DrawPrint: Design of a Dry Erase Plotting Device

Billy Koech
Electrical Engineering
Harvard School of Engineering and Applied Sciences
Cambridge, MA United States of America
bkoech@college.harvard.edu

***Abstract*— Pen plotters have many potential applications in design, visualization and education as teaching aids to name a few. While numerous pen plotters designs exist online few are designed for integration with dry erase boards. DrawPrint is a pen plotter that takes vectorized images and prints them out on a dry erase board attached to it. This report explores the design and specifications of DrawPrint.**

Keywords—Gcode, two axis motion, microcontroller, linear actuator

I. Introduction

Drawprint is designed to print out images onto a dry erase board. It consists of a two-axis motion system attached onto a dry erase board using aluminum 80 20 accessories. The two-axis motion system is actuated by a set of three stepper motors. The stepper motors translate the torque via a GT2 timing belt. The stepper motors are controlled by an Arduino Mega 2560 microcontroller. High level functionality such as image parsing, vectorization, translation and resizing is written in a high-level language(python) and can be executed on any microcontroller or computer with a python installation. Communication between the low-level microcontroller(Arduino) and the high-level microcontroller(Raspberry Pi) is done through USB serial.

This report follows the following structure. Section II presents the mechanical design of the device and the technical specifications of the stepper motors. Section III discusses the circuit design and technical specifications for interfacing between the stepper motors and the low-level microcontroller. It then transitions to Section IV for a walkthrough of the functions and Finite State Machine(FSM) written to control the stepper motors. This is then followed by a walkthrough in Section V of the high-level modules written in python for parsing vectorized image data and converting them to instruction for the device. Finally, Section VI discusses the challenges and potential areas of improvement.

II. Mechanical Design

A. Two Axis motion system.

The entire two axis motion system is built on an aluminum 80 20 frame. The dimensions of the frame were measured to match that of a dry erase board of size 816 by 1096 Appendix I consist of a drawing package that documents the dimensions of each component.

X motion is achieved by two sets of three wheels that ride on aluminum 80 20 bars as shown in Figure 1. One set sits at the top and the other at the bottom. The three wheels are held in place by a custom designed acrylic plate in the shape of a triangle also shown in Figure 1.

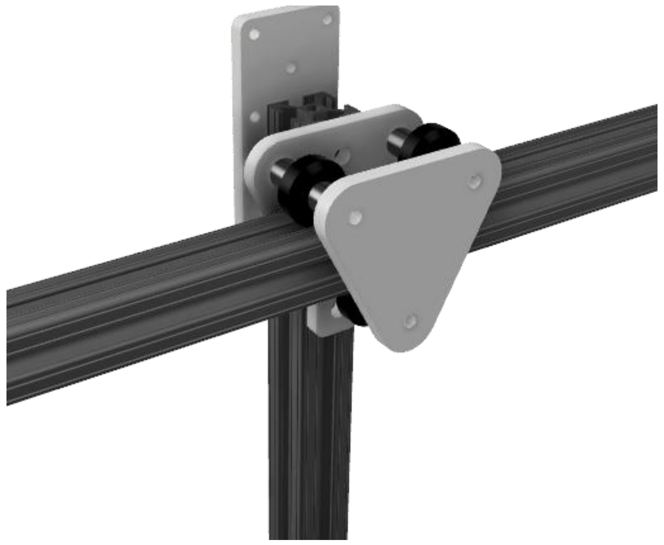


Fig. 1. Top X axis roller. X axis motion is achieved through a set of three wheels that ride on an aluminum 80 20 bar

The top and bottom roller are connected together via a vertical bar on which another Roller is placed in order to achieve Y axis motion(shown in Figure 2). With this setup the device is capable of traversing a two-dimensional space. The torque to move these three rollers is translated via a GT2 timing belt driven be stepper motors. The GT2 timing belt is attached onto the rollers belt slots and fastened in place using cable ties.

B. Z axis – linear actuatuaton.

The rack and gear setup shown in figure 2 allows for linear actuation by converting the rotational motion of a servo motor into a linear one. A dry erase marker which serves as a drawing tool is mounted onto the end of the rack. In order to detect when the pen reaches the wall, a button is place between the rack and the marker which then gets triggered when force is exerted on it.

C. Stepper motors

The X axis rollers are driven by two stepper motors(one at the top and another at the bottom) to prevent any skewing. The motor used here is a 12V DC NEMA 17 1.8 degrees 2-phase stepper motor with a GT2 timing pulley wheel with a pitch diameter of 12.73mm. Using the NEMA 17 datasheet, the calculations below are done in order to determine that the step size:

Pitch diameter of pulley, D = 12.73mm

Circumference of pulley $= \pi D = \pi \; x \; 12.73mm = 39.99 \; mm$

Step Size $= \frac{1.8 \; degrees}{360 \; degrees} \; x \; 39.99mm = 0.2mm$

The Y axis roller is driven by a 12V DC Unipolar 4 phase Howard Ind. P/N 1-19-4202 motor with a step angle of 3.6 degrees. The resulting step size is 0.4mm (twice the step size of the NEMA 17 motor). Section V shows how the difference in step sizes is factored into the code in order to achieve 1:1 aspect ratio when moving both axes by the same distance.

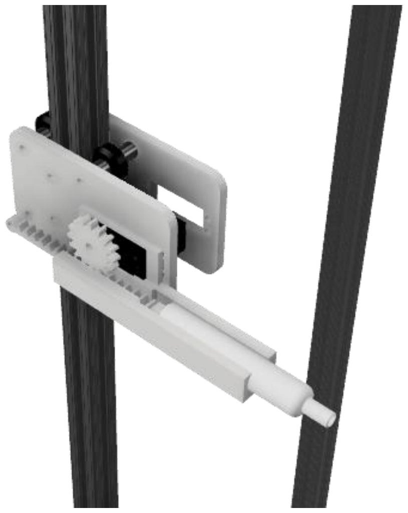


Fig. 2. Y axis roller with linear actuator mounted

III. Circuit Design

A. System Block diagram

Figure 3 and 4 show the system block diagrams for the actuator system and the sensor system. The actuator system starts from the high-level microcontroller where images are parsed to generate coordinates that are then passed to the low-level microcontroller over USB serial. The low-level microcontroller then sends signals to modulate the motor controllers which then move the motors.

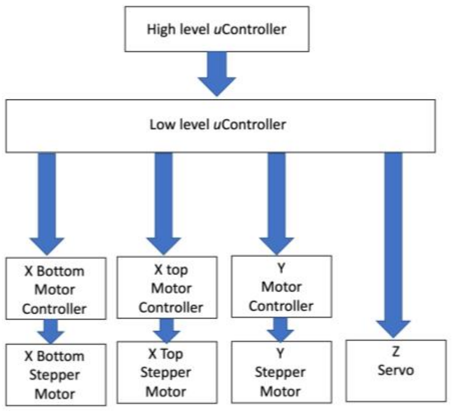


Fig. 3. Actuator system Diagram

Four stop switches are installed on the ends of two axes in order to serve as limit detectors. The system diagram below(Figure 4) shows these switches including the button connected to the Z motion marker. The stop switches are active high, and they are also connected to a CD4072B CMOS OR gate which sends its output to an interrupt pin of an Arduino. The purpose of this is to interrupt the procedural flow of the microcontroller's program in case any stop switch is pressed while printing.

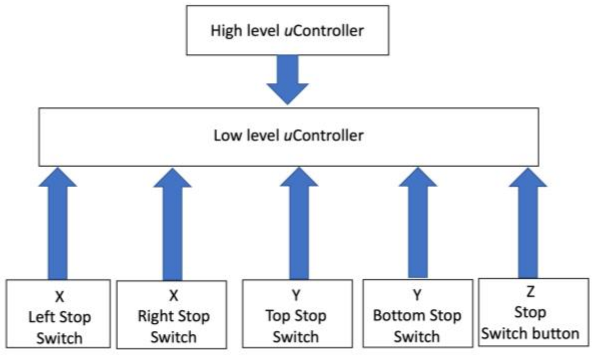


Fig. 4. Sensors system Diagram

B. Motor Controller Circuit.

Below are the electrical specifications of the stepper motors used:

TABLE I. Stepper Motor Electrical Specifications

Motor Specifications		
Specification	***NEMA 17***	***Howard Ind. P/N 1-19-4202***
Rated Voltage (V)	12	12
Rated Current(mA)	350	150

Given these motor specifications, the L293D Quadruple half H Motor Driver is selected as a driver because it’s Voltage supply range (4.5 – 3.6 V) encompasses that needed for the stepper motors and it has a peak output current of 1.2A Per channel. Figure 5 shows the circuit diagram for X top stepper. The circuit diagram for the other stepper motors resembles that of the X top stepper. A full schematic diagram of all the circuits is included in Appendix II and III.

