Multipurpose Cartesian Co-ordinate Robot

Index

1. Introduction

1.1 Problem Statement:

The basic idea behind the product is the convenience. In today's robotics market, 3D printers are ubiquitous. Discerning the demand for 3d printing, apparently automation on construction sites is fledgling. There is a demand for machines which can paint the wall. Similarly, some products exist which can draw or sketch. Fulfilling the same requirement, at Robolab Technologies, we think that, automation is the future in the sectors such as, arts, construction, wall painting and interior designing. Therefore, to take the step forward in such advancements, we decided to make a robot which can draw, write and sketch. A 2D robot whose drawing assembly can reach ends of A3 or A4 size of papers.

Stipulated applications include:
• Decoration Drawing
• Computer artwork
• Technical drawing
• Notes and cards
• Writing signatures
• Signing diplomas and other certificates
• "Hand-written" lists

1.2 Applications:

The CCR is an extremely versatile machine, designed to serve a wide variety of everyday and specialized drawing and writing needs. You can use it for almost any task that might normally be carried out with a handheld pen. It allows you to use your computer to produce writing that appears to be handmade, complete with the unmistakable appearance of using a real pen (as opposed to an inkjet or laser printer) to address an envelope or sign one's name. And it does so with precision approaching that of a skilled artist, and — just as importantly — using an arm that never gets tired.

Cartesian Coordinate Robot can be used by a genuinely diverse range of people, including:

- Digital artists, using CCR to plot their artwork
- Celebrities, politicians, and elected officials, using CCR as a signature machine
- University officials and other educators, to sign diplomas and certificates
- Educators, introducing students to digital design and fabrication
- Real estate and insurance agents, who would very much like you to open their "handwritten" envelopes
- Online retailers, including a personalized thank you note with your order

- Makerspaces and hackerspaces, providing a versatile low-cost fabrication tool
- Tinkerers, extending CCR beyond writing implements (etching tools, lasers, LEDs for light painting, vacuum pick-up tools, etc.)
- Pen and ink manufacturers, using CCR to test their pens and inks
- Smartphone and tablet hardware makers, using a stylus to test their hardware
- Mobile device software authors, using a stylus to test their software
- People without use of their hands, who would like to send "handwritten" letters
- Woodworkers, laying out joinery markings directly onto wood
- Research scientists, as a low-cost XY motion platform
- Galleries, for numbering of limited-edition artwork
- Calligraphers, who could use a little wrist relief for certain types of busywork.

1.2 **Additional specifications:**

Specifications	**Value**
Usable Pen Travel	297 × 420 mm (11.69 × 16.53 inches)
Vertical pen travel	12 mm. (0.47 inches)
Maximum XY Travel Speed	5000 mm/min. (8.33 cm/s)
XY Resolution	5.95 steps per mm (151 steps per inch)
Reproducibilty	0.1 mm (0.005 inches)
Avg. allowable Pen Dimension	10 mm (4/10")
Total Dimensions in fully contracted state	600 × 700 × 124 mm
Total Dimensions in fully Extended state	945 × 700 × 124 mm
Maximum Height	124mm (4.88 inches)
Footprint	600 × 700 mm (23.62 × 27.56 inch)
Physical Weight	5.1 kg (11.25 Lb)

Software and Programming Prerequisites	**Application**
Basic C programming	To create CCR Base Code
Arduino IDE	To Program Arduino UNO
Inkscape 0.91	To Draw Required Files
J Tech Laser Tool Extension for Inkscape	To Generate G-code from Inkscape Drawing
GRBL-MI and Xloader	To Upload GRBL Firmware in Arduino UNO
Universal G-code Sender, UGS	To Send the G-code to Arduino and perform Drawing Task
G-code Reader	To learn G-codes

Performance:

- Usable pen travel (inches): Over 11.69 × 16.53 inches (Just over A3 letter size).
- Usable pen travel (millimeters): 297 × 420 mm (Just over A3 size).
- Vertical pen travel: 12 mm. (0.47 inches)
- Maximum XY travel speed: 5000 mm/min. (8.33 cm/s).
- Native XY resolution: 151 steps per inch (5.95 steps per mm).
- Reproducibility (XY): Typically, better than 0.005 inches (0.1 mm) at low speeds. (This is an Example)

