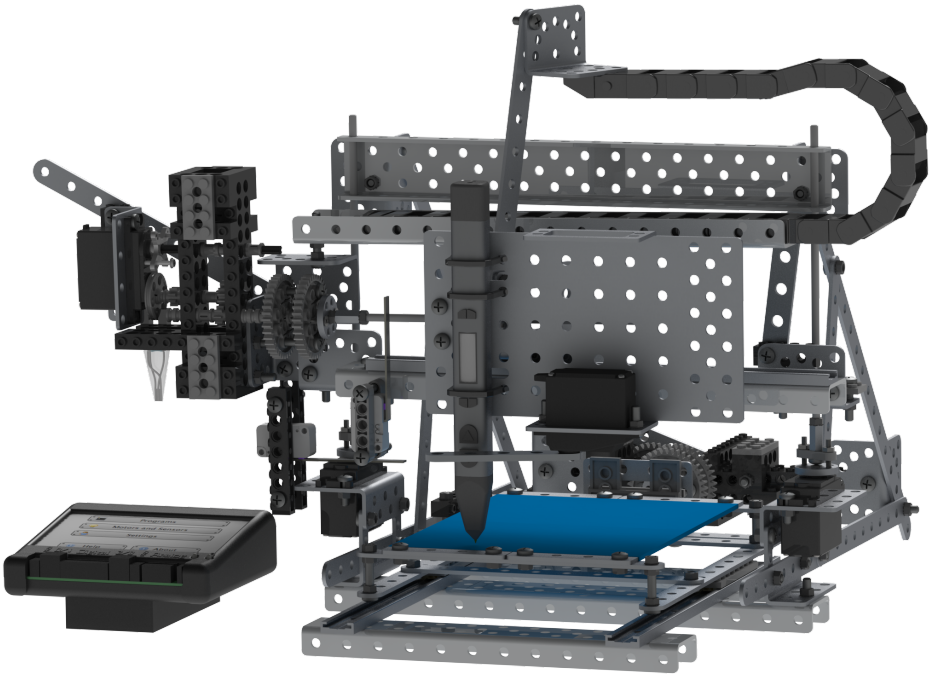


3D Pen Printer

Created with KIPR Botball Kit Parts
and the AIO 3D Printing Pen

The University of Oklahoma
Aerospace and Mechanical Engineering School
212 Felgar Hall, 865 Asp Avenue
Norman, OK 73019

designed and created by
Riley Cotter
rileycotter@ou.edu

supervised by
Dr. David Miller

INTRODUCTION

This experimental 3D printer was created with parts that would come in a KIPR Botball Kit and the AIO 3D Printing Pen. The printer uses FDM (Fused Deposition Modeling), an additive manufacturing technology, to produce any 3D model. The program 3DPrinter was written specifically to run on the KIPR Wallaby. 3DPrinter finds a gcode file on a USB placed into the Wallaby and executes the gcode commands. CURA an opensource 3D model slicing application was used to generate the gcode. PLA (Polylactic Acid), a biodegradable bioplastic, was the printing material chosen and recommended for the AIO 3D Printing Pen.

LINKS TO SUPPLEMENTARY RESOURCES

User Manual Supporting Documentation.

- Assembly video of the 3d Pen Printer can be found at:
  https://www.youtube.com/watch...
- GitHub link to all the **Code** used and SolidWorks Assembly:
  https://github.com/RileyCotter/3DPenPrinter
- Cura Quick Start Guide:
  https://ultimaker.com/en/resources/34185-quick-start-guides
- A video of the running 3D Pen Printer can be found at:
  https://www.youtube.com/watch...

Additional Links.

- Aerospace and Mechanical Engineering at University of Oklahoma can be found at:
  http://www.ou.edu/coe/ame.html
- Botball Kits can be found at:
  http://botballstore.org/categories/botball
- Information about KIPR can be found at:
  https://www.kipr.org/
- The AIO 3D Printing Pen can be found at:
  http://www.zeus.aiorobotics.com/3dpen
- Cura: Open Source 3D printer slicing application can be found at:
  https://ultimaker.com/en/products/ultimaker-cura-software

CONTENTS

1 Introduction **1**

2 Links to Supplementary Resources **1**

3 3D Printer Operation **3**

4 Physical Features **8**

5 Code Features **10**

6 Printer Assembly **12**

7 Wire Diagram **28**

8 Parts Required **29**

9 Appendix **34**

Step 1. Design/Obtain (.STL) or another acceptable file

- **Design** a 3D part.
  -There are two basic forms of 3D modeling. Solid modeling and Surface modeling.

  -Solid modeling creates solid 3D models of as if they real world solid parts. Solid modeling is typically used for detailed high precision parts made for use rather than aesthetic.[4] A popular free Solid modeling software is SketchUpFree:3D.

  -Surface modeling creates a surface model of the part, using only the skin of the part. Surface modeling is typically used for aesthetic purposes, where the dimensions of the part are not critical.[4] A popular free Surface modeling software is blender.

  -Regardless of whether the part is created using Solid or Surface modeling, the part must be made with some foresight that it is being 3D printed. Avoid overhangs, use a wall thickness of of atleast 0.8mm, and avoid large flat surfaces that may warp during the print.[5]

- **Obtain** a 3D part.
  -If you don't have any ideas of what to print there are many places online to download and share 3D models.

  -Thingiverse, GrabCAD, and 3D Warehouse have active communities of designers and makers. The majority of parts on these websites are free, but there is a marketplace setup to sell prints as well.

- Cura accepts STL, OBJ, 3MF, and X3D file formats.

- STL file format tends to be the most typical file format for sharing 3D models and has all the required information need for this 3D Printer. STL format is commonly used in 3D printing because many 3D printer slicing software support it and most CAD software has the ability to export models in STL format. STL format file sizes are typically smaller than other formats because the file generates only the surface geometry of the modeled object. For this reason 3D model makers and 3D printing enthusiasts can easily share models on the internet without long download and upload times.[3]

Step 2. Upload and Slice with Slicer

- For Step 2: the instructions reflect those given in the Cura Quick Start Guide listed in the additional links section.

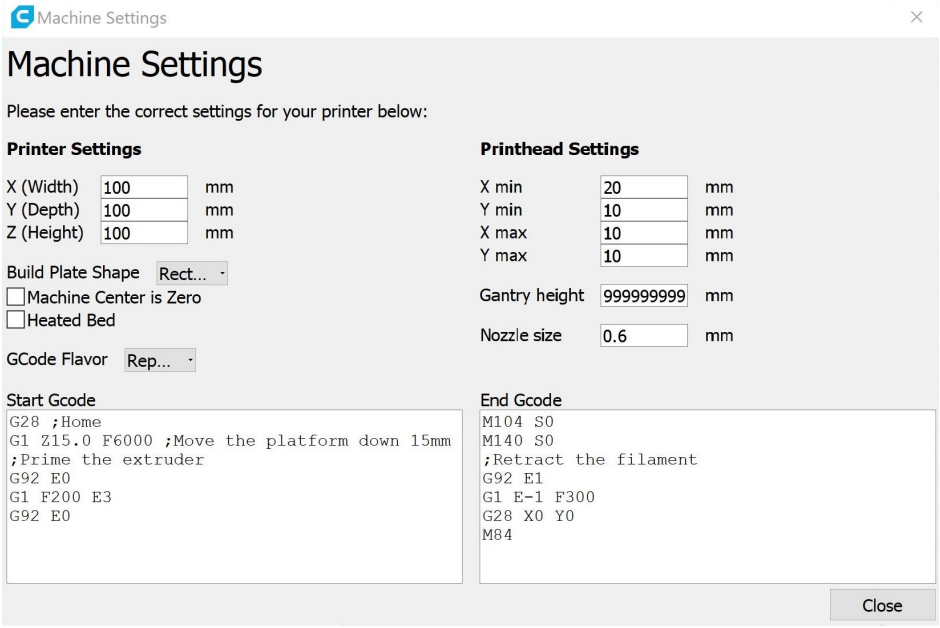
- **Confirm** that the 3D printer has been added correctly.
  -Go to Settings > Printer > Add Printer

  -Select *Custom Printer* from drop down menu.

  -At the bottom of that window you will be able to name the 3D printer as you wish.

  -You should now see the Machine Settings window shown below. If you do not, go to Settings > Printer > Manage Printers and select machine settings on the right hand side.

  -Fill out your custom printer machine settings to match the figure below.



  - Note: The Nozzle size is a parameter worth looking at for tuning the accuracy of the print.

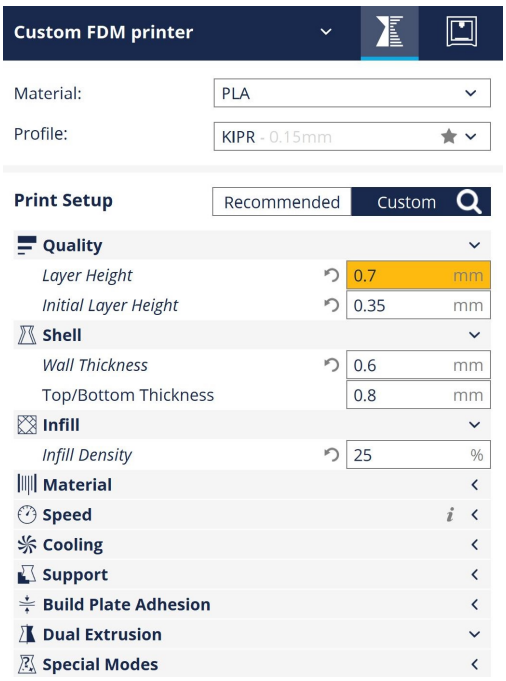
- **Upload** the STL file.
  -Find the saved STL model file on your computer, and load it into Cura.

- **Adjust** the model on the print bed.
  -Using the: Move, Scale, Rotate, and Mirror tools available on the left hand side of the window, position the model on the 3D print bed.

  -Keep in mind that you want to position the part in a way that it will print strong and accurately. 3D printed parts tend to fail at the layer mating surfaces (along the grain). And avoid as much unsupported overhang as possible.

- **Check** slicer settings.
  -Select the Custom icon in the slicer settings.

  -Fill out your custom slicer settings to match the figure below.



  -The closed drop down menus either not supported by the code written for the 3D printer or not advised to be turned on. Except, potentially turning on the Build Plate Adhesion to produce a Skirt to prep the filament in the nozzle for the print.

  -Note: The Quality, Shell, and Infill may all need to be changed while tuning the 3D printer.

- **Preview** model layers.
  -Click on the View Mode icon that is just under the positioning tools on the left side of the window.

  -View Mode allows you to check the infill and print path of each layer using the slider.

- **Save** to USB.
  -Cura should detect your USB when plugged in.

  -If it doesn't you can still hit the save to file button in the right bottom corner of the window, and save the g-code file to your USB.

  -Name the file something you can easily remember as you will need the name while configuring the code.

  -Safely eject your USB from the computer and plug it into the USB ports at the top of the Wallaby.

Step 3. Configure Code on the Wallaby

- **Connect** Computer to Wallaby.
  -There are two ways of connecting the Wallaby to your computer. Wired or Wireless.

  -Wired (preferred method). Attach the Wallaby to the computer using the USB A to micro USB cord. Turn on the Wallaby and wait until it says connected. It should have a green signal icon in the bottom corner of the screen. Open a web browser and enter 192.168.124.1:8888. This will open the KISS IDE (Integrated development environment).

  -Wireless. Turn on the Wallaby and wait until it says connected. Connect to the Wallaby by finding its wi-fi network on your computer. Open a web browser and enter 192.168.125.1:8888. This will open the KISS IDE (Integrated development environment).

- **Create** Project.
  -Open Kiss IDE

  -Add a project to the Wallaby by clicking on Add Project at the bottom of the Project Explorer.

  -Name the Project *3DPrinter*.

- **Create** Include Files.
  -Add an include file to the *3DPrinter* project under Included Files in the Project Explorer.

  -Name this include file *3dPrinter*.

  -Add another include file to the *3DPrinter* project under Included Files in the Project Explorer.

  -Name this include file *gcode*.

- **Populate** Files.
  -Download the files from GitHub using the link in the Links to Supplementary Resources section.

  -Copy, Paste, and, Save the code into their respective files.

- **Rename** the G-code File.
  -Remember to change the name of the g-code file to whatever you saved it as.

- **Set** Open and Closed Servo Positions.
  -Remember to change both the closed and open positions of the servo.

Step 4. Confirm 3D Printer is Ready for Use

- **Check** Physical 3D Printer.
  -Confirm each axes moves away from the switch with a positive power output to the motors. And each of the axes moves freely.

  -Confirm all connections are tight.

  -Confirm all lever switches are working properly.

  -Confirm wiring is plugged in correctly and securely.

  -Confirm that all lever switches are correctly placed to stop the axes at the home position.

  - Level the print bed.

Step 5. Run 3D Printer

- **Press** Run on the Wallaby
  -Making sure that you have the 3DPrinter program selected in the GUI on the Wallaby, press Run in the top right hand of the screen.

  -The printer will first home itself on the home lever switches and then follow the slicer path generated by Cura.

- **Monitor** Printer During Print
  -To **pause** the 3D Printer, press the right button on the Wallaby in between g-code commands. To continue the print, press the right button again.

  -To **stop** the 3D Printer, press the left button in between g-code commands.
  In the case of an emergency, turnoff the Wallaby by the onoff switch on the side.

Step 6. Remove Part from Print Bed

- **Remove** the Part.
  -Use a 3D printer print release tool to pry the 3D print off of the print bed.

  -Replace any damaged or missing tape.

Three Axis Printer.
The 3D Pen Printer's extruding pen can move along 3 different axes. We will call the left right movement axis the X axis, the depth movement toward and away the Y axis, and the up down movement the Z axis.

- **X** Axis.
  -The X axis is controlled with only one motor and has a end stop lever switch for the home feature in the code.

  -The axis is geared to be 10 times faster than the motor rotation speed. This is done by using 3 axles and 2 different gearing combinations. The motor has a 40 tooth gear mounted to its face, turning at the exact speed of the motor. This gear is mated with an 8 tooth gear on an adjacent axle. This intermediate axle is spinning 5x the speed of the motor. The intermediate axle also has a 16 tooth gear mated with a 8 tooth gear on the axis that is connected to a threaded rod. This axis is spinning 2x faster than the intermediate axis, meaning that it spins 10x faster than the motor speed.

  -The threaded rod at the end of the gearing is connected to Extruder Assembly that is on a track in the direction of the axis. The threaded rod is connected to the assembly using a 8-32 nut, meaning 1 rotation of the motor creates a .3125 inch or 7.935 mm movement.

  -The axis is monitored by an encoder. The encoder was made using a break beam senor and a wheel with holes in it mounted on the intermediate axis. The encoder wheel has 6 holes in it resulting in 12 break beam changes a revolution. Being on intermediate axis, it spends at a rate half the speed of the threaded rod. This means the encoder has an accuracy of approximately .25 mm of linear motion of the Extruder Assembly.

- **Y** Axis.
  -The Y axis is also controlled with only one motor and has a end stop lever switch for the home feature in the code.