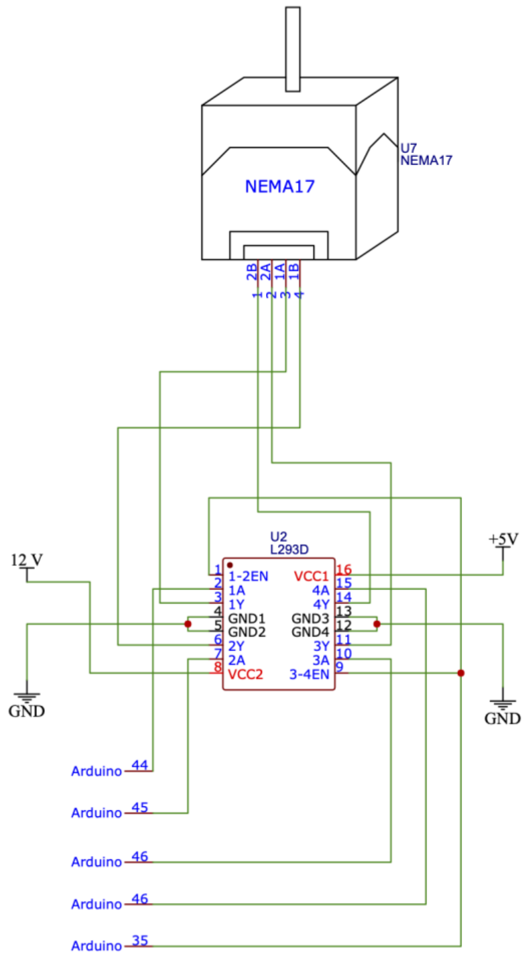


Fig. 5. Circuit diagram for motor contoller driving the X top stepper

C. *Linear Actuator*

A 180-degree servo motor with a 3D printed rack and gear attachment is used to realize linear actuation for the purpose of engaging and disengaging the marker from the dry erase board. Figure 6 below shows the circuit diagram for the servo:

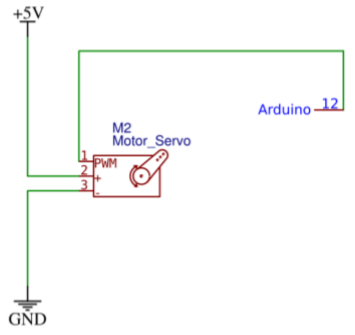


Fig. 6. Servo motor circuit for linear actuator

D. *Stop Switches*

The common terminal of each stop switch is connected to 5V source and the normally open(NO) terminal is connect to a 10K pull down resistor to make the switches active high. Each of the NO terminals is connected to a CD4072B CMOS OR gate whose output is connected to the interrupt pin of the Arduino Mega 2560. Figure 7 below shows the circuit diagram for the stop switches.

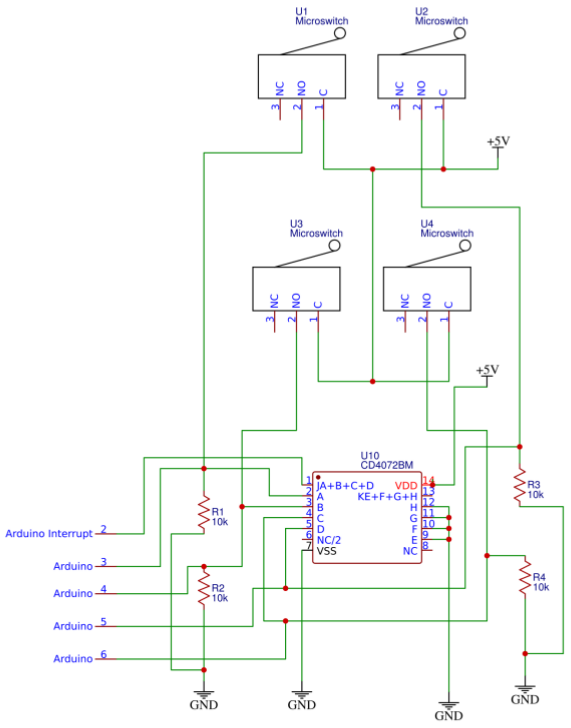


Fig. 7. Stop switch schematic

IV. Software Design - Low Level Microcontroller

As mentioned previously, the microcontroller used in this project is the Arduino Mega 2560. It was chosen because it has a higher number of IO pins and more memory in comparison to other Arduino models. The total number of pins used in this project is 22. The full program is provided in Appendix IV. Section A below is dedicated to walking through the functions implemented in the C program. Section B discusses the FSM design.

A. Functions and Abstractions

The C program consists of the following core functions each of which is discussed in this section:

- Move_y
- Move_x
- Move_z_linear
- Probe_z
- Move_axes
- Go_to_coordinates
- Calibrate
- Set_x_y
- ListenToPi
- ParseSerialData

The C program uses the stepper library in order control the steppers. Stepper library consists of an abstracted class called Stepper from which objects are instantiated by passing the following parameters respectively: number of steps per revolution, A lead of winding 1, B lead of winding 1, A lead of winding 2, B lead of winding 2. The code block below shows the instantiation of the three steppers:

```
// Initialize stepper object for x
motion
Stepper xTopStepper (200, X_TOP_PIN_1,
X_TOP_PIN_2, X_TOP_PIN_3, X_TOP_PIN_4);
Stepper xBtmStepper (200, X_BTM_PIN_1,
X_BTM_PIN_2, X_BTM_PIN_3, X_BTM_PIN_4);
// Initialize stepper object for y
motion
Stepper yStepper (200, Y_PIN_1, Y_PIN_2,
Y_PIN_3, Y_PIN_4);
```

Motion in the X axis is facilitated by two stepper motors(one at the top and one at the bottom). Since programs written in the Arduino IDE cannot write to multiple pins simultaneously a function that alternates between the top and bottom stepper motor is implemented as show below:

```
//x forward motion function
void move_x(int no_of_steps) {
  for (int i=1; i <= abs(no_of_steps);
i++){
    // move top x stepper
     xTopStepper.step(-
1*(no_of_steps/abs(no_of_steps)));
    // move bottom x stepper

xBtmStepper.step(no_of_steps/abs(no_of_s
teps));
  }
}
```

Motion in the Y axis relies on just one motor therefore the Stepper library’s step() function is sufficient; Move_y is a wrapper of step().

Motion in the Z direction is facilitated by a Servo motor. In the default state, the write function of the servo library takes an angle θ as a parameter and turn the servo to that angle. In order to achieve a rotation by θ instead of a rotation to θ, the function below uses a global variable z_pos to keep track of the position of the servo and from that position make adjustments:

```
// update z angle and write to servo
    z_pos += angle_step_size;
    zServo.write(z_pos);
```

In order to ensure that the pen touches the board when drawing, a probing function is written to measure and record the distance to the board. This function is called when the device is being calibrated.

```
int probe_z(void){
  int depth;
  while(!zCalibrateFlag){
    Serial.println("Calibration: probing
z");
    if(!digitalRead(SHARPIE_BTN)){
      zCalibrateFlag = true;   // reset
      depth = z_pos;           // store
      reset_z();               // go home
      break;
      }
    move_z_linear(1);
  }
  return depth;
}
```

In order to achieve **diagonal motion**, the X steppers and Y stepper would have to move at the same time. However, since Arduino IDE C programs cannot write to multiple pins simultaneously, a method similar to that of moving the X axis stepper is implemented in Move_axes. Move_axes follow the following conditions :

- If either X or Y is zero then move the non-zero axis; this is the normal single axis linear motion
- If X and Y are equal then alternate between the two axes moving each by one step to achieve motion along a 45° angle.
- If Y is greater than X then find the ratio Y/X and move the X-axis once for every Y/X steps in the Y-axis direction.
- If X is greater than Y then find the ration X/Y and move the Y-axis once for every X/Y steps in the X-axis direction.

The code for this is attached in the Appendix IV.

The **go_to_coordinates functions** is an abstraction of the move_axes function. It finds the difference between the current position(stored in a global variable) and the destination and passes that difference to the move_axes function as shown below.

```
void go_to_coordinates(int x, int y, int
z){
  // find difference between current
position and next position
  int x_diff = x - x_pos;
  int y_diff = y - y_pos;
  int z_diff = z - z_pos;
  //move to new postion and set x_pos,
y_pos and z_pos
  move_axes(x_diff, y_diff, z_diff);
  x_pos = x;
  y_pos = y;
  // z is updated in move_axes
}
```

The **calibrate function** is used to measure the width and height of the plane of motion and return the device to its origin. It leverages the stop switches to determine the limits. Below is the algorithm for calibration:

- Move to the left until the switch is triggered. Move to the right while counting steps. Return to the left. Reset global X position variable to zero.
- Move to the bottom until the switch is triggered. Move to the top while counting steps. Return to the bottom. Reset global Y position variable to zero.

The code for this is attached in the Appendix IV.

The purpose of **the set_x_y function** is to return the axes to home position and then apply an offset from the edges so as to keep the stop switches from constantly being pressed. The code for this is attached in Appendix IV.

The **listenToPi function** as the name suggests listens to the serial port for serial data. When serial data is available the function stores the data as an array of characters. The python program(discussed later in Section V) is designed to follow the following specific format when sending commands over serial:

```
GCODE,X,Y,Z\n
```

The **parseSerialData function** therefore iterates over the characters and extracts values for GCODE, X,Y and Z and storing them in global variables. The code for both listentoPi and parseSerialData is attached in the Appendix IV

B. *Finite State Machine(FSM)*

The main routine is written as an FSM with 5 states:

- Reset state
- Set home state
- Calibration State
- Listen State
- Print State

The transition table 2 below shows the Present State and Next State(PSNS) conditions:

TABLE II. FSM STATES TRANSITION TABLE

PSNS table			
Current State	***Output***	***Input***	***Next State***
RESET	Low power mode	Wake up from interrupt pin	SET HOME
		(No input)	RESET
SET HOME	Go home(0,0,0) and probe Z	Never calibrated(first time running)	CALIBRATION
		Calibrated	LISTEN
CALIBRATION	Invoke calibrate() function	Not calibrated	CALIBRATION
		Calibrated	SET HOME
LISTEN	Listen and parse serial data	“calibrate” command from serial	CALIBRATE
		“complete” command from serial	RESET
		G-code data from serial	PRINT
		(No input)	LISTEN
PRINT	Move axes to G-code (print)	Rollers triggering stop switch (error)	SET HOME
		(No input)	LISTEN

C. *Interrupt Service Routine(ISR)*

Two interrupt service routines are included in this program. The first has already been discussed in circuit design section (stop switch interrupt): if any of the stop switches are triggered while the device is in the PRINT state then the motors are shut down and an error flag variable is toggled. This stops the device from damaging any of the parts. Once the issue has been resolved the device resumes printing from where it left off.

