.print("dA: ");
  Serial.print(results.deltaA);
  Serial.print(", dB: ");
  Serial.println(results.deltaB);

  // Print the current position
  Serial.print("Origin: (");
  Serial.print(originX);
  Serial.print(", ");
  Serial.print(originY);
  Serial.println(")");
```

```
  // Print the target position
  Serial.print("Target: (");
  Serial.print(targetX);
  Serial.print(", ");
  Serial.print(targetY);
  Serial.println(")");

  return results;
}

// Moves captured pieces to side
void capture(String color) {
  if (color == "Black") {
    capturedX = -0.7;
    capturedY = 8 - 0.53333*blackCaptured;
    blackCaptured++;
  } else {
    capturedX = -1.4;
    capturedY = 8 - 0.53333*whiteCaptured;
    whiteCaptured++;
  }
}
```