  -The Y axis is also geared to be 10 times faster than the motor rotation speed in the same way that the X axis is geared. This axis also has an encoder that is set up the exact same way as the X axis.

- **Z** Axis.
  -The Z axis is controlled with only two motors that move simultaneously and has a end stop lever switch for the home feature in the code.

  -The Z axis motors move together to lift the Up/Down Frame, which carries the X axis and Extruder Assembly. The motors are connected directly to the threaded rods meaning one rotation of the motor is .03125 inches or .79375 mm of linear movement.

  -Note: The are not encoders on the Z axis motors, this means that the rotation of the motors is estimated using the back emf position feature built into the Wallaby. The ticks per rotation values in the functions in the g-code include files might need to be changed for each motor.

AIO 3D Printing Pen.
The AIO 3D Printing Pen was chosen because of its reliability and relatively inexpensive cost. As a 3D printing pen it works phenomenally well, but as an extruder on this 3D Pen Printer it has some characteristics that if different would make the printer much more capable.

- **Operation**.
  -Plug the pen in to turn on.

  -Press the In button once to start the heating nozzle, wait for nozzle to reach operating temperature.

  -Feed Filament into the pen by double pressing the In button. Press the In button again to stop.

- **Settings** recommended while printing.
  -Speed 1

  -190 C Temperature

- **Qualities** to look for in a replacement pen.
  -One of the major problems with this pen is that it has retraction. When the In button is released the filament is retracted almost a centimeter into the pen. This causes a pause between moves without extrusion to extrusion.

  -Another of the major problems with this pen is its flow-rate. Even at a speed of 1, the pen isn't moving quite fast enough for the flow rate.

Main File Configuration (main.c).
The main.c file contains all of the output and input port configurations, contains the open and closed servo positions, begins multiple processes to monitor the x and y axis rotation, creates shared memory on the Wallaby for these processes to communicate, opens and initializes g-code file, and of course has the main g-code processing and executing loop.

- **Output and Input** Port Configuration.
  -In the figure below, you can see that this is where all of the ports are configured for the Wallaby. If the wiring diagram in the Printer Assembly section has been used correctly, there should be no reason to change the port configurations.

  -If you decide you want to change which port on the wallaby is connected to which component this is where you would make these changes.

  -The open and closed positions for the servo pressing the extrude button on the 3D pen can be found on lines (89-92) at the bottom of the figure. Open position is the button not being pressed, while closed position is the button being pressed.

```
59    /* X-Axis Configuration */
60    X_AXIS.MOTOR_PORT = 0;
61    X_AXIS.SWITCH_PORT = 0;
62    X_AXIS.ENCODER_PORT = 8;
63    X_AXIS.previous = &previous_x;
64    X_AXIS.global_position = &Global_X;
65    X_AXIS.encoder_counter = 0;
66    X_AXIS.encoder_position = 0;
67
68    /* Y-Axis Configuration */
69    Y_AXIS.MOTOR_PORT = 1;
70    Y_AXIS.SWITCH_PORT = 1;
71    Y_AXIS.ENCODER_PORT = 9;
72    Y_AXIS.previous = &previous_y;
73    Y_AXIS.global_position = &Global_Y;
74    Y_AXIS.encoder_counter = 0;
75    Y_AXIS.encoder_position = 0;
76
77    /* Z1-Axis Configuration */
78    Z1_AXIS.MOTOR_PORT = 2;
79    Z1_AXIS.SWITCH_PORT = 2;
80    Z1_AXIS.previous = &previous_z1;
81    Z1_AXIS.global_position = &Global_Z1;
82
83    /* Z2-Axis Configuration */
84    Z2_AXIS.MOTOR_PORT = 3;
85    Z2_AXIS.SWITCH_PORT = 3;
86    Z2_AXIS.previous = &previous_z2;
87    Z2_AXIS.global_position = &Global_Z2;
88
89    /* Extruder Configuration */
90    PEN_EXTRUDER.SERVO_PORT = 2;
91    PEN_EXTRUDER.SERVO_POSITION_OPEN = 1660;
92    PEN_EXTRUDER.SERVO_POSITION_CLOSED = 1770;
```

- **G-Code** File Configuration.
  -In the figure below you can see the g-code file being opened and the pointer in that file being initialized.

  -It is important the file_name is named correctly. "TestBlock.gcode" should be renamed to whatever you have named your g-code file.

```
131
132     //*********Setup G-Code File Read**********//
133     char file_name[30] = "TestBlock.gcode"; // **************!!!__File Name To Change__!!!
134     char file_location[50] = "/media/usb/"; // file location
135     strcat(file_location,file_name); // strcat combines the two strings
136     mount_USB(); // Mounts Inserted USB Stick
137     FILE * fp; // Intitalize file pointer
138     fp = fopen(file_location,"r"); // Open G-Code file
139     char * line; // Initialize variable for line of G-Code
140
```

Include Files Configuration (3dPrinter.h, gcode.h).
3dPrinter.h contains all of the functions of the 3D Printer except what the g-code actually does. gcode.h contains the functions that directly reflect the g-code command.

- **3dPrinter.h**
  - The function execute_command (lines 10-66) determines which g-code function from gcode.h to execute, looking at the letter (G, M, T) and the following number(s).

  - The function get_gcode (lines 70-87) takes the file pointer that is in the open g-code file, and returns the line of g-code that the pointer is at. If the pointer is at the end of the file, the function returns "End".

  - The function process_gcode (lines 91-122) gets the letter and number associated to the g-code. Then passes it to execute_gcode.

  - The function mount_USB (lines 126-148) creates a path to the USB.

  - The function unmount_USB (lines 152-154) deletes the path to the USB.

  - The function start_motor_start_encoder_counter (lines 158-172) creates a new process that counts the number of times the signal has changed from the break beam sensors.

  - The function start_motor_start_encoder_position (lines 175-205) uses the encoder counter and the motor rotation direction to create a position based off of the encoder.

  - The function move_encoder_relative_position (lines 208-224) moves a motor a specific amount of encoder ticks.

  - The function move_encoder_to_position (lines 227-243) takes the global encoder position and moves the motor to a desired target position.

- **gcode.h**
  - The function gcode_G0 (lines 27-249) and gcode_G1 (lines 251-472) are both movement commands. G0 is the command to move without extrusion. G1 is the command to move with extrusion.

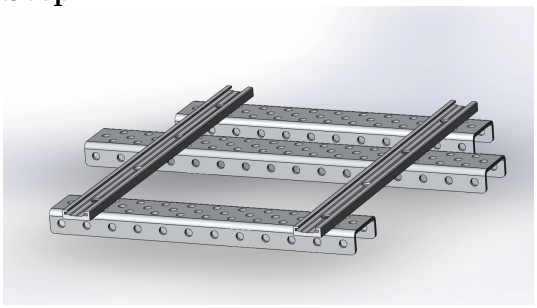
  - The function gcode_G28 (lines 474-545) homes the 3D printer. The X, Y, and Z axes move in the negative direction until they push the end stop lever sensor.

PRINTER ASSEMBLY

This printer assembly is accompanied with an assembly animation video found in the Additional Links section.

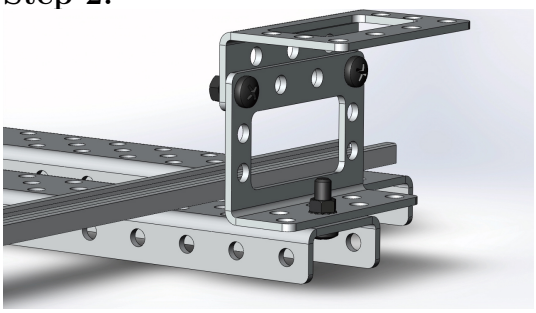
Printer Base/Main Assembly.
The Printer Base is the base for the printer. From here after every assembly it gets added to the main assembly starting with this base. During the printer base assembly it is important to leave the linear tracks loose, because when the print bed is installed you can make sure that the print bed moves easily along the track and forward and backward at a right angle to the 3x16 Channels.

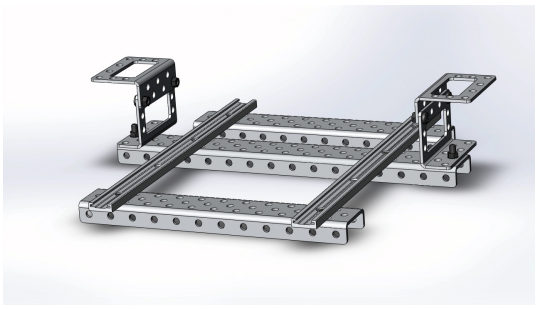
- **Step 1.**



-3x16 Channel x1
-3x11 Channel x2
-Linear Slide x2
-M3 Screw 5mm x6
-M3 Nut x6

- **Step 2.**





-Servo Bracket x4
-8-32 1/4” Screw x4
-8-32 1/2” Screw x2
-8-32 KEPS Nut x6

- **Finished.**

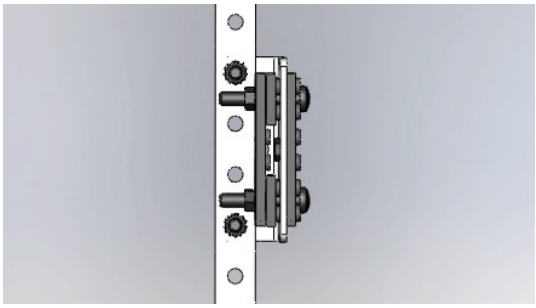
Print Bed Assembly.
It's important to leave the 8-32 nut suspended between the Lego pieces loose and free to move until the threaded rod axis is installed. This will insure that it moves freely.

- **Step 1 & 2.**

-1x19 Strap x2
-1x10 Strap x2
-8-32 1/4" x4
-8-32 KEPS Nut x4

-M3 Screw 5mm x4
-Linear Bearing x4
-2x8 Tile w/Holes x2

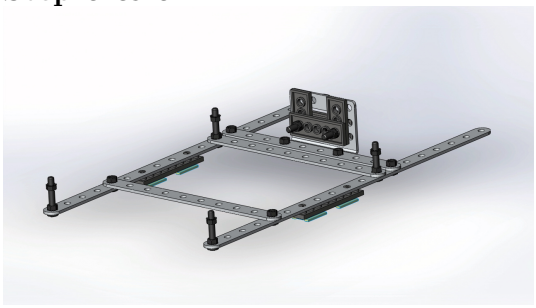
- **Step 3 & 4.**

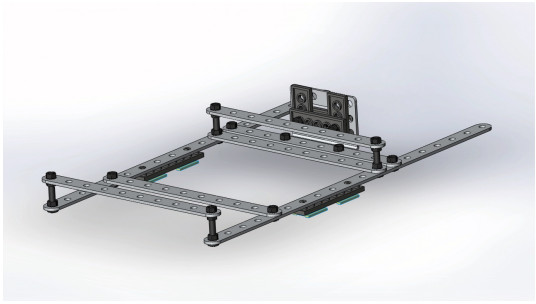


-Servo Bracket x1
-8-32 1/4" x2
-8-32 KEPS Nut x2

-8-32 1" x2
-8-32 KEPS Nut x3
-2x4 Tile w/Holes x2
-2x6 Tile w/Holes x2

- **Step 5 & 6.**

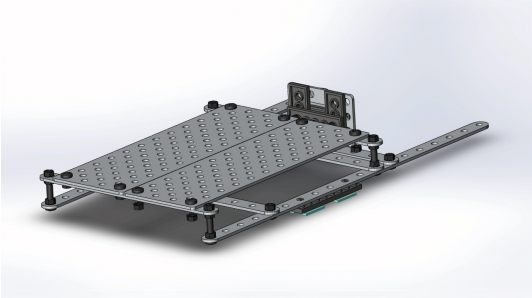
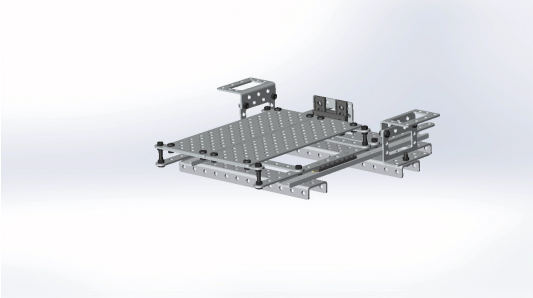




-1x10 Strap x1
-8-32 1" x4
-8-32 KEPS Nut x8

-1x10 Strap x2
-8-32 KEPS Nut x4

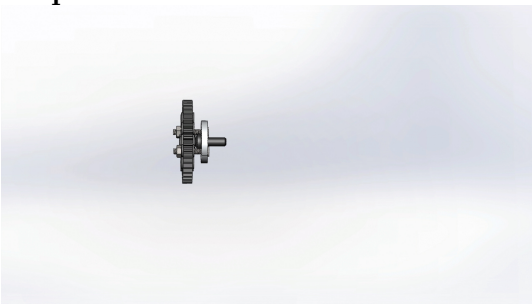
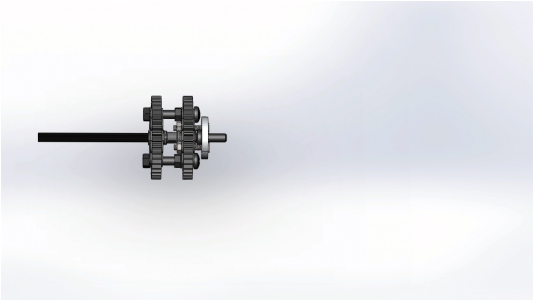
- **Step 7 & Add** Print Bed to Main Assembly

-4x12 Plate x2
-8-32 1/4" Screw x8
-8-32 KEPS Nut x8

X or Y Axis Assembly.
The X and Y threaded rod axis are the exact same parts and assembly.

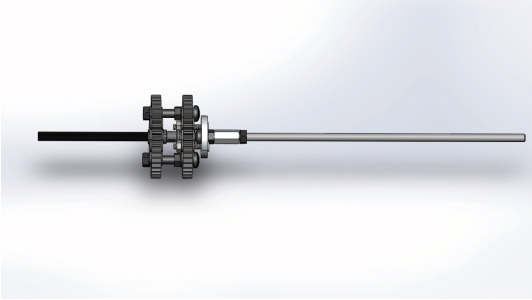
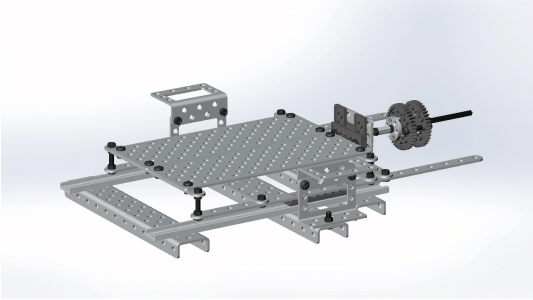
- **Step 1 & 2.**

-M3 Nut x2
-40 Tooth Gear x1
-M3 Screw 15mm x2
-Round Metal Servo Horn x1
-8-32 1/2" x1

-8-32 KEPS Nut x4
-40 Tooth Gear x1
-Axle Bushing x1
-8-32 1" Screw x2
-Axle 80mm x1

- **Step 3 & Add** Axis to Assembly

-8-32 6" All Thread x1
-8-32 1/2" Standoff x1
-8-32 KEPS Nut x2

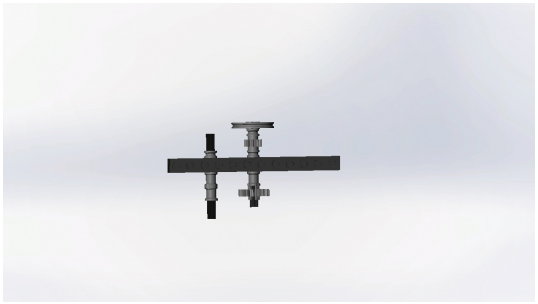
Y Axis Motor and Gear Assembly.
The y axis motor assembly is shown first, and then the gear assembly is shown with the addition of the motor assembly part way through.