The second ISR function is invoked by a reset button should the user decide to stop the device. This function shuts down the motor drivers indefinitely until the microcontroller is restarted.

V. Software Design – High Level Microcontroller

The python program and modules written for this device is intended to be executed in a raspberry pi. However, it is also compatible with computers that have the pyserial module and much of the empirical design and study of this device was done via serial communication from a computer. G-code used in those experiments is generated from an open source online module (cba mods) that converts png images to G-code files[1]. This section is dedicated to discussing how G-code is parsed using the gcode_parser.py module the interface between the python program and Arduino using the Ardino_interface.py module.

A. *Main application*

The main application takes user input from a G-code file. It then uses the gcode_parse.py module to parse the G-code into a format compatible with the listenToPi function(implemented in C Arduino) that has already been discussed in Section IV. The application then loops through the G-code file while sending each line to Arduino via serial using ArduinoInterface.py module. The code for the modules and the main application can be found in Appendix V.

B. *G-code parser module*

The G-code parser module consists of two main functions:

- Parse_gcode
- Rescale

The G-code parser module consists of two main functions: Parse_gcode function takes a G-code file and simplifys it by omitting G-code instruction that are irrelevant to DrawPrint such as rotation speed. It creates a sequence of strings that follow this general format: GCODE,X,Y,Z

The Rescale function takes an already parsed list and rescales the image such that the largest dimension of the image is as large as it's corresponding axis on the printer in terms of number of steps. Both the length and width of the image are scaled by the same factor so as to maintain aspect ratio. Since the step size of the Y stepper is twice as large as the step size of the X steppers (discussed in Section II C) then the Y steps are halved so as to prevent skewing.

C. *Serial interface module*

The serial interface module is built on top of the pyserial module. It consists of two functions

- Response
- Send

Response queries the serial port for data in form of lines and returns it as a string. Send takes a parsed G-code line and formats it to match the format below before sending it over serial:

```
GCODE,X,Y,Z\n
```

VI. Potential Areas of Improvement

A. *Printing rate*

The device is only able to print complete images when the stepper motors are running at 60rpm. At higher rpm values(>80rpm) the Y axis stepper motor has tendency to skip steps due to inertia. This could potentially be improved by replacing the motor with one of higher torque or adding one more stepper motor to the Y axis so as to increase the overall torque.

B. *Erasing*

The device does not automatically erase before printing. An erase feature would allow for the device to potentially erase and draw at the same time. This would also be useful as it would save time spent by the user erasing the board by hand.

C. *Cable management*

Cables from the Y roller occasionally get in the way of bottom X roller. This has the potential to damage the cables of the Y axis. A flexible tube or a spiral wrap would solve this challenge.

D. *Wireless Printing*

Raspberry pi has the capability of running a local server that can be accesses by any other device in the local network. A future area of improvement would be integrating the open source online cba mods[1] that converts png images to G-code directly with the printing application. Such a scenario would allow a user to upload print jobs directly to a website and have them printed. A website with device management tools is also a potential future improvement.

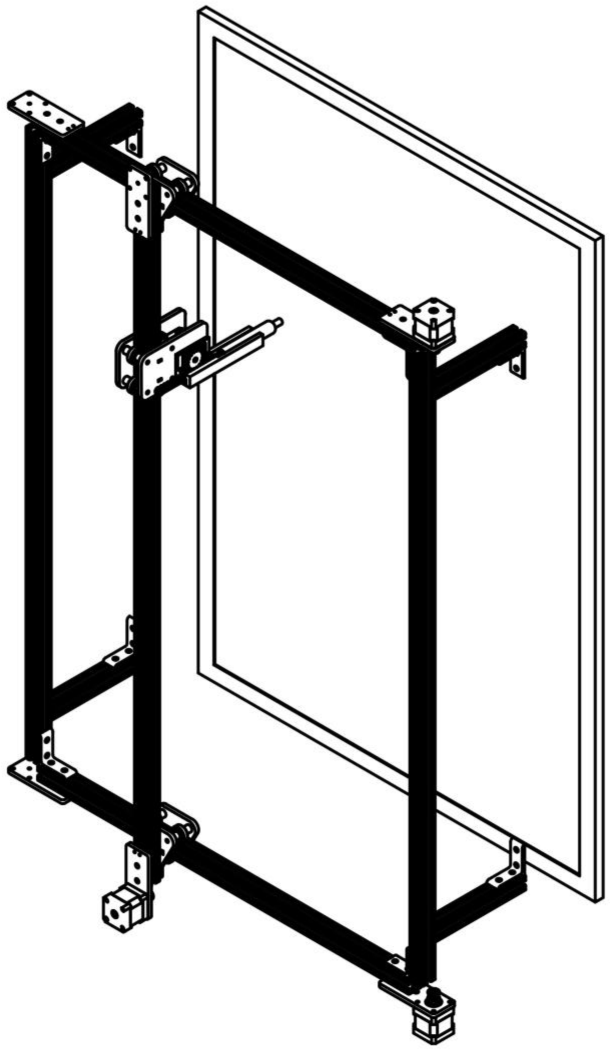
Acknowledgment

The author gratefully acknowledges the contribution of Professor David Abrams as an advisor for the Harvard Independent Research and Study Course (ES91hfr)

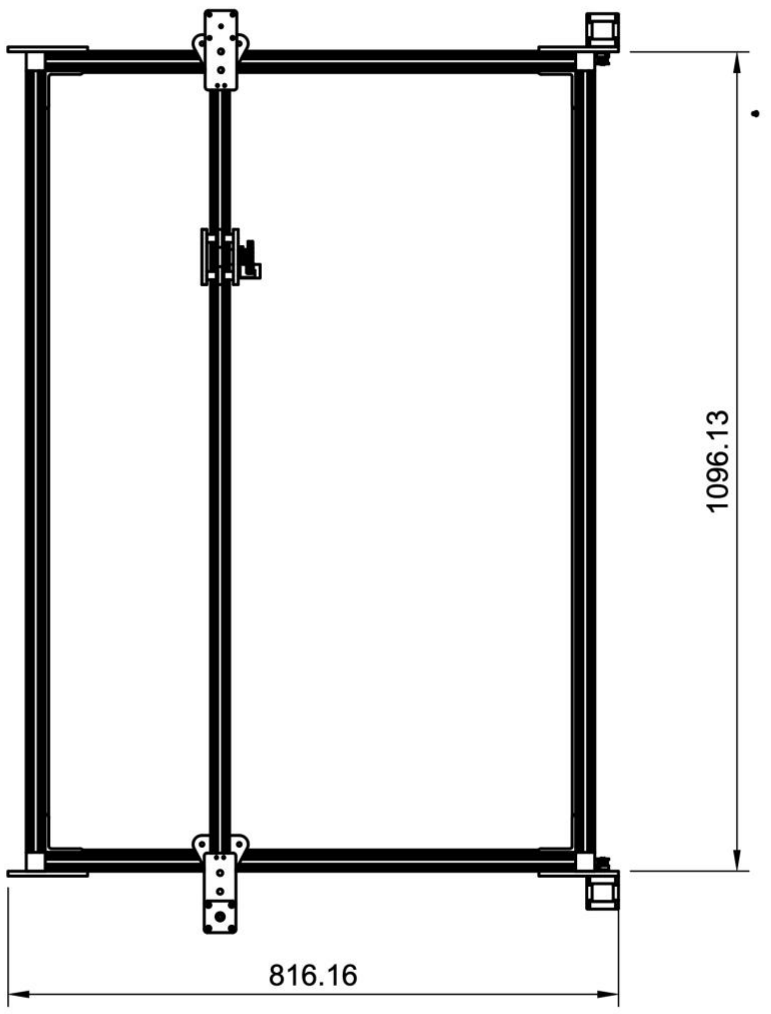
References

[1] "mods." [Online]. Available: http://mods.cba.mit.edu/. [Accessed: 10-Dec-2019].

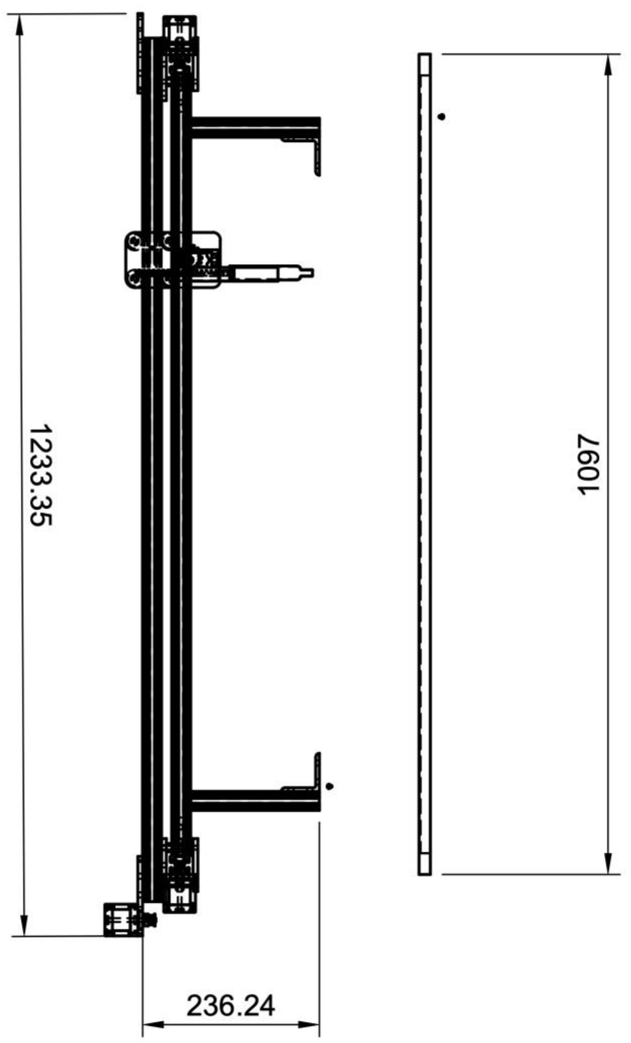
APPENDIX I – DRAWING PACKAGE



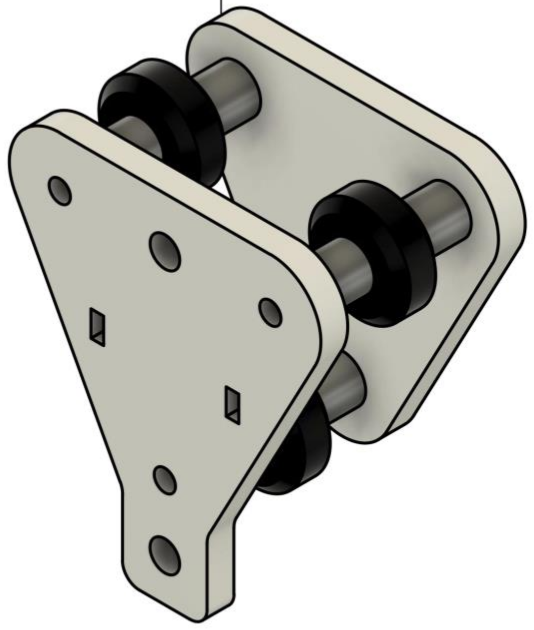
Dept.	Technical reference	Created by Billy Koech 12/11/19	Approved by		
		Document type	Document status		
		Title Full Device	DWG No.		
			Rev.	Date of issue	Sheet 1/9

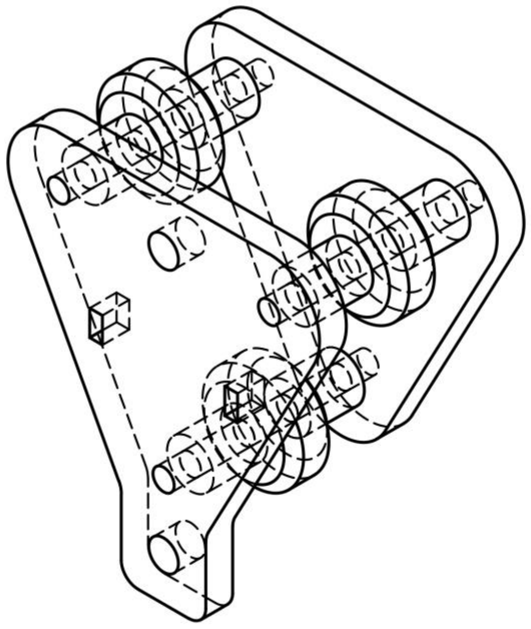


Dept.	Technical reference	Created by Billy Koech 12/11/19	Approved by
		Document type	Document status
		Title Full Device Front	DWG No.
			Rev. / Date of issue / Sheet 2/9

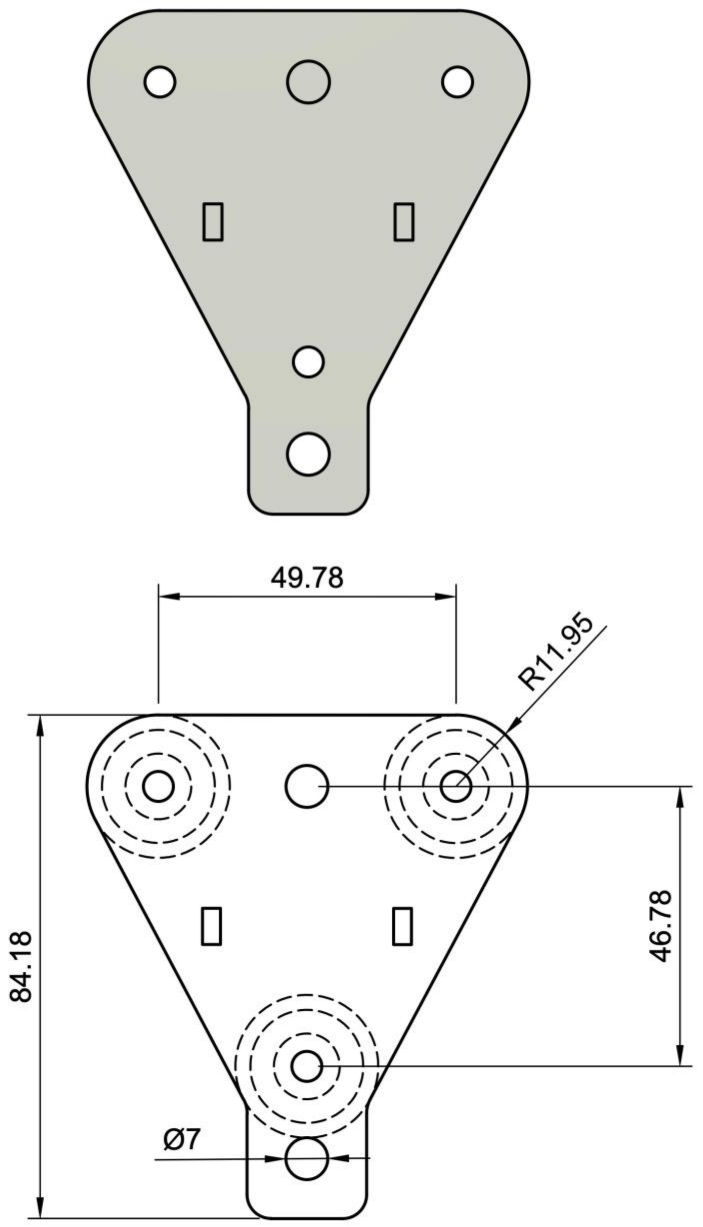


Dept.	Technical reference	Created by Billy Koech 12/11/19	Approved by
		Document type	Document status
		Title Full Device side	DWG No.
			Rev. | Date of issue | Sheet 3/9

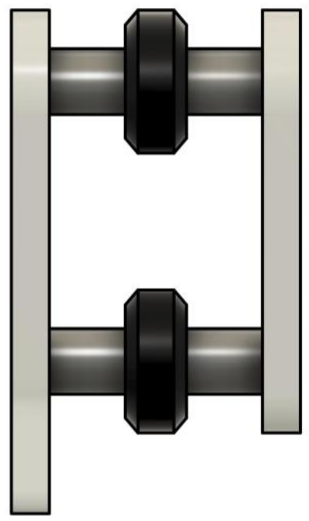


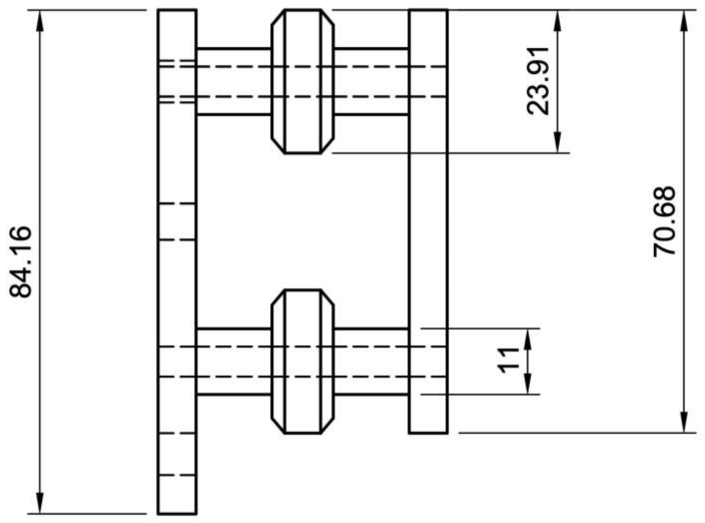


Dept.	Technical reference	Created by Billy Koech 12/11/19	Approved by		
		Document type	Document status		
		Title Top X roller	DWG No.		
			Rev.	Date of issue	Sheet 4/9