- **Step 1 & 2.**

-Silver Large Head Screw x1
-40 Tooth Gear x1
-Plus White Servo Horn x2
-Sliver Servo Screw x2
-Black Motor x1

-8-32 KEPS Nut x3
-Servo Bracket x1
-8-32 1/2" Screw x1
-8-32 1/4" Screw x2

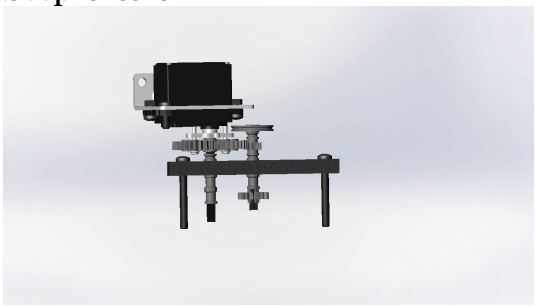
- **Step 3 & 4**

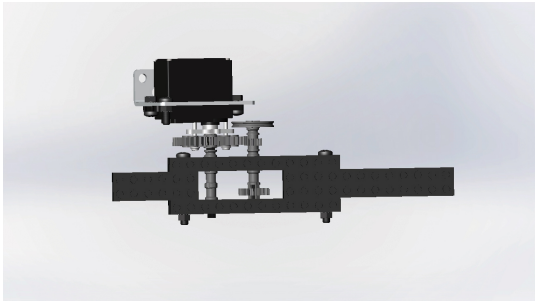


-Axle 48mm x1
-1x12 Brick w/ Holes x2
-8 Tooth Gear x1
-16 Tooth Gear x1
-1/2 Axle Bushing x2
-Axle Bushing x1
-Pully/Encoder Wheel x1

-Axle 48mm x1
-1/2 Axle Bushing x1
-Axle Bushing x2

- **Step 5 & 6**

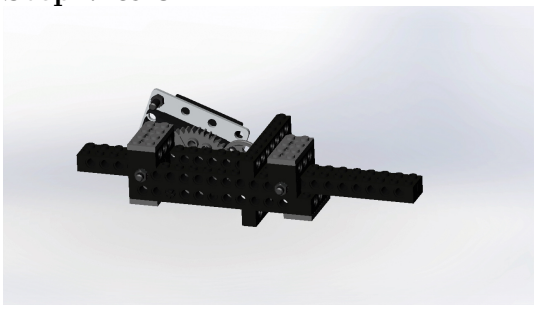


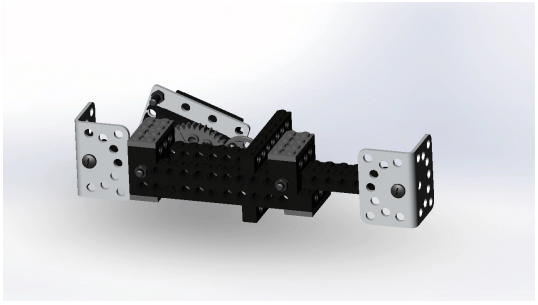


-8-32 1 1/2" Screw x2
-Motor Assembly x1

-8-32 KEPS Nut x2
-1x12 Brick w/ Holes x4
-1x6 Brick w/ Holes x2

- **Step 7 & 8**

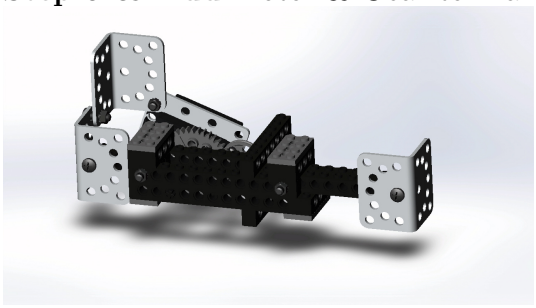


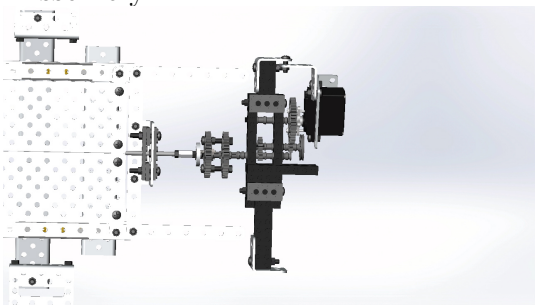


-1x4 Brick w/ Holes x6
-2x4 Tile w/ Holes x4
-1x2 Brick w/ Holes x2
-1x8 Brick w/ Holes x2

-L Bracket x2
-8-32 1" Screw x2
-8-32 KEPS Nut x2

- **Step 9 & Add** Motor & Gear to Main Assembly





-L Bracket x1
-8-32 1/2" Screw x1
-8-32 KEPS Nut x2

-Axle Bushing x2
-1/2 Axle Bushing x4
-8 Tooth Gear x1

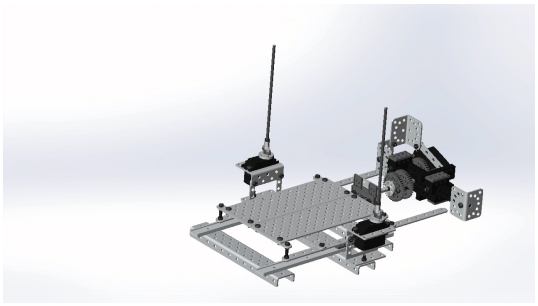
Z Axes Assembly.
There are two Z Axes and the assembly procedure is the smae for both of them.

- **Step 1 & 2.**

-M3 Screw 15mm x2
-Round Metal Servo Horn x1
-Large Head Servo Screw x1
-Black Motor x1

-1x3 Thin Lift Arm x1
-8/32 1/4" Screw x1
-8/32 1/2" Standoff x1
-M3 Nut x2

- **Step 3 & Add** Axis to Assembly

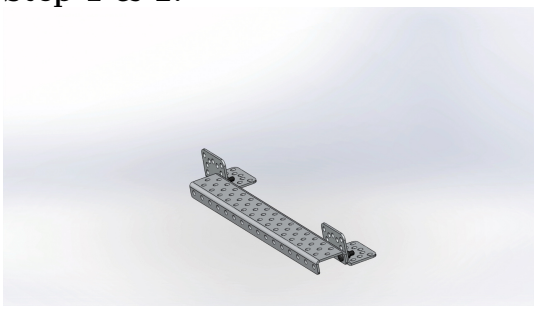


-8-32 6" All Thread x1
-8-32 KEPS Nut x2

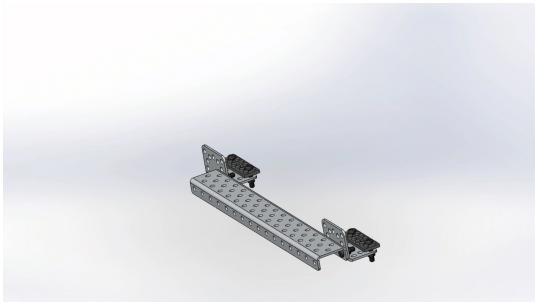
-8-32 KEPS Nut x2
-8-32 1/2" Screw x2

Up/Down Frame Assembly.
The Up/Down Frame runs on the Z axes. The nuts connecting the frame to the Main Assembly need to be loose until the threaded rod is inserted.

- **Step 1 & 2.**

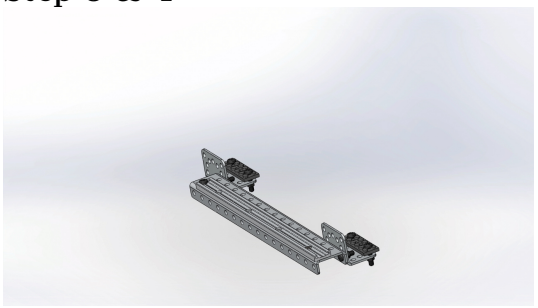


-8-32 KEPS Nut x3
-8-32 1/2" Screw x3
-L Bracket x2
-3x16 Channel x1

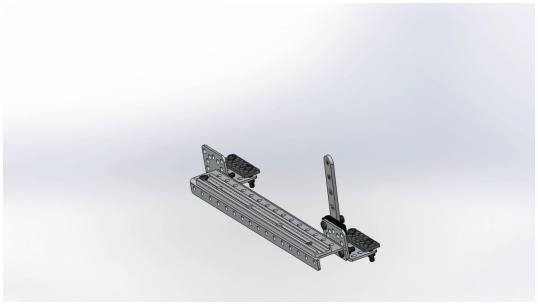


-8-32 KEPS Nut x6
-2x6 Tile w/ Holes x2
-8-32 3/4" Screw x4

- **Step 3 & 4**

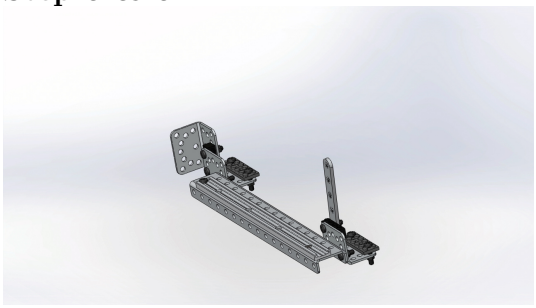


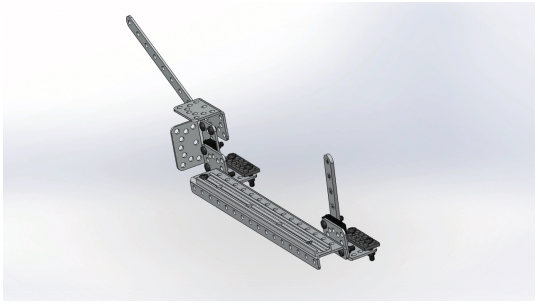
-Linear Slide Track x1
-8-32 1/2" Screw x1
-8-32 KEPS Nut x1
-M3 15mm Screw x1
-M3 Nut x1



-1x5 Metal Servo Arm x1
-8-32 3/4" Screw x1
-8-32 KEPS Nut x1
-1x4 Brick w/ Holes x1

- **Step 5 & 6**

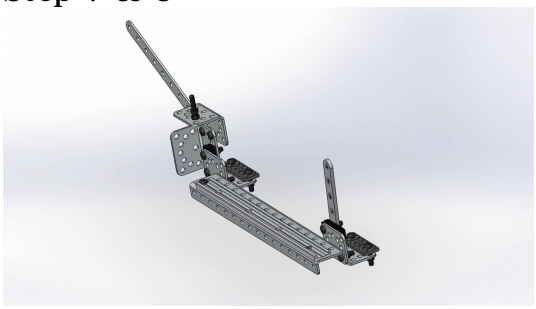


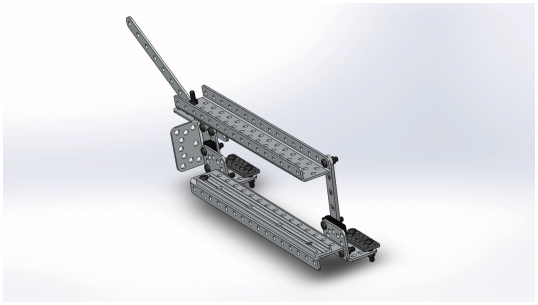


-8-32 3/4" Screw x2
-8-32 KEPS Nut x2
-1x4 Brick w/ Holes x1

-8-32 1/2" Screw x2
-8-32 KEPS Nut x2
-L Bracket x1
-1x10 Strap x1

- **Step 7 & 8**

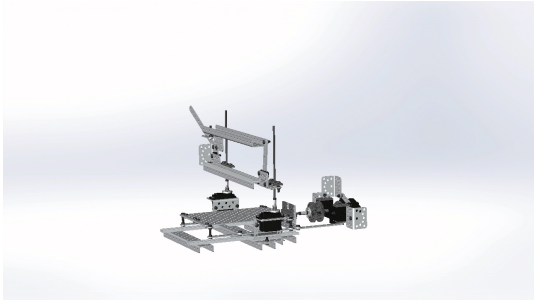




-8-32 1" Screw x1
-8-32 KEPS Nut x2

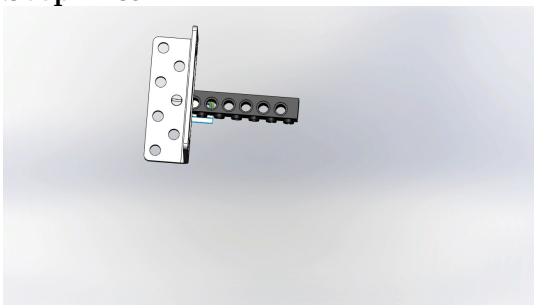
-8-32 1/2" Screw x1
-8-32 KEPS Nut x2
-3x16 Channel x1

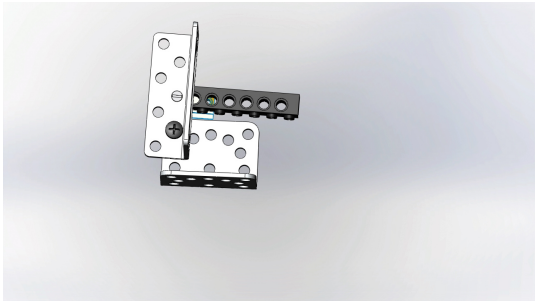
- **Add** Up/Down Frame to Main Assembly



Extrusion Frame Assembly.
The y axis motor assembly is shown first, and then the gear assembly is shown with the addition of the motor assembly part way through.

- **Step 1 & 2.**

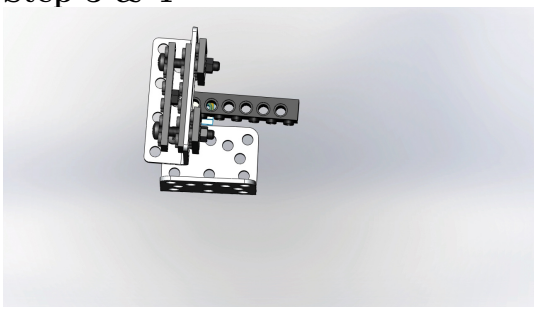


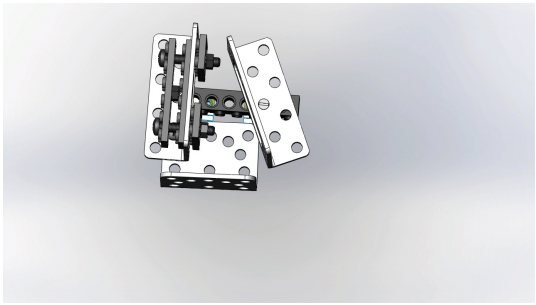


-M3 10mm Screw x1
-Linear Slide Bearing x1
-1x8 Brick w/ Holes x1
-L Bracket x1

-L Bracket x1
-8-32 KEPS Nut x1
-8-32 1/2" Screw x1

- **Step 3 & 4**

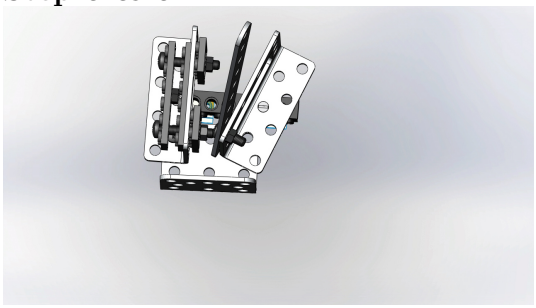


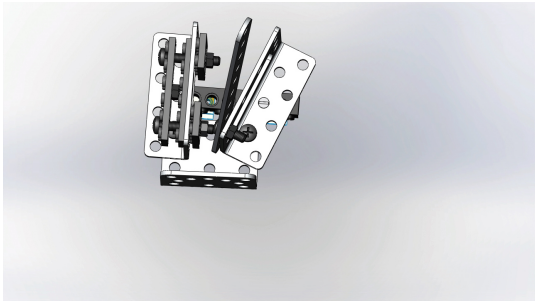


-2x4 Tile w/Holes x2
-2x6 Tile w/ Holes x2
-8-32 1" Screw x2
-8-32 KEPS Nut x3

-M3 10mm Screw x1
-Linear Slide Bearing x1
-Servo Bracket x1

- **Step 5 & 6**

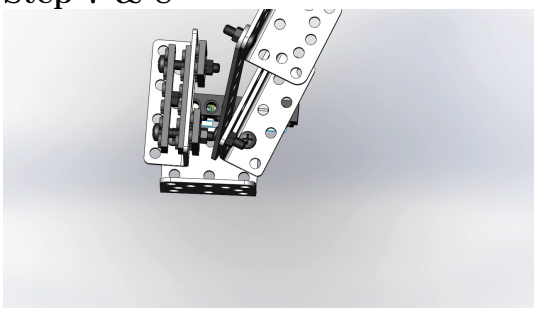


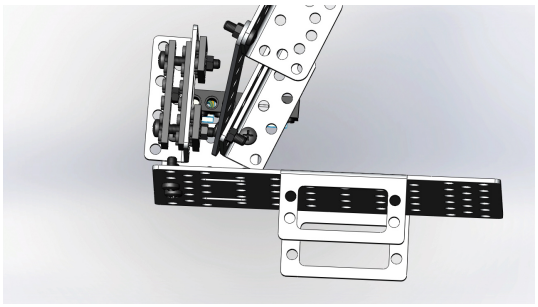


-8-32 1/2" Screw x1
-8-32 KEPS Nut x1
-1x10 Strap x1

-8-32 KEPS Nut x1
-8-32 1/2" Screw x1

- **Step 7 & 8**

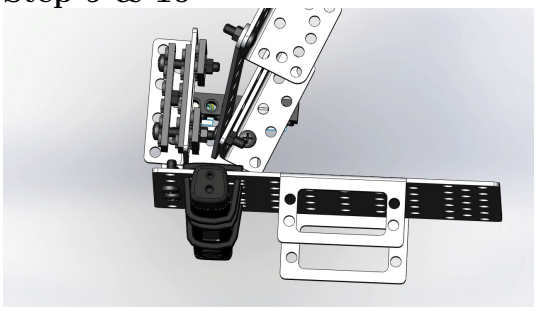


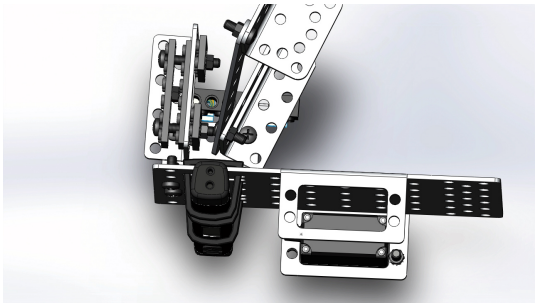


-L Bracket x1
-8-32 1/2” Screw x1
-8-32 KEPS Nut x1

-Wallaby Chassis x1
-8-32 1/2” Screw x2
-8-32 KEPS Nut x2

- **Step 9 & 10**

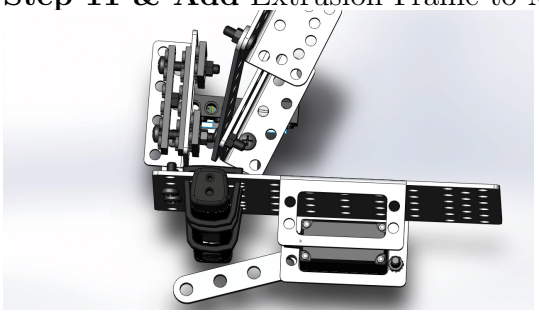


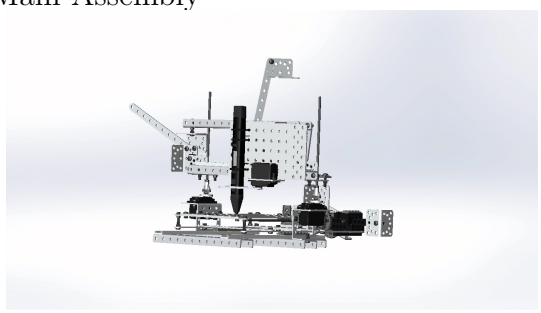


-AIO 3D Printer Pen x1
-Ziptie 6” x3
-8-32 KEPS Nut x2

-Black Motor x1
-8-32 1/2” Screw x2
-8-32 KEPS Nut x2

- **Step 11 & Add** Extrusion Frame to Main Assembly





-Large Head Servo Screw x1
-1x5 Metal Servo Arm x1

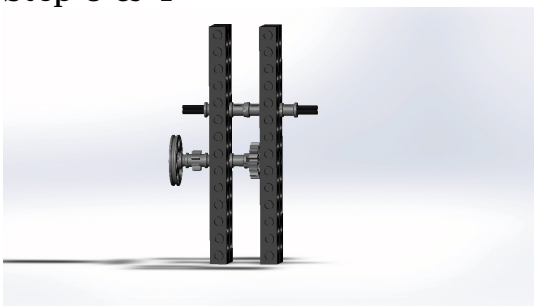
X Axis Motor and Gear Assembly.
The x axis motor assembly is shown first, and then the gear assembly is shown with the addition of the motor assembly part way through.

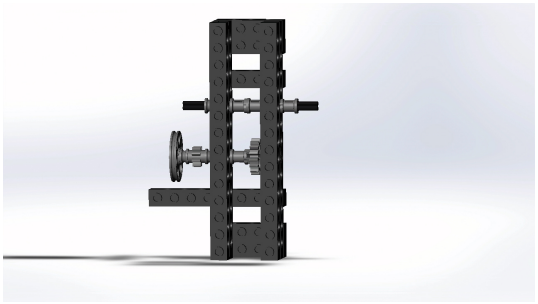
- **Step 1 & 2.**

-Silver Large Head Screw x1
-40 Tooth Gear x1
-Plus White Servo Horn x2
-Sliver Servo Screw x2
-Black Motor x1

-8-32 KEPS Nut x3
-Servo Bracket x1
-8-32 1/2" Screw x1
-8-32 1/4" Screw x2

- **Step 3 & 4**

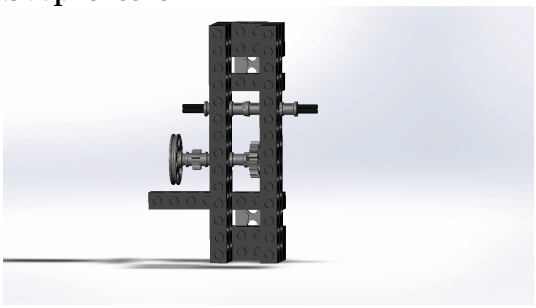


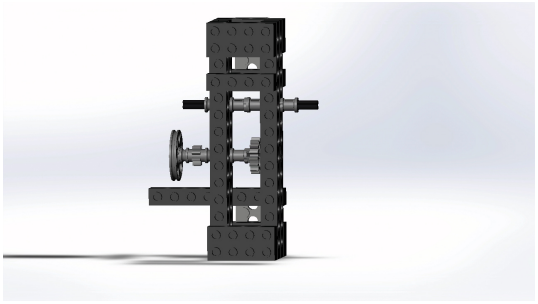


-Axle 48mm x1
-1x12 Brick w/ Holes x4
-8 Tooth Gear x1
-16 Tooth Gear x1
-1/2 Axle Bushing x3
-Axle Bushing x4
-Pully/Encoder Wheel x1 -Axle 64mm x1

-1x4 Brick w/ Holes x5
-1x8 Brick w/ Holes x1

- **Step 5 & 6**

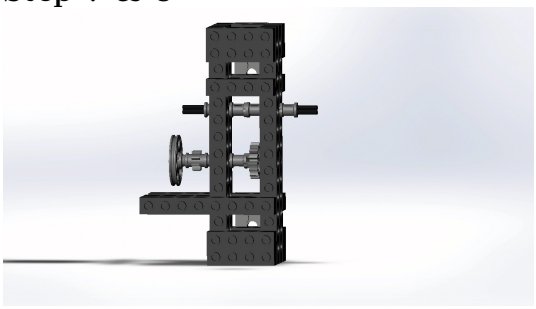


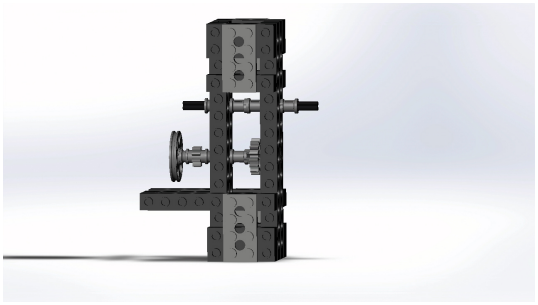


-2x6 Tiles w/ Holes x2

1x4 Brick w/ Holes x5

- **Step 7 & 8**

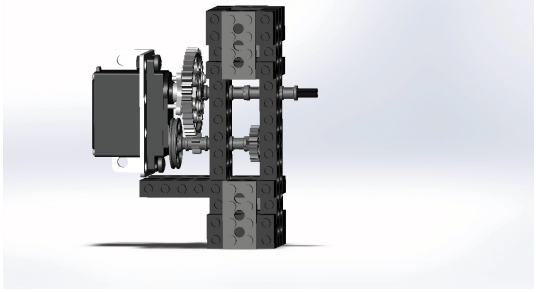


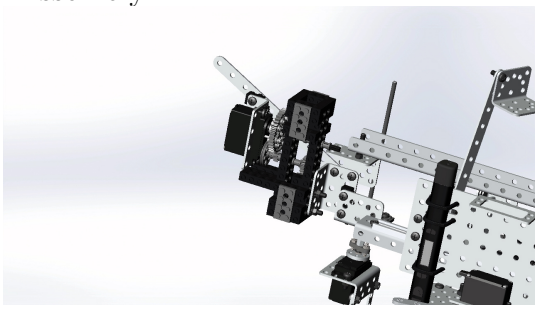


-1x8 Brick w/ Holes x1
-1x4 Brick w/ Holes x1

-2x6 Tile w/ Holes x2

- **Step 9 & Add** Motor & Gear to Main Assembly



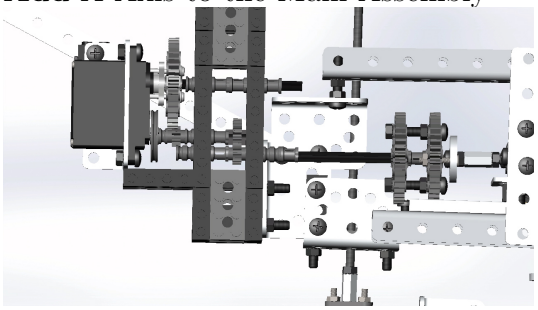


-Add Motor to Assembly

-1x2 Brick w/ Holes x1
-8-32 3/4" Screw x1
-8-32 1/2" Screw x2
-8-32 KEPS Nut x3

Add X Axis to the Main Assembly.
Add the X Axis (which build procedure is above) to the Main Assembly.

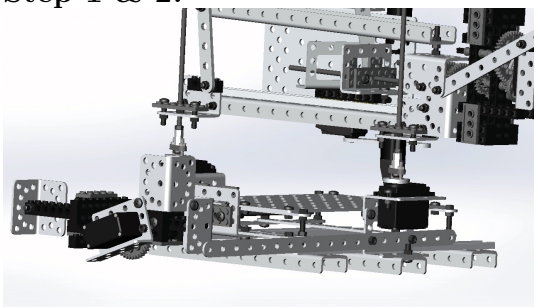
- **Add** X Axis to the Main Assembly



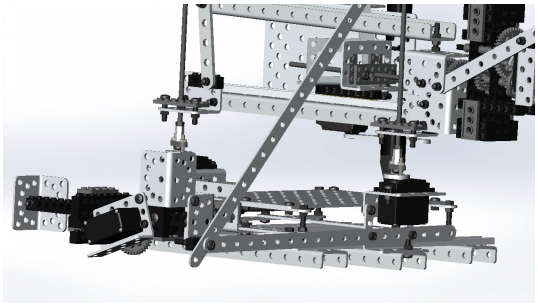
-8 Tooth Gear x1
-1/2 Axle Bushing x4
-Axle Bushing x2

Bracing the Main Assembly.
Add Bracing to the Main Assembly.