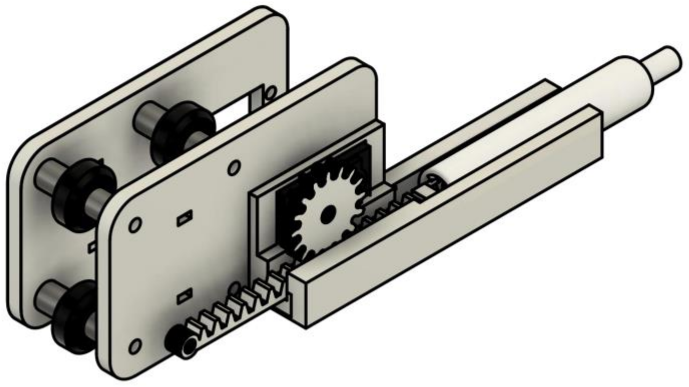


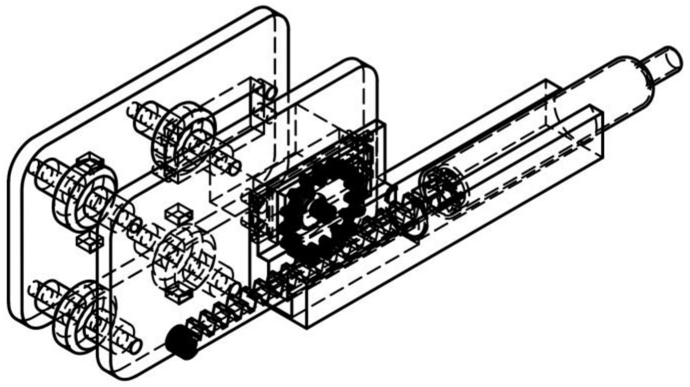
Dept.	Technical reference	Created by Billy Koech 12/11/19	Approved by
		Document type	Document status
		Title Top x Roller Front	DWG No.
			Rev. | Date of issue | Sheet 5/9



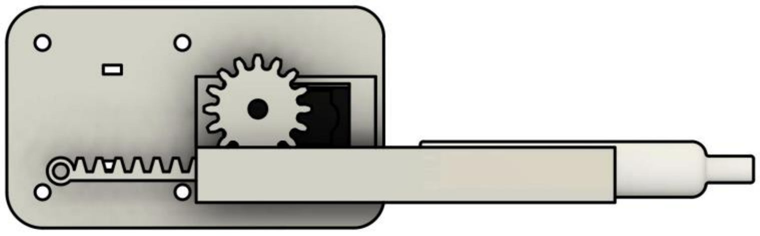


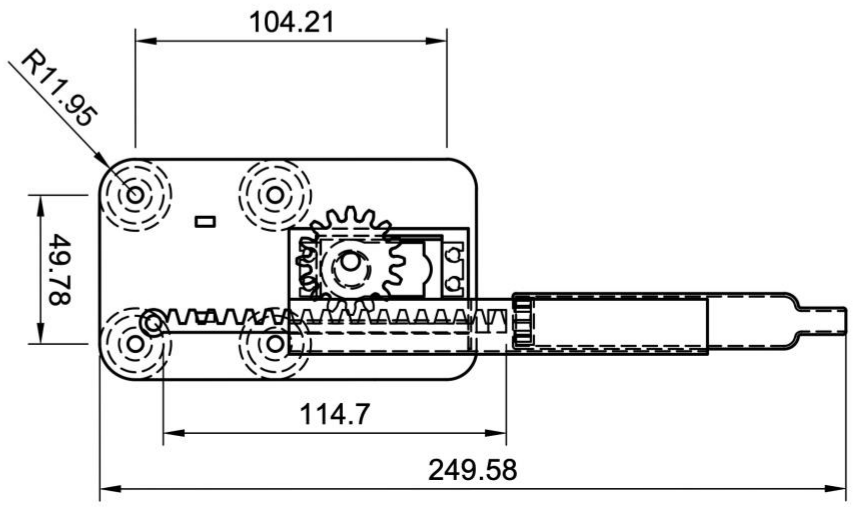
Dept.	Technical reference	Created by Billy Koech 12/11/19	Approved by
		Document type	Document status
		Title Top X Roller Side	DWG No.
			Rev. / Date of issue / Sheet 6/9



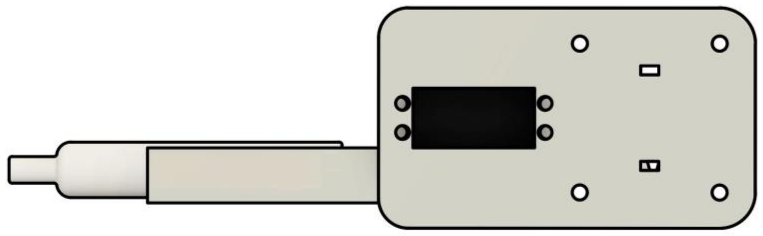


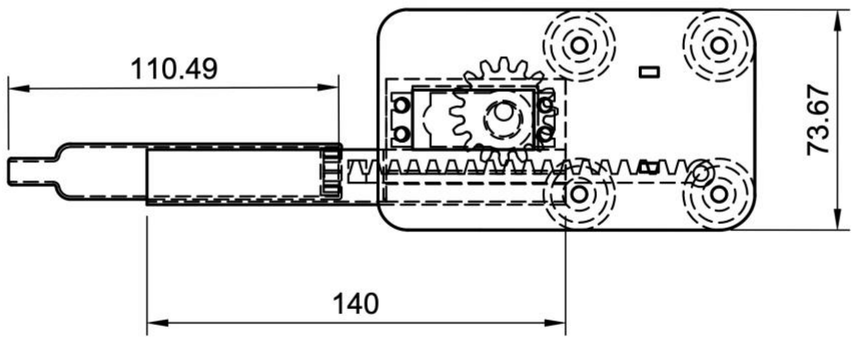
Dept.	Technical reference	Created by Billy Koech 12/11/19	Approved by
		Document type	Document status
		Title Y Roller	DWG No.
			Rev. / Date of issue / Sheet 7/9





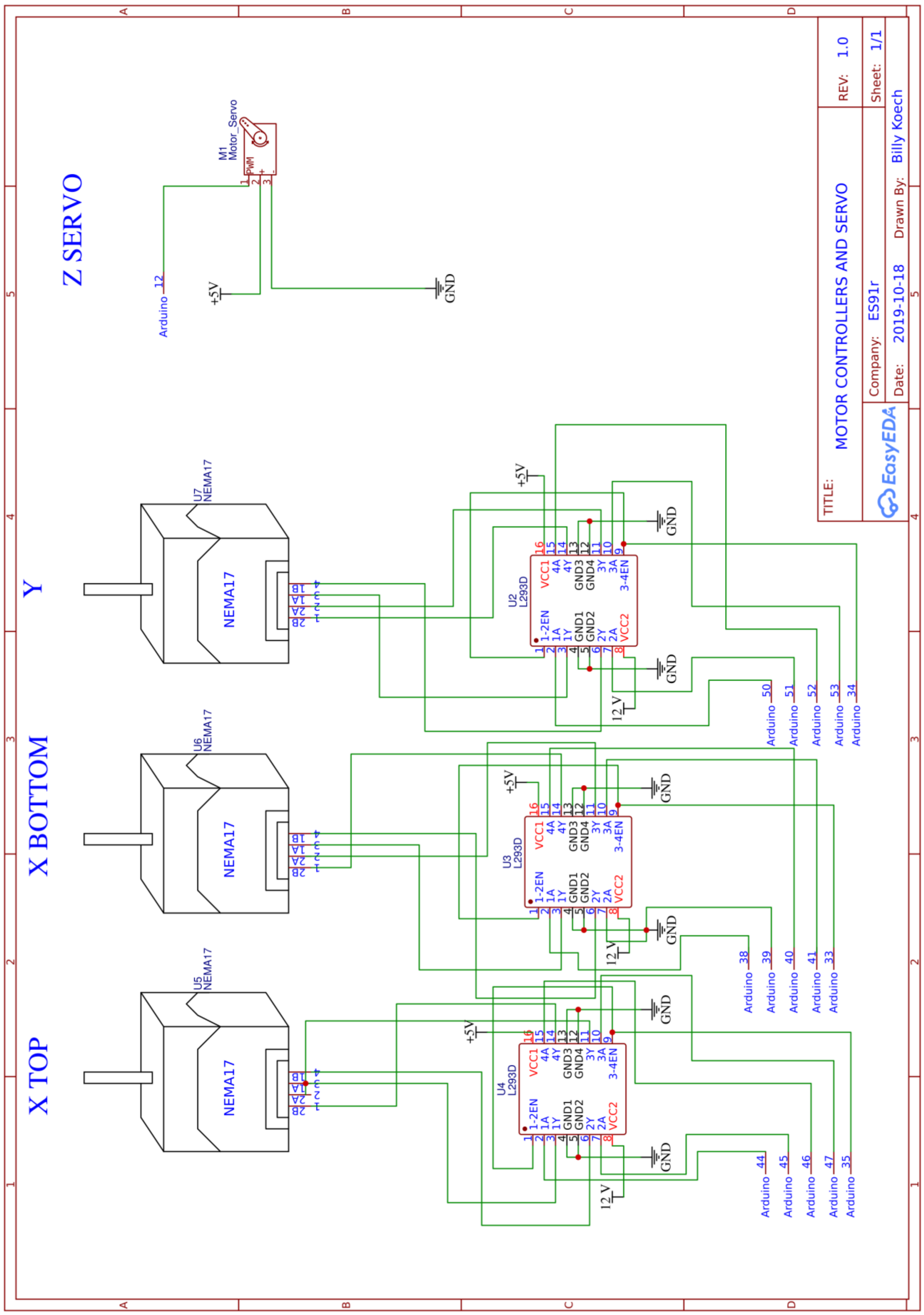
Dept.	Technical reference	Created by Billy Koech 12/11/19	Approved by
		Document type	Document status
		Title **Y Roller Side**	DWG No.
			Rev. / Date of issue / Sheet 8/9