- **Step 1 & 2.**

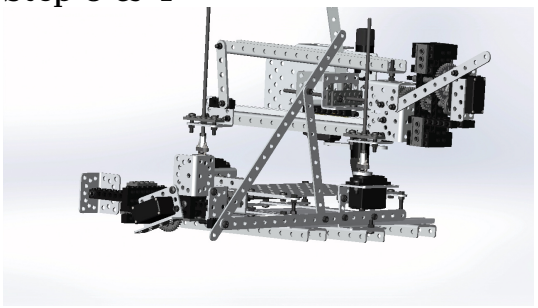


-8-32 KEPS Nut x2
-8-32 1/4" Screw x2
-1x19 Strap x1

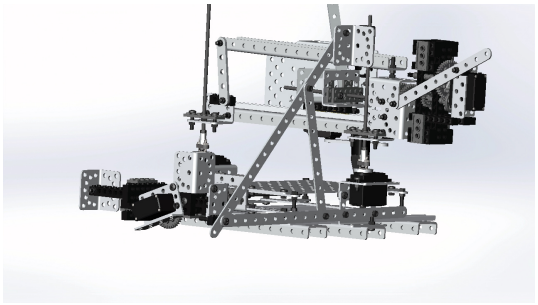


-8-32 KEPS Nut x1
-8-32 1 1/2" Screw x1
-1x19 Strap x1
-1x2 Brick w/ Holes x1

- **Step 3 & 4**

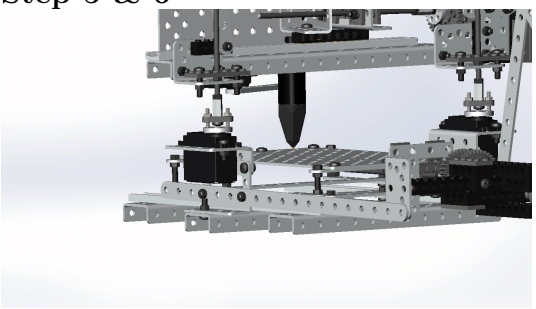


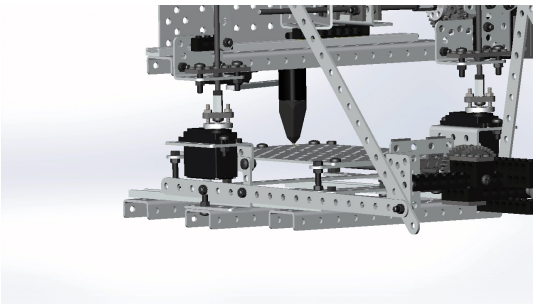
-8-32 KEPS Nut x2
-8-32 1/4" Screw x2
-1x10 Strap x1



-8-32 KEPS Nut x1
-8-32 1/4" Screw x1
-L Bracket x1

- **Step 5 & 6**

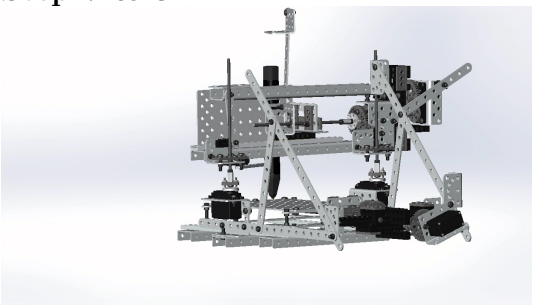


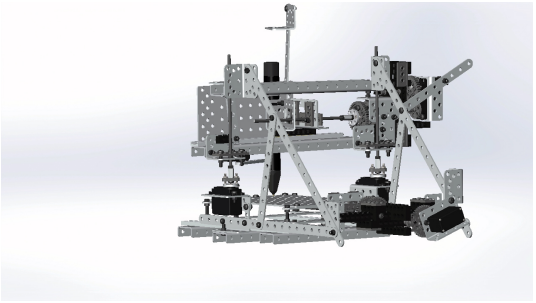


-8-32 KEPS Nut x2
-8-32 1/4” Screw x2
-1x19 Strap x1

-8-32 KEPS Nut x1
-8-32 1/4” Screw x1
-1x19 Strap x1

- **Step 7 & 8**

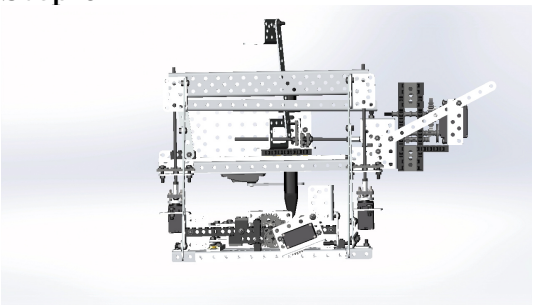




-8-32 KEPS Nut x2
-8-32 1/4” Screw x2
-1x10 Strap x1

-8-32 KEPS Nut x1
-8-32 1/4” Screw x1
-L Bracket x1

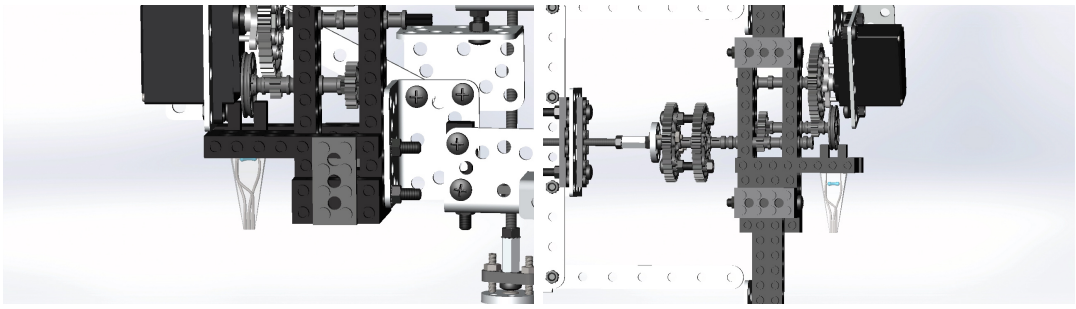
- **Step 9**



-3x16 Channel x1
-8-32 KEPS Nut x2
-8-32 1/4” Screw x2

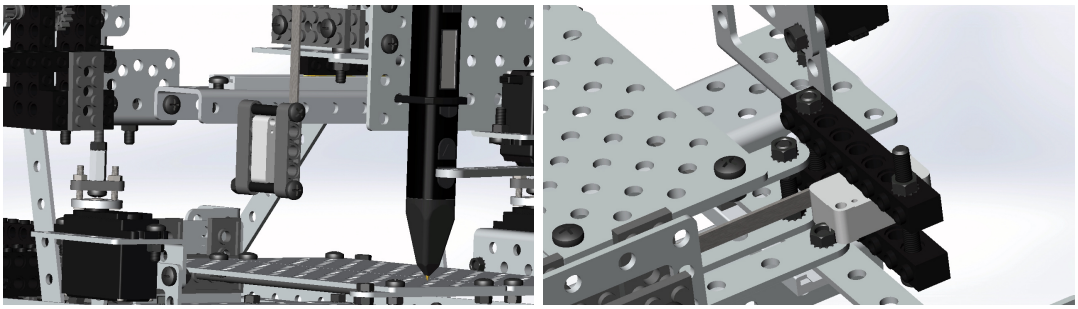
Add Break Beam Sensors to the Main Assembly.
Add the Break Beam Sensors to the Main Assembly.

- **X** & **Y** Axes



Add Endstop Switches to the Main Assembly.
The Endstop Switches are made to be adjustable. Position the Switch in a way that it will be triggered in the home position of the printer.

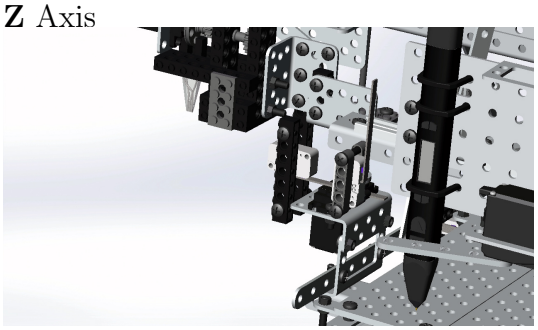
- **X** & **Y** Axes



-8-32 KEPS Nut x2
-8-32 1 1/2" Screw x2
-1x5 Lift Arm x1
-Endstop Lever Switch

-8-32 KEPS Nut x4
-8-32 1 1/2" Screw x2
-1x8 Brick w/ Holes x2
-Endstop Lever Switch

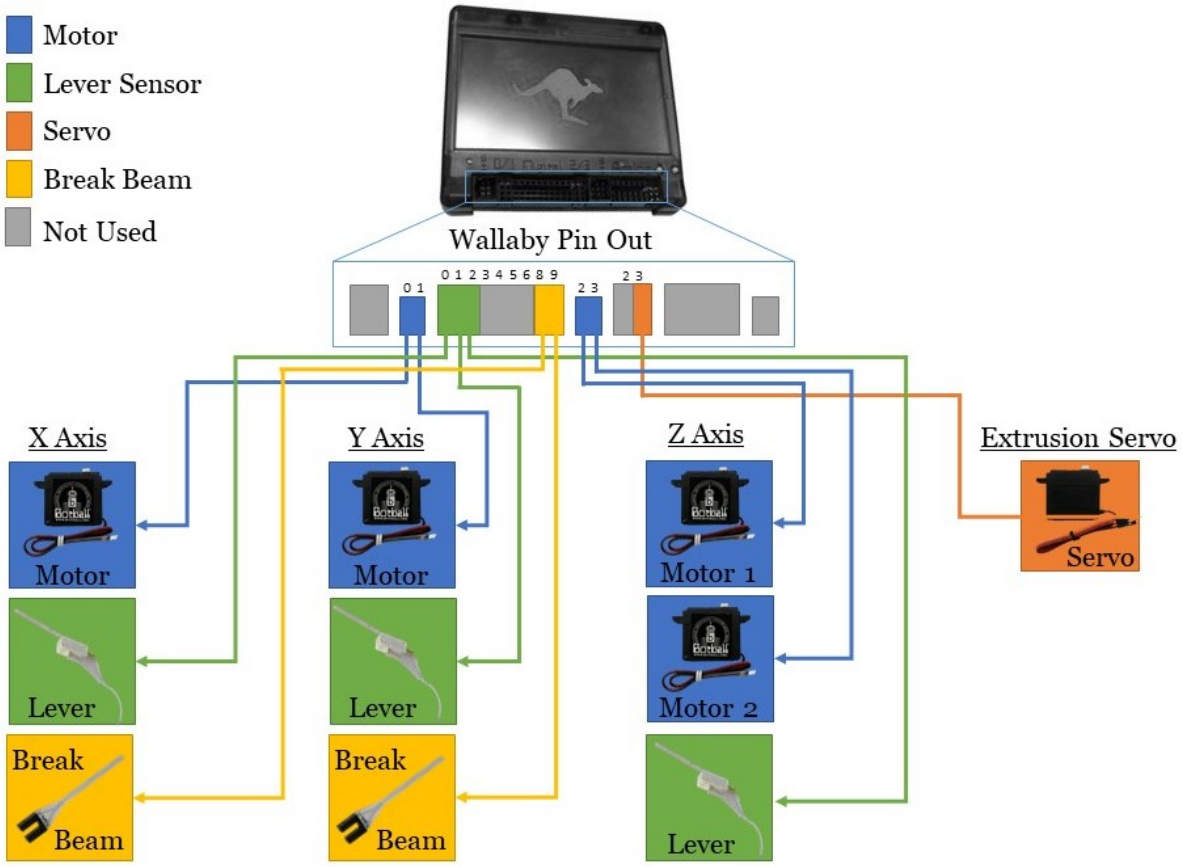
- **Z** Axis



-8-32 1 1/2" Screw x2
-8-32 1/2" Screw x2
-8-32 KEPS Nut x2

-1x12 Brick w/ Holes x1
-1x8 Brick w/ Holes x1
-Endstop Lever Switch

WIRE DIAGRAM

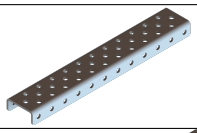
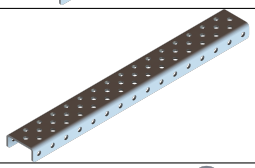
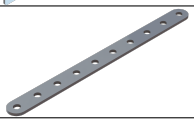
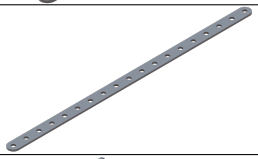
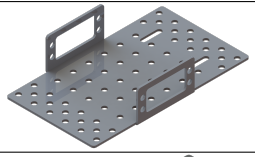
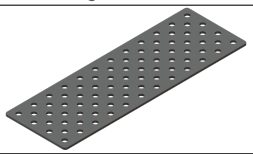
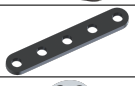


PARTS REQUIRED

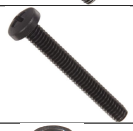
AIO 3D Printer Pen

Part	Picture	Quantity
AIO 3D Printer Pen Pen,USB Outlet,Charging Cord,Filament		1

Metal Parts

Part	Picture	Quantity
3x11 Channel		2
3x16 Channel		4
1x10 Strap		9
1x19 Strap		6
Wallaby Chassis		1
4x12 Plate		2
L Bracket		11
Servo Bracket		9
1x5 Servo Horn		2
Round Servo Horn		4

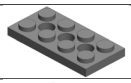
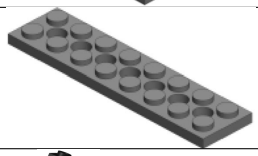
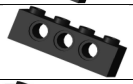
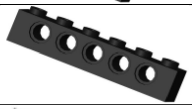
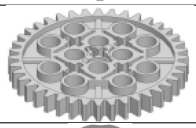
Screws/Nuts/Threaded Rod/Standoffs

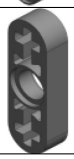
Part	Picture	Quantity
Screw 8-32 0.25in		34
Screw 8-32 0.50in		32
Screw 8-32 0.75in		8
Screw 8-32 1.00in		15
Screw 8-32 1.50in		9
Nut 8-32 Keps		121
Threaded Rod 8-32 6.00in		4
Standoff 8-32 0.50in		4
Screw M3 5mm		10
Screw M3 10mm		2
Screw M3 15mm		9
Nut M3		15
Silver Servo Screw		4
Silver Servo Screw Large Head		4

Electronics

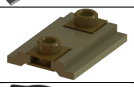
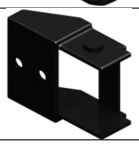
Part	Picture	Quantity
KIPR Wallaby		1
Wallaby Battery Charger		1
Wallaby Battery		1
USB A to Micro USB Cable		1
KIPR Motor		4
KIPR Servo		1
Plus White Servo Horn		2
Break Beam Sensor		2
Lever Sensor		3
Sensor Extension Cable		3
Servo Extension Cable		3

Lego Parts

Part	Picture	Quantity
Axle 48mm 3706		3
Axle 64mm 3707		1
Axle 80mm 3737		2
2x4 Tile w/ Holes 3709b		12
2x6 Tile w/ Holes 32001		6
2x8 Tile w/ Holes 3738		2
1x2 Brick w/ Holes 3700		4
1x4 Brick w/ Holes 3701		19
1x6 Brick w/ Holes 3894		2
1x8 Brick w/ Holes 3702		8
1x12 Brick w/ Holes 3895		7
1x14 Brick w/ Holes 32018		4
8 Tooth Gear 3647		4
16 Tooth Gear 4019		2
40 Tooth Gear 3649		6
Axle Bushing 3713		13
1/2 Axle Bushing 4265c		14

Pully/Encoder Wheel 4185		2
1x5 Lift Arm 32316		2
1x3 Lift Arm Thin 6632		2

IGUS Parts

Part	Picture	Quantity
Linear Slide Track		3
Linear Bearing		6
Wire Chain Link		24
Wire Chain End		2

Miscellaneous Parts

Part	Picture	Quantity
Roll of Masking Tape		1
Zip-Tie		5
3-in-1 Oil		1

REFERENCES

[1] *What Material Should I Use For 3D Printing?*. (English) PLA(Poly Lactic Acid), Feb 2013. http://3dprintingforbeginners.com/filamentprimer/

[2] *What Material Should I Use For 3D Printing?*. (English) ABS (Acrylonitrile Butadiene Styrene), Feb 2013. http://3dprintingforbeginners.com/filamentprimer/

[3] Gambody. *STL and OBJ Files 101: Exporting, Viewing and Repairing Guide*. (English), June 2016. https://3dprintingforbeginners.com/stl-and-obj-files-101/

[4] Jack Davies. *3D Modeling CAD Software*. (English) 3dhubs, 2017. https://www.3dhubs.com/knowledge-base/3d-modeling-cad-software

[5] Robin Brockotter. *Key design considerations for 3D Printing*. (English) 3dhubs, 2017. https://www.3dhubs.com/knowledge-base/key-design-considerations-3d-printing...

APPENDIX

main.c

```
// 3D Printer Code for KIPR Wallaby //
//

// Designed and coded by Riley Cotter
// Under the supervision of Dr. David Miller
// Aerospace and Mechanical Engineering
// Gallogly College of Engineeing
// University of Oklahoma

struct AXIS {
    int MOTOR_PORT; // Motor Port on KIPR Wallaby connected to Axis Motor
    int SWITCH_PORT; // Digital Port on KIPR Wallaby connected to Home Zeroing Switch
    int ENCODER_PORT; // Digital Port on KIPR Wallaby connected to Break-Beam Sensor
    float* previous; // Previous G-Code Axis Value
    float* global_position; // Global Position
    int counter_memory_id; // Memory Segment ID for Shared Memory (encoder counter)
    long* encoder_counter; // Shared Memory for communication between child process to parent.
    int position_memory_id; // Memory Segment ID for Shared Memory (encoder counter)
    int* encoder_position; // Shared Memory for communication between child process to parent.
    int counter_process_memory_id; // Memory Segment ID for Shared Memory (child process id)
    int* counter_child_process; // Shared Memory for communication between child process to parent.
    int position_process_memory_id; // Memory Segment ID for Shared Memory (child process id)
    int* position_child_process; // Shared Memory for communication between child process to parent.
} AXIS;

struct EXTRUDER {
    int SERVO_PORT; // Servo Port on KIPR Wallaby that pushes the feed button on the pen.
    int SERVO_POSITION_OPEN; // Servo Position when NOT pushing the button on the pen.
    int SERVO_POSITION_CLOSED; // Servo Position when pushing the button on the pen.
} EXTRUDER;

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <kipr/botball.h>
#include <signal.h>
#include <3dPrinter.h>

int main()
{
    // Declare variables for struct pointers //
    float previous_x = 0; // Previous gcode X coordinate (milimeters)
    float previous_y = 0; // Previous gcode Y coordinate (milimeters)
    float previous_z1 = 0; // Previous gcode Z coordinate (milimeters)
    float previous_z2 = 0; // Previous gcode Z coordinate (milimeters)
    float Global_X = 0; // Previous global ("where the program thinks the extrusion tip should be") X coordinate (encoder ticks)
    float Global_Y = 0; // Previous global ("where the program thinks the extrusion tip should be") Y coordinate (encoder ticks)
    float Global_Z1 = 0; // Previous global ("where the program thinks the extrusion tip should be") Z1 coordinate (encoder ticks)
    float Global_Z2 = 0; // Previous global ("where the program thinks the extrusion tip should be") Z2 coordinate (encoder ticks)
```

```
    struct AXIS X_AXIS; // Declaring X-Axis as a Axis Structure
    struct AXIS Y_AXIS; // Declaring Y-Axis as a Axis Structure
    struct AXIS Z1_AXIS; // Declaring Z1-Axis as a Axis Structure
    struct AXIS Z2_AXIS; // Declaring Z2-Axis as a Axis Structure
    struct EXTRUDER PEN_EXTRUDER; // Declaring Extruder as a Extruder Structure

    /* X-Axis Configuration */
    X_AXIS.MOTOR_PORT = 0;
    X_AXIS.SWITCH_PORT = 0;
    X_AXIS.ENCODER_PORT = 8;
    X_AXIS.previous = &previous_x;
    X_AXIS.global_position = &Global_X;
    X_AXIS.encoder_counter = 0;
    X_AXIS.encoder_position = 0;

    /* Y-Axis Configuration */
    Y_AXIS.MOTOR_PORT = 1;
    Y_AXIS.SWITCH_PORT = 1;
    Y_AXIS.ENCODER_PORT = 9;
    Y_AXIS.previous = &previous_y;
    Y_AXIS.global_position = &Global_Y;
    Y_AXIS.encoder_counter = 0;
    Y_AXIS.encoder_position = 0;

    /* Z1-Axis Configuration */
    Z1_AXIS.MOTOR_PORT = 2;
    Z1_AXIS.SWITCH_PORT = 2;
    Z1_AXIS.previous = &previous_z1;
    Z1_AXIS.global_position = &Global_Z1;

    /* Z2-Axis Configuration */
    Z2_AXIS.MOTOR_PORT = 3;
    Z2_AXIS.SWITCH_PORT = 3;
    Z2_AXIS.previous = &previous_z2;
    Z2_AXIS.global_position = &Global_Z2;

    /* Extruder Configuration */
    PEN_EXTRUDER.SERVO_PORT = 2;
    PEN_EXTRUDER.SERVO_POSITION_OPEN = 1660;
    PEN_EXTRUDER.SERVO_POSITION_CLOSED = 1770;

    int shared_segment_size = 0x6400;

    /* Allocate a shared memory segment for each shared memory variable.  */
    X_AXIS.counter_memory_id = shmget (IPC_PRIVATE, shared_segment_size, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    Y_AXIS.counter_memory_id = shmget (IPC_PRIVATE, shared_segment_size, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    X_AXIS.position_memory_id = shmget (IPC_PRIVATE, shared_segment_size, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    Y_AXIS.position_memory_id = shmget (IPC_PRIVATE, shared_segment_size, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    X_AXIS.counter_process_memory_id = shmget (IPC_PRIVATE, shared_segment_size, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    Y_AXIS.counter_process_memory_id = shmget (IPC_PRIVATE, shared_segment_size, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    X_AXIS.position_process_memory_id = shmget (IPC_PRIVATE, shared_segment_size, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    Y_AXIS.position_process_memory_id = shmget (IPC_PRIVATE, shared_segment_size, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);

    /* Attach the shared memory segment for each shared memory variable. */
    X_AXIS.encoder_counter = (long*) shmat (X_AXIS.counter_memory_id, 0, 0);
    Y_AXIS.encoder_counter = (long*) shmat (Y_AXIS.counter_memory_id, 0, 0);
    X_AXIS.encoder_position = (int*) shmat (X_AXIS.position_memory_id, 0, 0);
    Y_AXIS.encoder_position = (int*) shmat (Y_AXIS.position_memory_id, 0, 0);
    X_AXIS.counter_child_process = (int*) shmat (X_AXIS.counter_process_memory_id, 0, 0);
    Y_AXIS.counter_child_process = (int*) shmat (Y_AXIS.counter_process_memory_id, 0, 0);
    X_AXIS.position_child_process = (int*) shmat (X_AXIS.position_process_memory_id, 0, 0);
    Y_AXIS.position_child_process = (int*) shmat (Y_AXIS.position_process_memory_id, 0, 0);

    /* Start Motor Encoders */
    start_motor_encoder_counter (X_AXIS.encoder_counter, X_AXIS.ENCODER_PORT, X_AXIS.counter_child_process); // Starts program that counts X axis encoder ticks
    start_motor_encoder_counter (Y_AXIS.encoder_counter, Y_AXIS.ENCODER_PORT, Y_AXIS.counter_child_process); // Starts program that counts Y axis encoder ticks
    start_motor_encoder_position (X_AXIS.encoder_position, X_AXIS.encoder_counter, X_AXIS.MOTOR_PORT, X_AXIS.position_child_process);
// Starts program that tracks X axis encoder ticks with direction given from motor
    start_motor_encoder_position (Y_AXIS.encoder_position, Y_AXIS.encoder_counter, Y_AXIS.MOTOR_PORT, Y_AXIS.position_child_process);
// Starts program that tracks Y axis encoder ticks with direction given from motor
    msleep(2000);

    // Setup Switches as Inputs //
    set_digital_output(X_AXIS.SWITCH_PORT, 0); // Set switch to input
    set_digital_output(Y_AXIS.SWITCH_PORT, 0); // Set switch to input
    set_digital_output(Z1_AXIS.SWITCH_PORT, 0); // Set switch to input

    // Enable Servos //
    enable_servos();
    set_servo_position(PEN_EXTRUDER.SERVO_PORT, PEN_EXTRUDER.SERVO_POSITION_OPEN); // Intialize extrusion servo placement

    //*********Setup G-Code File Read**********//
    char file_name[30] = "TestBlock.gcode"; // ********************************************************************************************!!!__File Name To Change__!!!
    char file_location[50] = "/media/usb/"; // file location
    strcat(file_location,file_name); // strcat combines the two strings
    mount_USB(); // Mounts Inserted USB Stick
    FILE * fp; // Intitalize file pointer
    fp = fopen(file_location,"r"); // Open G-Code file
    char * line; // Initialize variable for line of G-Code

    //*********Run G-Code Loop**********//
    while(1){
```

```
        line = get_gcode(fp); // Gets line of code if avalible, return "END" if the file has reached the end.
        if(strcmp(line,"END") == 0){
            break; // If the file has reached its end, break out of the file reading while loop.
        }
        //printf("%s", line); // Prints line of code to screen if desired, for preformance comment out.

        process_gcode(line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER); // Processes and excecutes the line of gcode

        // if statement checks to see if the left button is pressed on the wallaby. If so, it exists the program. //
        if(left_button() == 1){
            break;
        }

        // if statement checks to see if the right button is pressed on the wallaby. If so, it pauses the program until it is pressed again. //
        if(right_button() == 1){
            msleep(2000);
            set_servo_position(PEN_EXTRUDER.SERVO_PORT, PEN_EXTRUDER.SERVO_POSITION_OPEN); // Sets extruder servo position to the open position
            // while loops waits until right button is pressed to continue the program.
            while(right_button() != 1){
            }
        }
    }

    /* Kill Each Child Proccesses */
    kill(*X_AXIS.counter_child_process,SIGKILL);
    kill(*Y_AXIS.counter_child_process,SIGKILL);
    kill(*X_AXIS.position_child_process,SIGKILL);
    kill(*Y_AXIS.position_child_process,SIGKILL);

    /* Detach the shared memory segment for each process.  */
    shmdt (X_AXIS.encoder_counter);
    shmdt (Y_AXIS.encoder_counter);
    shmdt (X_AXIS.encoder_position);
    shmdt (Y_AXIS.encoder_position);
    shmdt (X_AXIS.counter_child_process);
    shmdt (Y_AXIS.counter_child_process);
    shmdt (X_AXIS.position_child_process);
    shmdt (Y_AXIS.position_child_process);

    /* Deallocate the shared memory segment for each process.  */
    shmctl (X_AXIS.counter_memory_id, IPC_RMID, 0);
    shmctl (Y_AXIS.counter_memory_id, IPC_RMID, 0);
    shmctl (X_AXIS.position_memory_id, IPC_RMID, 0);
    shmctl (Y_AXIS.position_memory_id, IPC_RMID, 0);
    shmctl (X_AXIS.counter_process_memory_id, IPC_RMID, 0);
    shmctl (Y_AXIS.counter_process_memory_id, IPC_RMID, 0);
    shmctl (X_AXIS.position_process_memory_id, IPC_RMID, 0);
    shmctl (Y_AXIS.position_process_memory_id, IPC_RMID, 0);

    //*********Close G-Code File Read**********//
    printf("Finished Printing!\n");
    fclose(fp); // Close g-code file
    unmount_USB(); // Unmounts USB stick
    return 0;
}
```