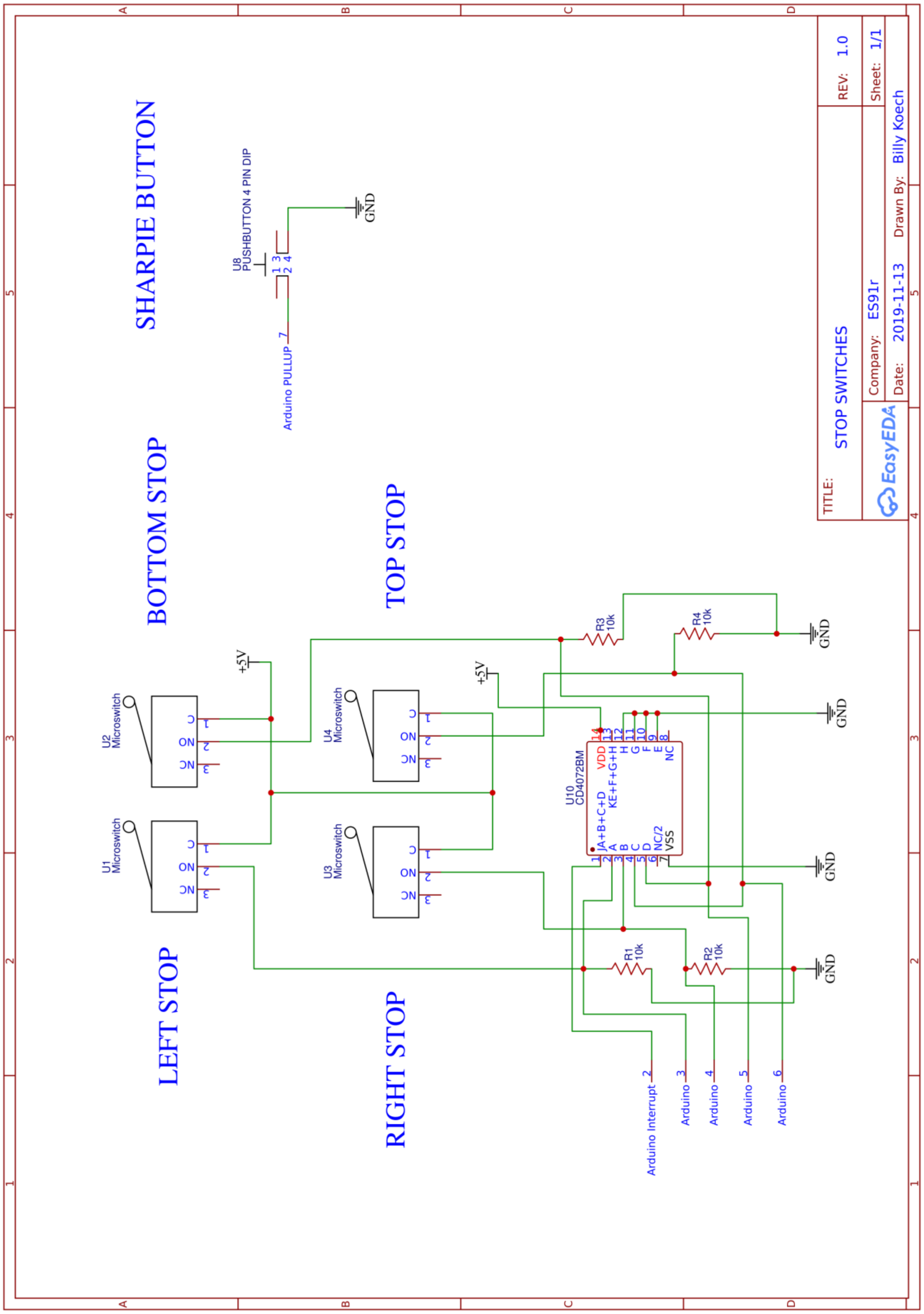


Dept.	Technical reference	Created by Billy Koech 12/11/19	Approved by		
		Document type	Document status		
		Title Y Roller Side	DWG No.		
			Rev.	Date of issue	Sheet 9/9

APPENDIX II – MOTOR CONTROLLER SCHEMATIC



APPENDIX IV – STOP SWITCH CIRCUIT SCHEMATIC



APPENDIX IV – ARDUINO C CODE

```
/*

Skeleton created by Billy Koech
Nov 15 2019
mru:
-- BKK  Nov 2019 - added move_z_linear
-- BKK Nov 2019 - refactored move_axes to traverse other angles
-- BKK Nov 2019 - updated calibrate function
-- BKK Nov 2019 - added set_x_y function
-- BKK Nov 2019 - added a reset switch
-- BKK Nov 2019 - Implemented FSM
-- BKK Dec 2019 - Removed all scratch and testing code


*/


//Declaration of pin numbers

// X TOP
#define X_TOP_PIN_1 44
#define X_TOP_PIN_2 45
#define X_TOP_PIN_3 46
#define X_TOP_PIN_4 47

//X BOTTOM
#define X_BTM_PIN_1 38
#define X_BTM_PIN_2 39
#define X_BTM_PIN_3 40
#define X_BTM_PIN_4 41

// Y STEPPER
#define Y_PIN_1 50
#define Y_PIN_2 52
#define Y_PIN_3 51
#define Y_PIN_4 53

//ENABLE PINS
#define Y_EN 34
#define X_TOP_EN 35
#define X_BTM_EN 33


// STOP SWITCHES
#define ALL_STOP_INTERRUPT 2
#define LEFT_STOP 3
#define RIGHT_STOP 4
#define TOP_STOP 6
#define BOTTOM_STOP 5
#define RST_INTERRUPT 18

// Sharpie
#define SHARPIE_BTN 13
```