3dPrinter.h

```
#include <gcode.h>
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <kipr/botball.h>

// Function executes g-code command, passed letter & number "ie: G28", and passed the full line of g-code //
void execute_command(const char * letter, const char * number,char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS,
struct EXTRUDER PEN_EXTRUDER){

    // switch statement checks and sorts for command letter //
    switch (letter[0]){

        case 'G':
            if(strcmp(number, "0") == 0){
                gcode_G0(line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER);
            }
            else if(strcmp(number, "1") == 0){
                gcode_G1(line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER);
            }
            else if(strcmp(number, "28") == 0){
                gcode_G28(line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER);
            }
            else if(strcmp(number, "92") == 0){
                gcode_G92(line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER);
            }
            else{
                printf("Command %s%s does not have a function!\n", letter, number); // If command is not coded
```

```
            }

            break;

        case 'M':

            if(strcmp(number, "84") == 0){
                gcode_M84(line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER);
            }
            else if(strcmp(number, "104") == 0){
                gcode_M104(line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER);
            }
            else if(strcmp(number, "106") == 0){
                gcode_M106(line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER);
            }
            else if(strcmp(number, "107") == 0){
                gcode_M107(line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER);
            }
            else if(strcmp(number, "109") == 0){
                gcode_M109(line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER);
            }
            else if(strcmp(number, "140") == 0){
                gcode_M140(line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER);
            }
            else{
                printf("Command %s%s does not have a function!\n", letter, number); // If command is not coded
            }

            break;

        case 'T':
            printf("Command %s%s does not have a function!\n", letter, number); // If command is not coded

            break;
    }

}


// Function gets and returns g-code on line at a time. //
char * get_gcode(FILE * fp){
    char * line = NULL; // Variable to hold line.
    size_t len = 0;

    if (fp == NULL){
        exit(EXIT_FAILURE); // If file does not open, Exit Failure!
    }

    if (getline(&line, &len, fp) == -1){
        return "END"; // Returns "END" at the files end. (ie. getline return NULL)
    }

    if (line){
        free(line);
    }

    return line;
}


// Function processes g-code on line at a time. //
void process_gcode(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER){

    char char_line[150];
    strcpy(char_line,line);
    char letter[1];
    letter[0] = char_line[0];
    char number[10];
    int i = 0;
    for(i=0;i<sizeof(line);i++){
        if(char_line[i] == letter[0]){
            i++;
            break;
        }
    }

    int j = 0;
    while(1){
        if(isdigit(char_line[i])){
            number[j] = char_line[i];
            j++;
        }
        else{
            break;
        }
        i++;
    }

    if(strcmp(letter,"G") == 0 || strcmp(letter,"M") == 0 || strcmp(letter,"T") == 0){
        execute_command(letter, number, line, X_AXIS, Y_AXIS, Z1_AXIS, Z2_AXIS, PEN_EXTRUDER); // Executes line of code if command letter is seen, see <gcode.h>
    }
```

```
}


// Function Mounts USB stick on to the Wallaby, this allows for g-code files to be processed. //
void mount_USB(){

    // If statement checks to see if directory exists, if it dosen't, it creates the directory. //
    DIR* dir = opendir("/media/usb"); // opendir makes sure the usb directory exists, returns 1 if found directory.
    if (dir){
        printf("Found USB Directory!\n");
        closedir(dir);
    }
    else{
        printf("Creating USB Directory /media/usb!\n");
        system("sudo  mkdir /media/usb"); // Make Directory to Mount Usb
    }

    // If statment checks to see if usb has been successfully mounted.
    if(!system("sudo mount /dev/sda1 /media/usb")){
        printf("USB Mount Successful!\n");
    }
    else{
        printf("USB Mount Failed!\n");
        printf("Make Sure USB is Inserted or start over the Wallaby!\n");
        exit(EXIT_FAILURE);
    }
}


// This function unmounts the USB. //
void unmount_USB(){
    system("sudo umount /media/usb"); // Unmount Usb
}


// This function starts a new program that keeps track of the motor position //
void start_motor_encoder_counter (long* encoder_counter, int digital_port, int* child_process_id){

    int sensor = digital(digital_port);
    int split;
    split = fork();
    if(split == 0){
        *child_process_id = getpid();
        while(1){
            if(sensor != digital(digital_port)){
                *encoder_counter = *encoder_counter + 1;
                sensor = digital(digital_port);
            }
        }
    }
}


void start_motor_encoder_position (int* encoder_position, long* encoder_counter, int MOTOR_PORT, int* child_process_id){

    clear_motor_position_counter(MOTOR_PORT);

    int split;
    int old_encoder_counter = *encoder_counter;
    int current_encoder_counter;
    int old_position = get_motor_position_counter(MOTOR_PORT);
    int current_position = get_motor_position_counter(MOTOR_PORT);
    split = fork();
    if(split == 0){
        *child_process_id = getpid();

        while(1){
            current_position = get_motor_position_counter(MOTOR_PORT);
            if(current_position > old_position){
                current_encoder_counter = *encoder_counter;
                *encoder_position = *encoder_position + (current_encoder_counter - old_encoder_counter);
                old_position = current_position;
                old_encoder_counter = current_encoder_counter;
            }
            else if(current_position < old_position){
                current_encoder_counter = *encoder_counter;
                *encoder_position = *encoder_position - (current_encoder_counter - old_encoder_counter);
                old_position = current_position;
                old_encoder_counter = current_encoder_counter;
            }
        }

    }
}


void move_encoder_relative_position (long* encoder_counter, int ticks, int MOTOR_PORT){

    int desired_position = *encoder_counter + abs(ticks) - 4;
    if(ticks > 0){
        move_at_velocity(MOTOR_PORT,1500);
        while(*encoder_counter < desired_position){
```

```
        }
    }
    else if(ticks < 0){
        move_at_velocity(MOTOR_PORT,-1500);
        while(*encoder_counter < desired_position){
        }
    }
    else{
    }
    freeze(MOTOR_PORT);
}


void move_encoder_to_position (int* encoder_position, int target_position, int MOTOR_PORT){

    int desired_position = target_position - *encoder_position;
    if(desired_position > 0){
        move_at_velocity(MOTOR_PORT,1500);
        while(*encoder_position < target_position - 4){
        }
    }
    else if(desired_position < 0){
        move_at_velocity(MOTOR_PORT,-1500);
        while(*encoder_position > target_position + 4){
        }
    }
    else{
    }
    freeze(MOTOR_PORT);
}
```

gcode.h

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <kipr/botball.h>

const int MOTOR_SPEED = 1500;

// Declaration of Funtions //
// G //
void gcode_G0(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER);
void gcode_G1(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER);
void gcode_G28(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER);
void gcode_G92(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER);
// M //
void gcode_M84(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER);
void gcode_M104(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER);
void gcode_M106(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER);
void gcode_M107(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER);
void gcode_M109(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER);
void gcode_M140(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER);
// End of Declarations //


// G //
void gcode_G0(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER){
    char command[150] = "";
    strcpy(command,line);
    //printf("%s", command);

    int i = 0;
    int X = 0;
    int Y = 0;
    int Z = 0;
    int X_dev = 4;
    int Y_dev = 4;
    int Z1_dev = 10;
    int Z2_dev = 10;
    int x_stop = 0;
    int y_stop = 0;
    int z1_stop = 0;
    int z2_stop = 0;
    int servo_used = 0;
    char temp_value[10] = "";
    char x_value_char[10] = "";
    char y_value_char[10] = "";
    char z_value_char[10] = "";
    float x_value_mms = 0;
    float y_value_mms = 0;
    float z_value_mms_1 = 0;
    float z_value_mms_2 = 0;
    int x_position_ticks = 0;
    int y_position_ticks = 0;
    int z_position_ticks_1 = 0;
```

```
int z_position_ticks_2 = 0;

for(i = 0;i<strlen(command);i++){

    if(command[i] == 'X'){
        while(command[i] != ' ' && i<strlen(command)){
            i++;
            temp_value[0] = command[i];
            strcat(x_value_char, temp_value);
        }
        X = 1;
        x_value_mms = atof(x_value_char);
    }
    if(command[i] == 'Y'){
        while(command[i] != ' ' && i<strlen(command)){
            i++;
            temp_value[0] = command[i];
            strcat(y_value_char, temp_value);
        }
        y_value_mms = atof(y_value_char);
        Y = 1;
    }
    if(command[i] == 'Z'){
        while(command[i] != ' ' && i<strlen(command)){
            i++;
            temp_value[0] = command[i];
            strcat(z_value_char, temp_value);
        }
        z_value_mms_1 = atof(z_value_char);
        z_value_mms_2 = atof(z_value_char);
        Z = 1;
    }
}
if(X == 0){
    x_value_mms = *X_AXIS.previous;
    x_stop = 1;
}
if(Y == 0){
    y_value_mms = *Y_AXIS.previous;
    y_stop = 1;
}
if(Z == 0){
    z_value_mms_1 = *Z1_AXIS.previous;
    z_value_mms_2 = *Z2_AXIS.previous;
    z1_stop = 1;
    z2_stop = 1;
}

if(X == 1 && *X_AXIS.previous <= x_value_mms){
    x_position_ticks = 7.55905511811*(x_value_mms - *X_AXIS.previous) + 7.55905511811*(*X_AXIS.global_position);
    *X_AXIS.global_position = 7.55905511811/7.55905511811*(x_value_mms - *X_AXIS.previous) + *X_AXIS.global_position;
}
if(X == 1 && *X_AXIS.previous > x_value_mms){
    x_position_ticks = 7.55905511811*(x_value_mms - *X_AXIS.previous) + 7.55905511811*(*X_AXIS.global_position);
    *X_AXIS.global_position = 7.55905511811/7.55905511811*(x_value_mms - *X_AXIS.previous) + *X_AXIS.global_position;
}
if(Y == 1 && *Y_AXIS.previous <= y_value_mms){
    y_position_ticks = 7.55905511811*(y_value_mms - *Y_AXIS.previous) + 7.55905511811*(*Y_AXIS.global_position);
    *Y_AXIS.global_position = 7.55905511811/7.55905511811*(y_value_mms - *Y_AXIS.previous) + *Y_AXIS.global_position;
}
if(Y == 1 && *Y_AXIS.previous > y_value_mms){
    y_position_ticks = 7.55905511811*(y_value_mms - *Y_AXIS.previous) + 7.55905511811*(*Y_AXIS.global_position);
    *Y_AXIS.global_position = 7.55905511811/7.55905511811*(y_value_mms - *Y_AXIS.previous) + *Y_AXIS.global_position;
}
if(Z == 1 && *Z1_AXIS.previous <= z_value_mms_1){
    z_position_ticks_1 = 1030*(z_value_mms_1 - *Z1_AXIS.previous) + 1030*(*Z1_AXIS.global_position);
    z_position_ticks_2 = 1270*(z_value_mms_2 - *Z2_AXIS.previous) + 1270*(*Z2_AXIS.global_position);
    *Z1_AXIS.global_position = (z_value_mms_1 - *Z1_AXIS.previous) + *Z1_AXIS.global_position;
    *Z2_AXIS.global_position = (z_value_mms_2 - *Z2_AXIS.previous) + *Z2_AXIS.global_position;
}
if(Z == 1 && *Z1_AXIS.previous > z_value_mms_1){
    z_position_ticks_1 = 1030*(z_value_mms_1 - *Z1_AXIS.previous) + 1030*(*Z1_AXIS.global_position);
    z_position_ticks_2 = 1270*(z_value_mms_2 - *Z2_AXIS.previous) + 1270*(*Z2_AXIS.global_position);
    *Z1_AXIS.global_position = (z_value_mms_1 - *Z1_AXIS.previous) + *Z1_AXIS.global_position;
    *Z2_AXIS.global_position = (z_value_mms_2 - *Z2_AXIS.previous) + *Z2_AXIS.global_position;
}

if(Z){
    set_servo_position(PEN_EXTRUDER.SERVO_PORT,PEN_EXTRUDER.SERVO_POSITION_OPEN);
    servo_used = 1;
}

if(abs(x_position_ticks - *X_AXIS.encoder_position) > 7.55905511811*10 || abs(y_position_ticks - *Y_AXIS.encoder_position) > 7.55905511811*10){
    set_servo_position(PEN_EXTRUDER.SERVO_PORT,PEN_EXTRUDER.SERVO_POSITION_OPEN);
    servo_used = 1;
}

while(X || Y || Z){
    //printf("Loop!\n");
    //X//
    if(X == 1 && *X_AXIS.previous <= x_value_mms){
        if(x_position_ticks-X_dev <= *X_AXIS.encoder_position){
```

```
            freeze(X_AXIS.MOTOR_PORT);
            x_stop = 1;
        }
        else{
            move_at_velocity(X_AXIS.MOTOR_PORT,MOTOR_SPEED);
        }
        //printf("X desired position : %d, X actual position : %d\n",x_position_ticks+30,get_motor_position_counter(X_AXIS.MOTOR_PORT));
    }
    if(X == 1 && *X_AXIS.previous > x_value_mms){
        if(*X_AXIS.encoder_position <= x_position_ticks+X_dev){
            freeze(X_AXIS.MOTOR_PORT);
            x_stop = 1;
        }
        else{
            move_at_velocity(X_AXIS.MOTOR_PORT,-MOTOR_SPEED);
        }
        //printf("X desired position : %d, X actual position : %d\n",x_position_ticks+30,get_motor_position_counter(X_AXIS.MOTOR_PORT));
    }

    //Y//
    if(Y == 1 && *Y_AXIS.previous <= y_value_mms){
        if(y_position_ticks-Y_dev <= *Y_AXIS.encoder_position){
            freeze(Y_AXIS.MOTOR_PORT);
            y_stop = 1;
        }
        else{
            move_at_velocity(Y_AXIS.MOTOR_PORT,MOTOR_SPEED);
        }
        //printf("Y desired position : %d, Y actual position : %d\n",y_position_ticks+30,get_motor_position_counter(Y_AXIS.MOTOR_PORT));
    }
    if(Y == 1 && *Y_AXIS.previous > y_value_mms){
        if(*Y_AXIS.encoder_position <= y_position_ticks+Y_dev){
            freeze(Y_AXIS.MOTOR_PORT);
            y_stop = 1;
        }
        else{
            move_at_velocity(Y_AXIS.MOTOR_PORT,-MOTOR_SPEED);
        }
        //printf("Y desired position : %d, Y actual position : %d\n",y_position_ticks+30,get_motor_position_counter(Y_AXIS.MOTOR_PORT));
    }

    //Z//
    if(Z == 1 && *Z1_AXIS.previous <= z_value_mms_1){
        if(z_position_ticks_1-Z1_dev <= get_motor_position_counter(Z1_AXIS.MOTOR_PORT)){
            freeze(Z1_AXIS.MOTOR_PORT);
            z1_stop = 1;
        }
        else{
            move_at_velocity(Z1_AXIS.MOTOR_PORT,MOTOR_SPEED);
        }
        if(z_position_ticks_2-Z2_dev <= get_motor_position_counter(Z2_AXIS.MOTOR_PORT)){
            freeze(Z2_AXIS.MOTOR_PORT);
            z2_stop = 1;
        }
        else{
            move_at_velocity(Z2_AXIS.MOTOR_PORT,MOTOR_SPEED);
        }
        //printf("Z1 desired position : %d, Z1 actual position : %d\n",z_position_ticks+30,get_motor_position_counter(Z1_AXIS.MOTOR_PORT));
        //printf("Z2 desired position : %d, Z2 actual position : %d\n",z_position_ticks+30,get_motor_position_counter(Z2_AXIS.MOTOR_PORT));
    }

    if(Z == 1 && *Z1_AXIS.previous > z_value_mms_1){
        if(get_motor_position_counter(Z1_AXIS.MOTOR_PORT) <= z_position_ticks_1+Z1_dev){
            freeze(Z1_AXIS.MOTOR_PORT);
            z1_stop = 1;
        }
        else{
            move_at_velocity(Z1_AXIS.MOTOR_PORT,-MOTOR_SPEED);
        }
        if(get_motor_position_counter(Z2_AXIS.MOTOR_PORT) <= z_position_ticks_2+Z2_dev){
            freeze(Z2_AXIS.MOTOR_PORT);
            z2_stop = 1;
        }
        else{
            move_at_velocity(Z2_AXIS.MOTOR_PORT,-MOTOR_SPEED);
        }
        //printf("Z1 desired position : %d, Z1 actual position : %d\n",z_position_ticks+30,get_motor_position_counter(Z1_AXIS.MOTOR_PORT));
        //printf("Z2 desired position : %d, Z2 actual position : %d\n",z_position_ticks+30,get_motor_position_counter(Z2_AXIS.MOTOR_PORT));
    }

    //msleep(500);
    //Check if Everything has Finished
    if(x_stop && y_stop && z1_stop && z2_stop){
        break;
    }
}

*X_AXIS.previous = x_value_mms;
*Y_AXIS.previous = y_value_mms;
*Z1_AXIS.previous = z_value_mms_1;
*Z2_AXIS.previous = z_value_mms_2;
```

```
    if(servo_used){
        set_servo_position(PEN_EXTRUDER.SERVO_PORT,PEN_EXTRUDER.SERVO_POSITION_CLOSED);
        msleep(2000);
    }

    printf("%2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f\n",x_value_mms,y_value_mms,z_value_mms_1,z_value_mms_2,
    *X_AXIS.global_position,*Y_AXIS.global_position,*Z1_AXIS.global_position,*Z2_AXIS.global_position,get_motor_position_counter(X_AXIS.MOTOR_PORT)/112.,
    get_motor_position_counter(Y_AXIS.MOTOR_PORT)/160., get_motor_position_counter(Z1_AXIS.MOTOR_PORT)/1030., get_motor_position_counter(Z2_AXIS.MOTOR_PORT)/1270.);
    printf("%2.2f %2.2f \n",*X_AXIS.encoder_position/7.55905511811, *Y_AXIS.encoder_position/7.55905511811);
    printf("\n");