APPENDIX IV – ARDUINO C CODE

```
//linear actuator servo
#define SERVO_PIN 12


// Load libraries and modules
#include <Stepper.h>
#include <Servo.h>


//Constants and Variables
int x_stepCount = 0; // number of steps it takes to cover the entire x length
int y_stepCount = 0; // number of steps it takes to cover the entire y length
int z_depth = 0;
int x_pos = 0; //default x position in steps
int y_pos = 0; //default y position in steps
int z_pos = 0; //default z position in degrees



// String data
String receivedData = "";
String parsed_g= "";
String parsed_x = "";
String parsed_y = "";
String parsed_z = "";

// Flags
bool serialDoneFlag = false;
bool zCalibrateFlag = false;
bool axesCalibration = false;


bool wakeUp = true;
bool errorFlag = false;


// FSM variables
int CurState;
int PrevState;


// Initialize stepper object for x motion
Stepper xTopStepper(200,X_TOP_PIN_1,X_TOP_PIN_2,X_TOP_PIN_3,X_TOP_PIN_4);
Stepper xBtmStepper(200, X_BTM_PIN_1,X_BTM_PIN_2,X_BTM_PIN_3,X_BTM_PIN_4);

// Initialize stepper object for y motion
Stepper yStepper(200,Y_PIN_1,Y_PIN_2,Y_PIN_3,Y_PIN_4);


//x forward motion function
void move_x(int no_of_steps) {
  for (int i=1; i <= abs(no_of_steps); i++){
    xTopStepper.step(-1*(no_of_steps/abs(no_of_steps))); // move top x stepper
    xBtmStepper.step(no_of_steps/abs(no_of_steps)); // move bottom x stepper

  }
}


//y motion function
void move_y(int no_of_steps) {
  yStepper.step(no_of_steps); // move top y stepper
}


// create servo object to control a servo
Servo zServo;



// params: angle size to move by. Negative values move backwards
void move_z_linear(int angle_step_size){

  // make sure final value is between 0 and 180
  if((z_pos + angle_step_size) < 181 && (z_pos + angle_step_size) > -1){
    // update z angle and write to servo
    z_pos += angle_step_size;
    zServo.write(z_pos);
  }else{
    // cant move any further
    Serial.print("Warning: z range limit reached. Z value");
    Serial.print(z_pos + angle_step_size);
    Serial.print("\n");

  }

}

// call this routine to reset z
void reset_z(void){
  z_pos = 0;
  zServo.write(z_pos);
}

int probe_z(void){
  int depth;
  while(!zCalibrateFlag){
    Serial.println("Calibration: probing z");
    if(!digitalRead(SHARPIE_BTN)){
      zCalibrateFlag = true;  // reset flag
      depth = z_pos;          // store depth
      reset_z();              // go back home
      break;
      }
    move_z_linear(1);
  }
  return depth;
}





// function to move all 3 axes simultaneosly
// takes x steps to move, z angle to set servo to and y steps to move

// Written to traverse different angles
void move_axes(int x_steps, int y_steps, int z_angle_steps){
  int xsteps = abs(x_steps);
  int ysteps = abs(y_steps);
  int ratio;


  // Move x and y at the same time
  //ratio between x and y
  if(y_steps==0){
    move_x(x_steps);

  }else if(x_steps==0){
    move_y(y_steps);

  }else if(y_steps==x_steps){
    while(xsteps!=0 || ysteps!=0){
      move_x(x_steps/abs(x_steps));
      xsteps -= 1;
      move_y(y_steps/abs(y_steps));
      ysteps -= 1;
     }

  }else if(ysteps>xsteps){
     ratio = ysteps/xsteps;
     while(ysteps>0){
      // for every step in x, move y by ratio
      move_x(x_steps/abs(x_steps)); // one step
      move_y(y_steps/abs(y_steps) * ratio); // one * ratio steps
      ysteps -= ratio;
     }

  }else if(xsteps>ysteps){
    ratio = xsteps/ysteps;
    while(xsteps>0){
      // for every step in y, move x by ratio
      move_y(y_steps/abs(y_steps)); // one step
      move_x(x_steps/abs(x_steps) * ratio); // one * ratio step
      xsteps -= ratio;
     }

  }else{
    Serial.println("Error: Undefined calibration condition");
  }

   // Move z
   move_z_linear(z_angle_steps);

}



void go_to_coordinates(int x, int y, int z){
  // find difference between current position and next position
  int x_diff = x - x_pos;
  int y_diff = y - y_pos;
  int z_diff = z - z_pos;

  //move to new postion and set x_pos, y_pos and z_pos
  move_axes(x_diff, y_diff, z_diff);
  x_pos = x;
  y_pos = y;
  // z is automatically updated
}

// calibration function
void calibrate(void){
  /* Step 1: Calibrate
   * Move hub towards -x until switch is triggered at the end
   * Move hub towards -y until switch is triggered at the top
   * Move +x while counting steps, store number of steps. No. of Steps = width.
   * Move -y while counting steps, store number of steps. No. of Steps = height.
   * Move hub back to origin, x = 0, y = 0
   */

   // flags
   bool leftTriggered = false;
   bool rightTriggered = false;
   bool topTriggered = false;
   bool btmTriggered = false;

   int switch_offset = 2; // steps away from switch

   x_stepCount = 0; // reset
   y_stepCount = 0; // reset


   // while left not triggered
   while(!leftTriggered){
    Serial.println("Calibration: moving left");
    // move x to left until switch is trigerred
    move_x(-1);
    if(digitalRead(LEFT_STOP)){
       leftTriggered = true;
       break;
```

```
     }

    }

    // while right not triggered
    while(!rightTriggered){
     Serial.println("Calibration: moving right");
     if(digitalRead(RIGHT_STOP)){
         rightTriggered = true;
         break;
      }
     // move x to right until switch is trigerred
     move_x(1);
     x_stepCount +=1; // count steps

     }

     //Todo: check step count and see if it matches empirical value
     move_x(-x_stepCount);//go back to origin
     x_pos = 0; //reset x coordinate to 0
     Serial.println("X calibration complete");


    //while bottom not triggered
    while(!btmTriggered){
     Serial.println("Calibration: moving down");
     if(digitalRead(BOTTOM_STOP)){
       btmTriggered = true;
       break;
       }
     // move y to bottom until switch is trigerred
     move_y(-1);

     }


    while(!topTriggered){
     Serial.println("Calibration: moving up");
     if(digitalRead(TOP_STOP)){
       topTriggered = true;
       break;
       }

     // move y to top until switch is trigerred
     move_y(1);
     y_stepCount +=1; // count steps
     }

    //Todo: check step count and see if it matches empirical value
    move_y(-y_stepCount);//go back to origin
    y_pos = 0; //reset y coordinate to 0

    Serial.println("Y calibration complete");
```

```
   Serial.print("X step count: ");
   Serial.println(x_stepCount);
   Serial.print("Y step count: ");
   Serial.println(y_stepCount);

}


// Function to go to home
void set_x_y(void){

   // flags
   bool leftTriggered = false;
   bool btmTriggered = false;

   int switch_offset = 100; // steps away from switch

   // while left not triggered
   while(!leftTriggered){
    Serial.println("Calibration: moving left");
    // move x to left until switch is trigerred
    move_x(-1);
    if(digitalRead(LEFT_STOP)){
       leftTriggered = true;
       break;
    }

   }

    //move away from switch
    move_x(switch_offset);

    x_pos = 0; //reset x coordinate to 0
    Serial.println("X home set");


   //while bottom not triggered
   while(!btmTriggered){
    Serial.println("Calibration: moving down");
    if(digitalRead(BOTTOM_STOP)){
      btmTriggered = true;
      break;
      }
    // move y to bottom until switch is trigerred
    move_y(-1);

    }

   //move away from switch
   move_y(switch_offset);

   y_pos = 0; //reset y coordinate to 0
   Serial.println("Y home set");
```

```
}




// fuction to listen for commands from master(Raspberry Pi)
void listenToPi() {
  serialDoneFlag = false;

  // Serial.print("Listening for serial data... \n");
  // Wait for data to be available
  while (Serial.available() > 0) {

    // store received data as a string
    // get new byte
    char inChar = (char)Serial.read();
    //add it to the receivedData
    receivedData += inChar;

    //break loop if incoming character is a new line
    if(inChar == '\n'){
      //if data received then break
      serialDoneFlag = true;
      break;
      }
  }

}


 void parseSerialData(String data){

  // variable to keep track of comma position
  //GCODE,X,Y,Z\n
  int commas = 0;

  //reset variables
  parsed_g= "";
  parsed_x = "";
  parsed_y = "";
  parsed_z = "";

  //iterate over data. 200 is the buffer size limit
  for(int i=0; i<=200; i++){

    //parse data
    if (data[i] == '\0'){ //end of data
      break; // exit
    }else if(data[i] != ',' && commas == 0){
      //get g part of g code
      parsed_g += data[i];
    }else if(data[i] !=',' && commas == 1){
      // get x value
      parsed_x += data[i];
    }else if (data[i] !=',' && commas == 2){
      // get y value
      parsed_y += data[i];
    }else if (data[i] !=',' && commas == 3){
      //get z value
      parsed_z += data[i];
    }

    //comma incrementation
    if(data[i] == ','){
      commas += 1;
    }else{
      // do nothing
    }

  }
 }


void setup() {

  //speed
  xTopStepper.setSpeed(60); //rpms
  xBtmStepper.setSpeed(60); //rpms
  yStepper.setSpeed(60); //rpms

  // attaches the servo on pin 9 to the servo object
  zServo.attach(SERVO_PIN);


  // reserve 200 bytes for the receivedData:
  receivedData.reserve(200);

  //reserve
  parsed_g.reserve(5);
  parsed_x.reserve(5);
  parsed_y.reserve(5);
  parsed_z.reserve(5);

  // initialize the serial port:
  Serial.begin(9600);

  //setup pinmode for actuators
  pinMode(X_TOP_PIN_1, OUTPUT);
  pinMode(X_TOP_PIN_2, OUTPUT);
  pinMode(X_TOP_PIN_3, OUTPUT);
  pinMode(X_TOP_PIN_4, OUTPUT);
  pinMode(X_BTM_PIN_1, OUTPUT);
  pinMode(X_BTM_PIN_2, OUTPUT);
  pinMode(X_BTM_PIN_3, OUTPUT);
  pinMode(X_BTM_PIN_4, OUTPUT);
  pinMode(Y_PIN_1, OUTPUT);
  pinMode(Y_PIN_2, OUTPUT);
  pinMode(Y_PIN_3, OUTPUT);
  pinMode(Y_PIN_4, OUTPUT);
  pinMode(Y_EN, OUTPUT);
  pinMode(X_TOP_EN, OUTPUT);
  pinMode(X_BTM_EN, OUTPUT);

  // set pinmode for sensors
  pinMode(ALL_STOP_INTERRUPT, INPUT);
  pinMode(LEFT_STOP, INPUT);
  pinMode(RIGHT_STOP, INPUT);
  pinMode(TOP_STOP, INPUT);
  pinMode(BOTTOM_STOP, INPUT);
  pinMode(SHARPIE_BTN, INPUT_PULLUP);
  pinMode(RST_INTERRUPT, INPUT_PULLUP);


  //Make sure drivers are off
  digitalWrite(X_TOP_EN, LOW);
  digitalWrite(X_BTM_EN, LOW);
  digitalWrite(Y_EN, LOW);

  // reset angle for sharpie
  reset_z();

  //attach interupts
  attachInterrupt(digitalPinToInterrupt(ALL_STOP_INTERRUPT), isr, RISING);
  attachInterrupt(digitalPinToInterrupt(RST_INTERRUPT), shutdown_motors, FALLING);


  //initialize fsm
  CurState = 0;
  PrevState = -1; // undefined

  Serial.println("Initialized");

}

void loop() {
  // put your main code here, to run repeatedly:

    // print state if state has changed
    if(CurState != PrevState){
      Serial.print("Current State: ");
      Serial.println(CurState);
    }
    // Update previous state
    PrevState = CurState;

    // FSM goes here
    switch (CurState){
      case 0: // RESET STATE
```

```
        // Low power mode

        // disable motor drivers
        digitalWrite(X_TOP_EN, LOW);
        digitalWrite(X_BTM_EN, LOW);
        digitalWrite(Y_EN, LOW);


        // wait for interrupt from raspberry pi
        if(wakeUp){
          // go to home
          CurState = 1;
        }else{
          // stay in current state
          CurState = 0;
        }

        break;

    case 1: //SET X Y HOME

        // enable motor drivers
        digitalWrite(X_TOP_EN, HIGH);
        digitalWrite(X_BTM_EN, HIGH);
        digitalWrite(Y_EN, HIGH);
        // go to home
        set_x_y();
        z_depth = probe_z();
        Serial.print("Probed z val: ");
        Serial.println(z_depth);

        if(!axesCalibration){
          // calibrate
          CurState = 2;
        }else{
          // listen state
          CurState = 3;
        }


        break;

    case 2: // CALIBRATION STATE

        calibrate();
        axesCalibration = true;

        // if not calibrated stay
        if(!axesCalibration){
          // calibrate
          CurState = 2;
        }else{
          // set home state
          CurState = 1;
         }

         break;

     case 3: // LISTEN STATE

         listenToPi(); // retrieve data

         // process data
         if(serialDoneFlag){
             // parse data
             parseSerialData(receivedData);
             // reset data holder
             receivedData = "";
             // Wait for data to be processed
             if(parsed_g == "" || parsed_x == "" || parsed_y == "" || parsed_z ==
"") {

               Serial.print("<---Something has not loaded yet --->");

             }else{
               Serial.print("<---Loaded --->");
             }
             //print parsed data
             Serial.println("parsed_g: " + parsed_g);
             Serial.println("parsed_x: " + parsed_x);
             Serial.println("parsed_y: " + parsed_y);
             Serial.println("parsed_z: " + parsed_z);
         }

         // process inputs
         if(parsed_g == "calibrate"){
           CurState = 2;  //calibration state
         }else if(parsed_g == "complete"){
           CurState = 0; // go back to reset state
         }else if(parsed_g == "G00Z" || parsed_g == "G01Z" || parsed_g == "G00X" ||
parsed_g == "G01X" || parsed_g == "Z"){
           CurState = 4; // print
         }else{
           CurState = 3; // stay here
         }

         break;


     case 4: // PRINT STATE

         // To do: optimize for rapid motion when not printing
         // for z axis indipendent motion
         if(parsed_g == "G00Z" || parsed_g == "G01Z" || parsed_g == "Z"){
           go_to_coordinates(x_pos, y_pos, parsed_z.toInt() * z_depth);
         // normal motion
```

```
          }else{
            go_to_coordinates(parsed_x.toInt(), parsed_y.toInt(), parsed_z.toInt() *
z_depth);
          }

          Serial.println("Done go_to_coordinates");

         // reset parsed data
            parsed_g= "";
            parsed_x = "";
            parsed_y = "";
            parsed_z = "";


          if(errorFlag){
            // if error then reset
            // To do: send message to RPI to pause
            CurState = 1;
            errorFlag = false;
          }else{
            CurState = 3;
          }

          break;


      default:
      Serial.println("Default state");
      break;
    }
}


void shutdown_motors(void){
  //Make sure drivers are off
  digitalWrite(X_TOP_EN, LOW);
  digitalWrite(X_BTM_EN, LOW);
  digitalWrite(Y_EN, LOW);
  reset_z();
}

void isr(){
  Serial.print("STOP SWITCH Triggered, State=>");
  Serial.println(CurState);
  // if printing
  if(CurState == 4){
  //shutdown and reset
  digitalWrite(X_TOP_EN, LOW);
  digitalWrite(X_BTM_EN, LOW);
  digitalWrite(Y_EN, LOW);
  reset_z();

  // calibration required
  errorFlag = true;
  }

}
```

APPENDIX V – A – PYTHON MAIN APPLICATION

```
##############################################################################
"""
Module to send commands data to arduino and drive printer

Created by: skeleton by Billy Koech
Date:       Novembe 13 2019

mru(most recent update):
November 13 2019   -
                  -
                  -

"""
##############################################################################


# Import modules and libaries

# user modules
import ArduinoSerial_Mac_Interface as AI
import gcode_parser as GP

# Variables

# Constants

# Troubleshooting
DEBUG = True


# Open file and parse data
file_name = input("File path: ")
if(file_name == ""):
    file_name = "testgcode/square1.nc"  # default
else:
    pass  # use user input


parsed_data = GP.parse_gcode(file_name)  # parse

# rescale data to fit to the printer's window
rescaled_parsed_data = GP.rescale(parsed_data)


# iterate over data and send line by line
for gcode in rescaled_parsed_data:
    AI.send(gcode)  # send
    while ("Done go_to_coordinates" not in AI.response()):
        if(gcode[0] == "Inches" or gcode[0] == "Millimeters"):
            print("Units: " + gcode[0])
            break  # skip sending this
        else:
            print("Going to coordinates ... " + str(gcode))  # wait for data to be
processed

AI.send(("G01X", "0", "0", "0"))  # return home
```

```
################################################################################
"""
Program to parse g-code into a list

Created by: skeleton by Billy Koech
Date:      Nov 4th 2019

mru(most recent update):
Nov 8th 2019    - deleted one of the .rstrip() from remove_extras
                - added write_non_string_data function
                - added print statements to start_log() and stop_log()

"""
################################################################################


# import modules

# global varibles and CONSTANTS
# STEP_SIZE = 0.2  # mm per step for NEMA 17 motor
# X_LENGTH = 1400 * STEP_SIZE  # Replace these with actual dimensions
# Y_LENGTH = 550 * STEP_SIZE  # Replace these with actual dimensions


DEVICE_X_STEPS = 2000  # steps
DEVICE_Y_STEPS = 2000  # steps

# Troubleshooting
DEBUG = True


# Function to convert file to lines; reads lines in file
# return list of of lines
def get_lines(file_name):

    # open and store in list
    file = open(file_name, "r")
    _data = file.readlines()
    # close
    file.close()

    # remove end of line character
    # new_list = []
    # for el in _data:
    #     new_list.append(el.rstrip())

    # return new_list
    return _data


# Function to parse G code into action, x, y, z list of tuples
# iterate over each element
```

```
def parse_gcode(file_name):
    # convert to list
    g_code_list = get_lines(file_name)

    output = []  # variable to hold output
    for instr in g_code_list:
        # check first three character then parse data; if first val == G00

        # rapid independent Z motion
        if instr[0:4] == "G00Z":
            output.append((instr[0:4], 0, 0, instr[4:instr.index("\n")]))
        # z linear move
        elif instr[0:4] == "G01Z":
            output.append((instr[0:4], 0, 0, instr[4:instr.index(" ")]))
        # x y z rapid move
        elif instr[0:4] == "G00X":
            output.append((instr[0:4], instr[4:instr.index('Y')],
                           instr[instr.index("Y") + 1: instr.index("Z")],
                           instr[instr.index("Z") + 1: instr.index("\n")]))
        # x y z linear move
        elif instr[0:4] == "G01X":
            output.append((instr[0:4], instr[4:instr.index('Y')],
                           instr[instr.index("Y") + 1: instr.index("Z")],
                           instr[instr.index("Z") + 1: instr.index("\n")]))
        #
        elif instr[0] == "Z":
            output.append((instr[0], 0, 0, instr[1:instr.index("\n")]))
        elif instr[0:3] == "G20":
            output.append(("Inches", 0, 0, 0))
        elif instr[0:3] == "G21":
            output.append(("Millimeters", 0, 0, 0))
        else:
            pass

    return output


# Function to scale x, y, z accordingly
# returns list of (Gcode, x, y, z) where x y z are in steps
def rescale(parsed_g_code_list):
    # convert all numbers from str to float
    float_parsed_gcode = []

    for each in parsed_g_code_list:
        # (G code, X, Y, Z)
        float_parsed_gcode.append((each[0], float(each[1]),
                                   float(each[2]), float(each[3])))

    # store x and y in separate lists
    x_list = []
    y_list = []
    for each in float_parsed_gcode:
        x_list.append(each[1])
        y_list.append(each[2])

    # find largest y value
    image_height = max(y_list)
    # find largest x value
    image_width = max(x_list)

    # rescale according to the larger dimension
    scale_factor = 1  # variable to store scale factor
    if image_width > image_height:
        scale_factor = DEVICE_X_STEPS / image_width
    elif image_height > image_width:
        scale_factor = DEVICE_Y_STEPS / image_height
    else:  # they must be equal
        scale_factor = DEVICE_X_STEPS / image_width
# DIVIDED Y BY HALF IN ORDER TO FIX Vertical scaling.
# Vertical scaling happens due to difference in motor step sizes
    output = []  # variable to hold output
    for each in float_parsed_gcode:
        output.append((each[0], int(each[1] * scale_factor),
                       int((each[2] * scale_factor) / 2),
                       int((each[3] * scale_factor) < 0)))

    return output


def main():
    if DEBUG:

        # Test for get_lines
        list_of_lines = get_lines("square-line-png-2.png-2.nc")
        print("\n list of lines: \n")
        for each in list_of_lines:
            print(each)

        # Test for parse_gcode
        print("\n parsed g code data: \n")
        parsed_gcode_data = parse_gcode("square-line-png-2.png-2.nc")
        print(parsed_gcode_data)

        # Test for rescale
        print("\n x y z rescaled and converted to steps: \n")
        rescaled_data = rescale(parsed_gcode_data)
        print(rescaled_data)
    else:
        pass  # do nothing


if __name__ == '__main__':
    main()
```

APPENDIX V – C – PYTHON ARDUINO SERIAL INTERFACE MODULE

```
###############################################################################
"""
Module to communicate with Arduino over serial on a computer

Created by: skeleton by Billy Koech
Date:       Nov 1st 2019

mru(most recent update):
Nov 1st 2019    -
                -
                -

"""
###############################################################################


# Import modules and libaries
import serial

# set device and bitrate
ser = serial.Serial('/dev/tty.usbmodem14201', 9600, timeout=0.5)  # timeout of 0.5
seconds

# Wait for arduino to initialize
while(True):
    serial_data = ser.readline().decode()
    if("Initialized" not in serial_data):
        print(serial_data)
    else:
        break

print("Serial Ready")

# get response over serial
def response():
    Response_val = ser.readline().decode()
    print("RESPONSE: " + Response_val + "\n")
    return Response_val

# format and send data over serial
def send(_gcode_data):
    # package data
    _data = _gcode_data[0] + "," + str(_gcode_data[1]) + "," + str(_gcode_data[2]) +
"," + str(_gcode_data[3]) + "\n"
    # encode and send command to arduino over serial
    ser.write(_data.encode())
```