}

void gcode_G1(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER){

    set_servo_position(PEN_EXTRUDER.SERVO_PORT,PEN_EXTRUDER.SERVO_POSITION_CLOSED);

    char command[150] = "";
    strcpy(command,line);
    //printf("%s", command);

    int i = 0;
    int X = 0;
    int Y = 0;
    int Z = 0;
    int X_dev = 4;
    int Y_dev = 4;
    int Z1_dev = 10;
    int Z2_dev = 10;
    int x_stop = 0;
    int y_stop = 0;
    int z1_stop = 0;
    int z2_stop = 0;
    int servo_used = 0;
    char temp_value[10] = "";
    char x_value_char[10] = "";
    char y_value_char[10] = "";
    char z_value_char[10] = "";
    float x_value_mms = 0;
    float y_value_mms = 0;
    float z_value_mms_1 = 0;
    float z_value_mms_2 = 0;
    int x_position_ticks = 0;
    int y_position_ticks = 0;
    int z_position_ticks_1 = 0;
    int z_position_ticks_2 = 0;

    for(i = 0;i<strlen(command);i++){

        if(command[i] == 'X'){
            while(command[i] != ' ' && i<strlen(command)){
                i++;
                temp_value[0] = command[i];
                strcat(x_value_char, temp_value);
            }
            X = 1;
            x_value_mms = atof(x_value_char);
        }
        if(command[i] == 'Y'){
            while(command[i] != ' ' && i<strlen(command)){
                i++;
                temp_value[0] = command[i];
                strcat(y_value_char, temp_value);
            }
            y_value_mms = atof(y_value_char);
            Y = 1;
        }
        if(command[i] == 'Z'){
            while(command[i] != ' ' && i<strlen(command)){
                i++;
                temp_value[0] = command[i];
                strcat(z_value_char, temp_value);
            }
            z_value_mms_1 = atof(z_value_char);
            z_value_mms_2 = atof(z_value_char);
            Z = 1;
        }
    }

    if(X == 0){
        x_value_mms = *X_AXIS.previous;
        x_stop = 1;
    }
    if(Y == 0){
        y_value_mms = *Y_AXIS.previous;
        y_stop = 1;
    }
    if(Z == 0){
        z_value_mms_1 = *Z1_AXIS.previous;
        z_value_mms_2 = *Z2_AXIS.previous;
        z1_stop = 1;
        z2_stop = 1;
```

```
}

if(X == 1 && *X_AXIS.previous <= x_value_mms){
    x_position_ticks = 7.55905511811*(x_value_mms - *X_AXIS.previous) + 7.55905511811*(*X_AXIS.global_position);
    *X_AXIS.global_position = 7.55905511811/7.55905511811*(x_value_mms - *X_AXIS.previous) + *X_AXIS.global_position;
}
if(X == 1 && *X_AXIS.previous > x_value_mms){
    x_position_ticks = 7.55905511811*(x_value_mms - *X_AXIS.previous) + 7.55905511811*(*X_AXIS.global_position);
    *X_AXIS.global_position = 7.55905511811/7.55905511811*(x_value_mms - *X_AXIS.previous) + *X_AXIS.global_position;
}
if(Y == 1 && *Y_AXIS.previous <= y_value_mms){
    y_position_ticks = 7.55905511811*(y_value_mms - *Y_AXIS.previous) + 7.55905511811*(*Y_AXIS.global_position);
    *Y_AXIS.global_position = 7.55905511811/7.55905511811*(y_value_mms - *Y_AXIS.previous) + *Y_AXIS.global_position;
}
if(Y == 1 && *Y_AXIS.previous > y_value_mms){
    y_position_ticks = 7.55905511811*(y_value_mms - *Y_AXIS.previous) + 7.55905511811*(*Y_AXIS.global_position);
    *Y_AXIS.global_position = 7.55905511811/7.55905511811*(y_value_mms - *Y_AXIS.previous) + *Y_AXIS.global_position;
}
if(Z == 1 && *Z1_AXIS.previous <= z_value_mms_1){
    z_position_ticks_1 = 1030*(z_value_mms_1 - *Z1_AXIS.previous) + 1030*(*Z1_AXIS.global_position);
    z_position_ticks_2 = 1270*(z_value_mms_2 - *Z2_AXIS.previous) + 1270*(*Z2_AXIS.global_position);
    *Z1_AXIS.global_position = (z_value_mms_1 - *Z1_AXIS.previous) + *Z1_AXIS.global_position;
    *Z2_AXIS.global_position = (z_value_mms_2 - *Z2_AXIS.previous) + *Z2_AXIS.global_position;
}
if(Z == 1 && *Z1_AXIS.previous > z_value_mms_1){
    z_position_ticks_1 = 1030*(z_value_mms_1 - *Z1_AXIS.previous) + 1030*(*Z1_AXIS.global_position);
    z_position_ticks_2 = 1270*(z_value_mms_2 - *Z2_AXIS.previous) + 1270*(*Z2_AXIS.global_position);
    *Z1_AXIS.global_position = (z_value_mms_1 - *Z1_AXIS.previous) + *Z1_AXIS.global_position;
    *Z2_AXIS.global_position = (z_value_mms_2 - *Z2_AXIS.previous) + *Z2_AXIS.global_position;
}

if(abs(z_position_ticks_1 - get_motor_position_counter(Z1_AXIS.MOTOR_PORT)) > 200 && Z == 1){
    set_servo_position(PEN_EXTRUDER.SERVO_PORT,PEN_EXTRUDER.SERVO_POSITION_OPEN);
    servo_used = 1;
}

while(X || Y || Z){
    //printf("Loop!\n");
    //X//
    if(X == 1 && *X_AXIS.previous <= x_value_mms){
        if(x_position_ticks-X_dev <= *X_AXIS.encoder_position){
            freeze(X_AXIS.MOTOR_PORT);
            x_stop = 1;
        }
        else{
            move_at_velocity(X_AXIS.MOTOR_PORT,MOTOR_SPEED);
        }
        //printf("X desired position : %d, X actual position : %d\n",x_position_ticks+30,get_motor_position_counter(X_AXIS.MOTOR_PORT));
    }
    if(X == 1 && *X_AXIS.previous > x_value_mms){
        if(*X_AXIS.encoder_position <= x_position_ticks+X_dev){
            freeze(X_AXIS.MOTOR_PORT);
            x_stop = 1;
        }
        else{
            move_at_velocity(X_AXIS.MOTOR_PORT,-MOTOR_SPEED);
        }
        //printf("X desired position : %d, X actual position : %d\n",x_position_ticks+30,get_motor_position_counter(X_AXIS.MOTOR_PORT));
    }

    //Y//
    if(Y == 1 && *Y_AXIS.previous <= y_value_mms){
        if(y_position_ticks-Y_dev <= *Y_AXIS.encoder_position){
            freeze(Y_AXIS.MOTOR_PORT);
            y_stop = 1;
        }
        else{
            move_at_velocity(Y_AXIS.MOTOR_PORT,MOTOR_SPEED);
        }
        //printf("Y desired position : %d, Y actual position : %d\n",y_position_ticks+30,get_motor_position_counter(Y_AXIS.MOTOR_PORT));
    }
    if(Y == 1 && *Y_AXIS.previous > y_value_mms){
        if(*Y_AXIS.encoder_position <= y_position_ticks+Y_dev){
            freeze(Y_AXIS.MOTOR_PORT);
            y_stop = 1;
        }
        else{
            move_at_velocity(Y_AXIS.MOTOR_PORT,-MOTOR_SPEED);
        }
        //printf("Y desired position : %d, Y actual position : %d\n",y_position_ticks+30,get_motor_position_counter(Y_AXIS.MOTOR_PORT));
    }

    //Z//
    if(Z == 1 && *Z1_AXIS.previous <= z_value_mms_1){
        if(z_position_ticks_1-Z1_dev <= get_motor_position_counter(Z1_AXIS.MOTOR_PORT)){
            freeze(Z1_AXIS.MOTOR_PORT);
            z1_stop = 1;
        }
        else{
            move_at_velocity(Z1_AXIS.MOTOR_PORT,MOTOR_SPEED);
        }
        if(z_position_ticks_2-Z2_dev <= get_motor_position_counter(Z2_AXIS.MOTOR_PORT)){
```

```
                freeze(Z2_AXIS.MOTOR_PORT);
                z2_stop = 1;
            }
            else{
                move_at_velocity(Z2_AXIS.MOTOR_PORT,MOTOR_SPEED);
            }
            //printf("Z1 desired position : %d, Z1 actual position : %d\n",z_position_ticks+30,get_motor_position_counter(Z1_AXIS.MOTOR_PORT));
            //printf("Z2 desired position : %d, Z2 actual position : %d\n",z_position_ticks+30,get_motor_position_counter(Z2_AXIS.MOTOR_PORT));
        }

        if(Z == 1 && *Z1_AXIS.previous > z_value_mms_1){
            if(get_motor_position_counter(Z1_AXIS.MOTOR_PORT) <= z_position_ticks_1+Z1_dev){
                freeze(Z1_AXIS.MOTOR_PORT);
                z1_stop = 1;
            }
            else{
                move_at_velocity(Z1_AXIS.MOTOR_PORT,-MOTOR_SPEED);
            }
            if(get_motor_position_counter(Z2_AXIS.MOTOR_PORT) <= z_position_ticks_2+Z2_dev){
                freeze(Z2_AXIS.MOTOR_PORT);
                z2_stop = 1;
            }
            else{
                move_at_velocity(Z2_AXIS.MOTOR_PORT,-MOTOR_SPEED);
            }
            //printf("Z1 desired position : %d, Z1 actual position : %d\n",z_position_ticks+30,get_motor_position_counter(Z1_AXIS.MOTOR_PORT));
            //printf("Z2 desired position : %d, Z2 actual position : %d\n",z_position_ticks+30,get_motor_position_counter(Z2_AXIS.MOTOR_PORT));
        }

        //msleep(500);
        //Check if Everything has Finished
        //printf("x_stop : %d, y_stop : %d, z1_stop : %d, z2_stop : %d\n",x_stop,y_stop,z1_stop,z2_stop);
        if(x_stop && y_stop && z1_stop && z2_stop){
            break;
        }
    }
    *X_AXIS.previous = x_value_mms;
    *Y_AXIS.previous = y_value_mms;
    *Z1_AXIS.previous = z_value_mms_1;
    *Z2_AXIS.previous = z_value_mms_2;

    if(servo_used){
        set_servo_position(PEN_EXTRUDER.SERVO_PORT,PEN_EXTRUDER.SERVO_POSITION_CLOSED);
        msleep(2000);
    }

    printf("%2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f %2.2f\n",x_value_mms,y_value_mms,z_value_mms_1,z_value_mms_2,
    *X_AXIS.global_position,*Y_AXIS.global_position,*Z1_AXIS.global_position,*Z2_AXIS.global_position,get_motor_position_counter(X_AXIS.MOTOR_PORT)/112.,
    get_motor_position_counter(Y_AXIS.MOTOR_PORT)/160., get_motor_position_counter(Z1_AXIS.MOTOR_PORT)/1030., get_motor_position_counter(Z2_AXIS.MOTOR_PORT)/1270.);
    printf("%2.2f %2.2f \n",*X_AXIS.encoder_position/7.55905511811, *Y_AXIS.encoder_position/7.55905511811);
    printf("\n");

}

void gcode_G28(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER){
    char command[150] = "";
    strcpy(command,line);
    //printf("%s", command);
    set_servo_position(PEN_EXTRUDER.SERVO_PORT,PEN_EXTRUDER.SERVO_POSITION_OPEN);

    int i = 0;
    int X = 0;
    int Y = 0;
    int Z = 0;

    for(i = 0;i<strlen(command);i++){

        if(command[i] == 'X'){
            move_at_velocity(X_AXIS.MOTOR_PORT,-MOTOR_SPEED);
            while(!digital(X_AXIS.SWITCH_PORT)){
            }
            freeze(X_AXIS.MOTOR_PORT);
            clear_motor_position_counter(X_AXIS.MOTOR_PORT);
            *X_AXIS.encoder_position = 0;
            X = 1;
        }
        if(command[i] == 'Y'){
            move_at_velocity(Y_AXIS.MOTOR_PORT,-MOTOR_SPEED);
            while(!digital(Y_AXIS.SWITCH_PORT)){
            }
            freeze(Y_AXIS.MOTOR_PORT);
            clear_motor_position_counter(Y_AXIS.MOTOR_PORT);
            *Y_AXIS.encoder_position = 0;
            Y = 1;
        }
        if(command[i] == 'Z'){
            move_at_velocity(Z1_AXIS.MOTOR_PORT,-MOTOR_SPEED+300);
            move_at_velocity(Z2_AXIS.MOTOR_PORT,-MOTOR_SPEED);
            while(!digital(Z1_AXIS.SWITCH_PORT)){
            }
            freeze(Z1_AXIS.MOTOR_PORT);
            freeze(Z2_AXIS.MOTOR_PORT);
```

```
            clear_motor_position_counter(Z1_AXIS.MOTOR_PORT);
            clear_motor_position_counter(Z2_AXIS.MOTOR_PORT);
            Z = 1;
        }
    }
    if(!X && !Y && !Z){
        move_at_velocity(X_AXIS.MOTOR_PORT,-MOTOR_SPEED);
        while(!digital(X_AXIS.SWITCH_PORT)){
        }
        freeze(X_AXIS.MOTOR_PORT);

        move_at_velocity(Y_AXIS.MOTOR_PORT,-MOTOR_SPEED);
        while(!digital(Y_AXIS.SWITCH_PORT)){
        }
        freeze(Y_AXIS.MOTOR_PORT);
        msleep(250);
        move_at_velocity(Z1_AXIS.MOTOR_PORT,-MOTOR_SPEED+300);
        msleep(250);
        move_at_velocity(Z2_AXIS.MOTOR_PORT,-MOTOR_SPEED);
        while(!digital(Z1_AXIS.SWITCH_PORT)){
        }
        msleep(250);
        freeze(Z1_AXIS.MOTOR_PORT);
        msleep(250);
        freeze(Z2_AXIS.MOTOR_PORT);

        clear_motor_position_counter(X_AXIS.MOTOR_PORT);
        clear_motor_position_counter(Y_AXIS.MOTOR_PORT);
        *X_AXIS.encoder_position = 0;
        *Y_AXIS.encoder_position = 0;
        clear_motor_position_counter(Z1_AXIS.MOTOR_PORT);
        clear_motor_position_counter(Z2_AXIS.MOTOR_PORT);
    }
}


void gcode_G92(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER){
}

// M //
void gcode_M84(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER){
}
void gcode_M104(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER){
}
void gcode_M106(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER){
}
void gcode_M107(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER){
}
void gcode_M109(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER){
}
void gcode_M140(char * line, struct AXIS X_AXIS, struct AXIS Y_AXIS, struct AXIS Z1_AXIS, struct AXIS Z2_AXIS, struct EXTRUDER PEN_EXTRUDER){
}
```