Physical:

- Major structural components are machined and/or acrylic plates.
- Holds pens and other drawing instruments up to 4/10" (10 mm) diameter.
- Overall dimensions: Approximately 600 × 700 × 124 mm in compressed state and 945 × 700 × 124 mm. in fully extended state.
- Maximum height with cable guides: Approximately 4.88 inches (12.4 cm).
- Footprint: Approximately 600 × 700 mm (60 × 70 cm, 23.62 × 27.56 inch) in contracted condition.
- Physical weight: 11.25 Lb (5.1 kg).

Software:

- Generate G-code of a drawing in Inkscape using J Tech extension tool.
- Use Universal G-code Sender UGS to send the G-code to the Robot
- Compatible with Mac, Windows, and Linux
- All software free to download and open source
- Internet access is required to download software.

Note: programming is not required to use the CCR.

1.4 Methodology:

1) Designing
2) Manufacturing and Assembly
3) Electronics, actuation and programming

2. Designing:

2.1 Literature Survey:

There are some Drawing Robots and CNC plotters available in the US market. However, their costs are too high. Apart from small CNC plotters which made from printers and DVD drives, Cartesian as the only real product in market exists in the US as "AxiDraw".

AxiDraw: AxiDraw comes in 3 versions and costs around 800 USD that is approximately equivalent to 56000 INR and is only available in the USA.

Cartesio:

- Cartesio is similar to Axidraw, the main differences are:
- Cartesio is a cartesian robot (xy movement) while Axidraw is a type of corexy movement (t-bot I think)
- Cartesio is based on Arduino, while Axidraw is based on the EBB Driver board
- Cartesio has a large working area (40×30), like an A3 paper, while Axidraw has a normal working area (30×20), like an A4 paper.
- Cartesio is 3d printed, while Axidraw is in metal (? this is not very clear watching the pictures)
- Axidraw can write with a fountain pen, Cartesio not (yet)
- Axidraw costs 450$, Cartesio costs 60$.

2.2 Basic Design and Mechanisms Idea

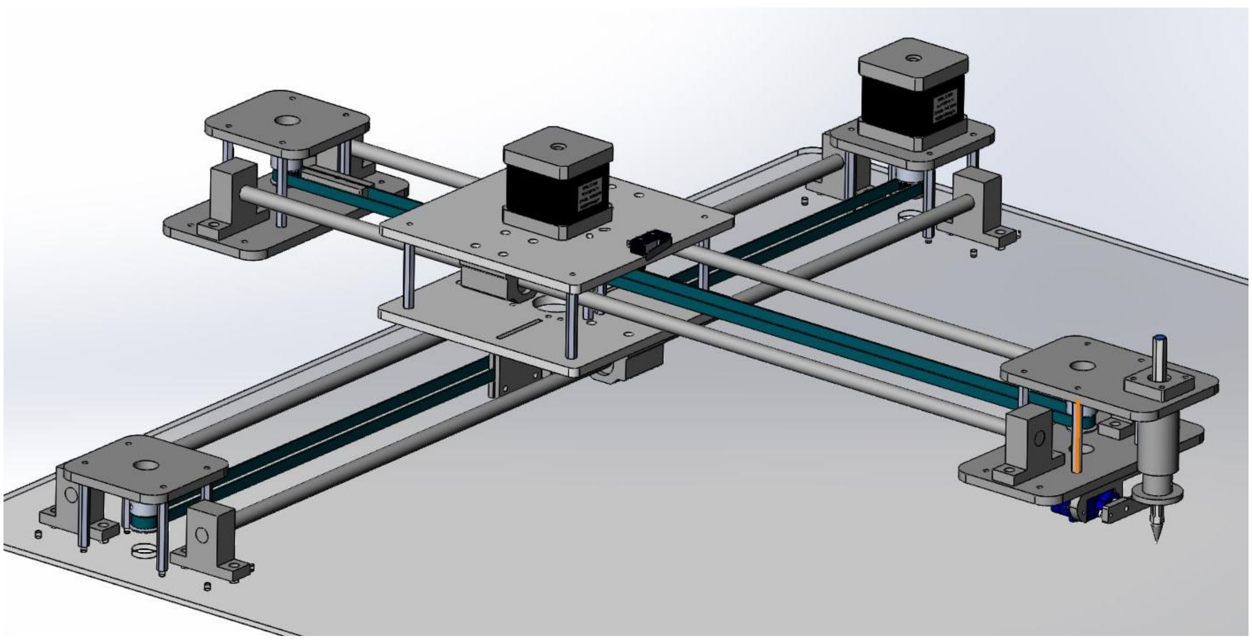
Basic layout of the robot and actuators as well as basic mechanism are decided after research on following references:

1. https://www.prusa3d.com/
2. https://www.instructables.com/id/Homework-Writing-Machine/
3. http://electricdiylab.com/how-to-make-grbl-cnc-v3-shield-based-mini-cnc-machine-from-scrap-dvd-drive/
4. https://www.axidraw.com/

2.3 Actual Modelling in Designing Software

The design of Cartesian Coordinate Robot is made in the software called SolidWorks by Dassault Systems. Some drawing machines consists of 3D printed parts, which may increase material and process cost. Easy solution to this was to avoid 3D printed parts in the system and make it using 2D parts and plates. Acrylic is the optimized material suggested by the company according to availability, process cost and acrylic already exists in the Robolab's Process system as 80% of Robots given to Robolab's are made of acrylic. Therefore, to maintain uniformity, Acrylic is the best option.

a) 3d Model:

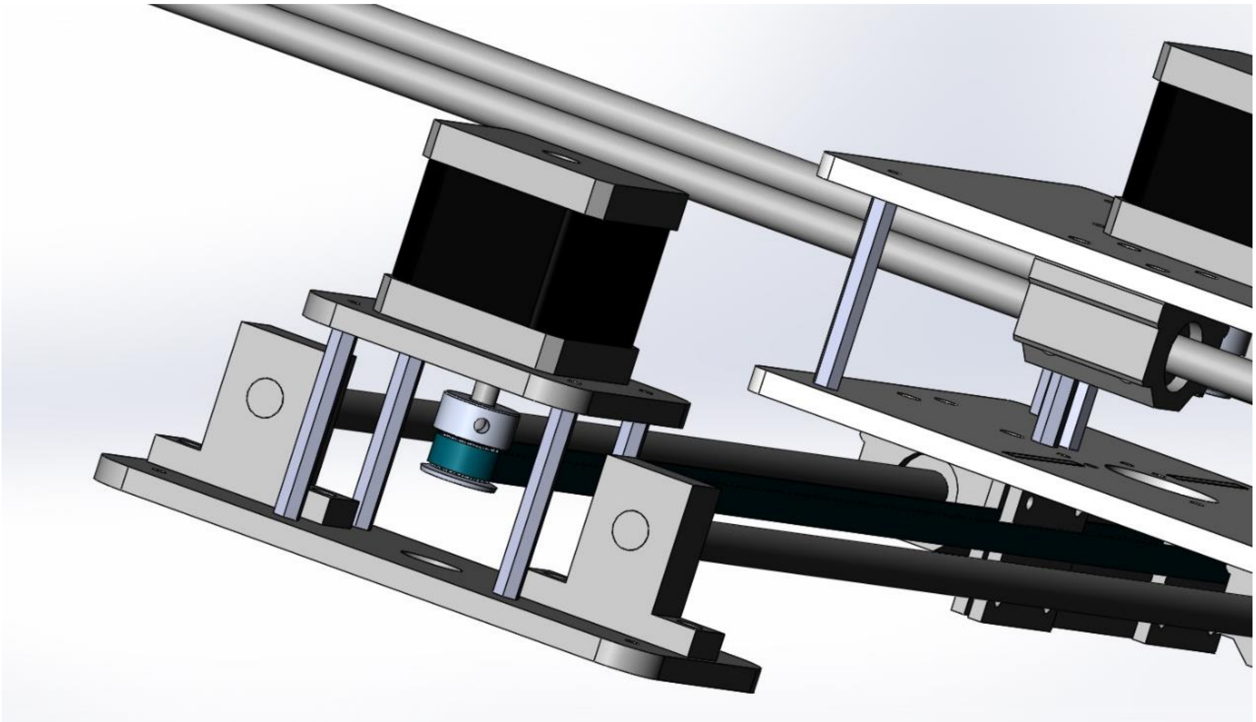


b) Number of Actuators and Type of Actuators Selection:

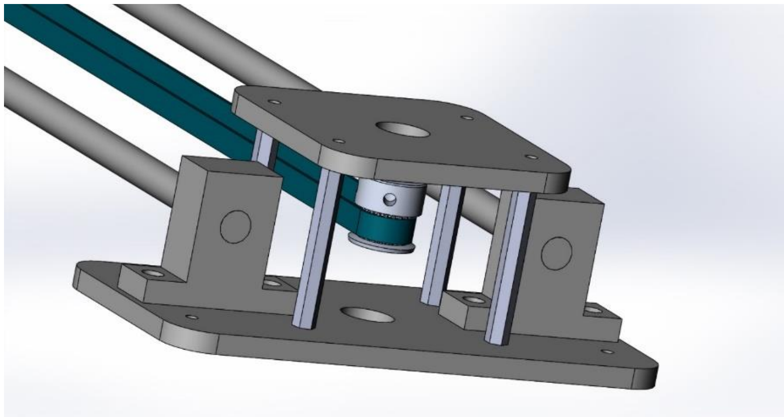
Cartesian Coordinate Robot works on dual stepper drive with timing belt pulley system. The basic task of the robot is motion in the X-Y coordinate axis. Both the axis motions are required to be independent of each other as it may require tracing some complicated curves in the 2D plane and giving dependent motion will only result into X=Y motion of the Robot. As a result, the robot requires 2 Separate motors. Stepper motor out of all are very precise motors without any feedback system. The resolution is of 1.8 degrees. Additionally, stepper motors are also used in equivalent devices like 3D printers.

c) Mechanisms:

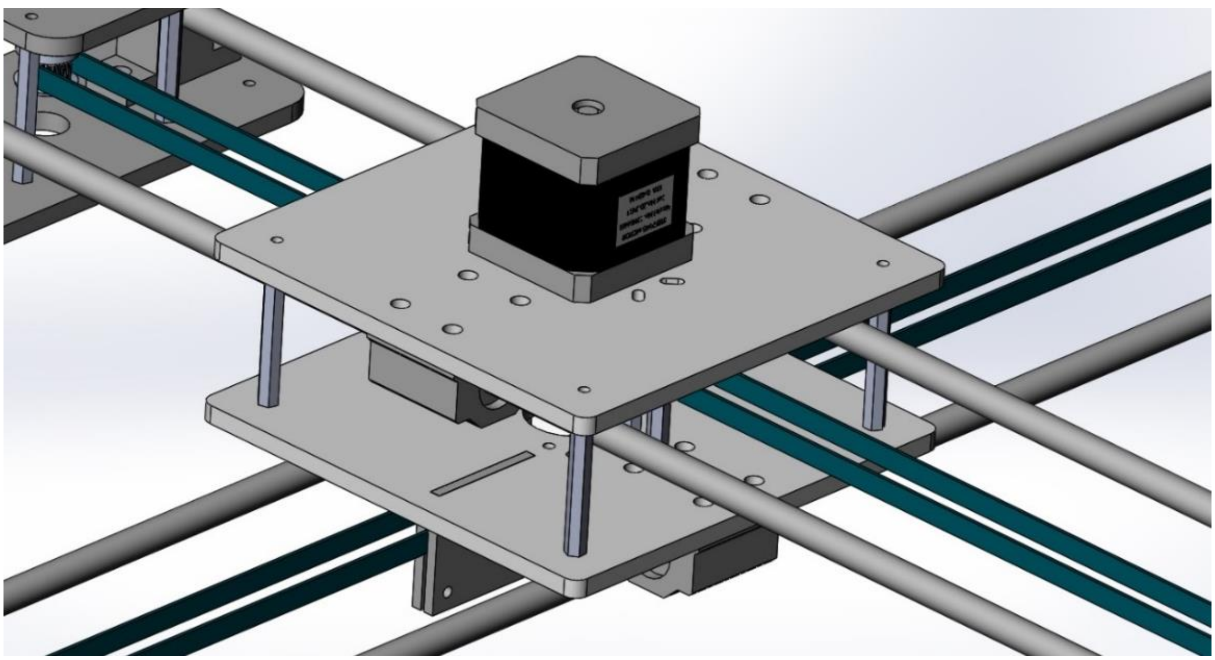
1) Y axis Drive:



Y axis is driven by a stepper motor clamped at the end of the of the axis. The motor is coupled with a GT2 pulley. The belt is clamped at the center of moving body. This Driving assembly consists of a stepper motor, GT2 timing pulley, GT2 timing belt, Rod holders, Guide rods. Rod holders are mounted on extreme subassemblies which clamps and fixes rods. GT2 pulley is supported between two collar bearings of 5/13 and a m5 bolt. Another GT2 bearing is mounted on NEMA 17 stepper motor. Belt assembly of three timing pulleys drives the middle three block system over linear bearing along guide rod.

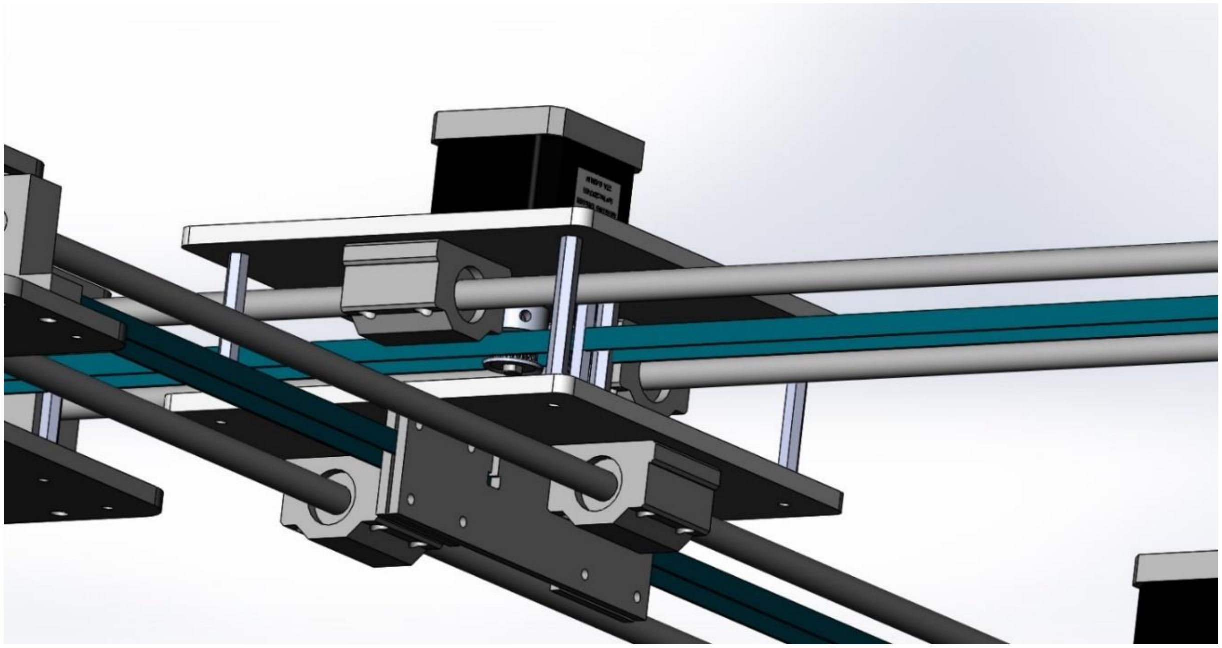


2) X axis Drive:

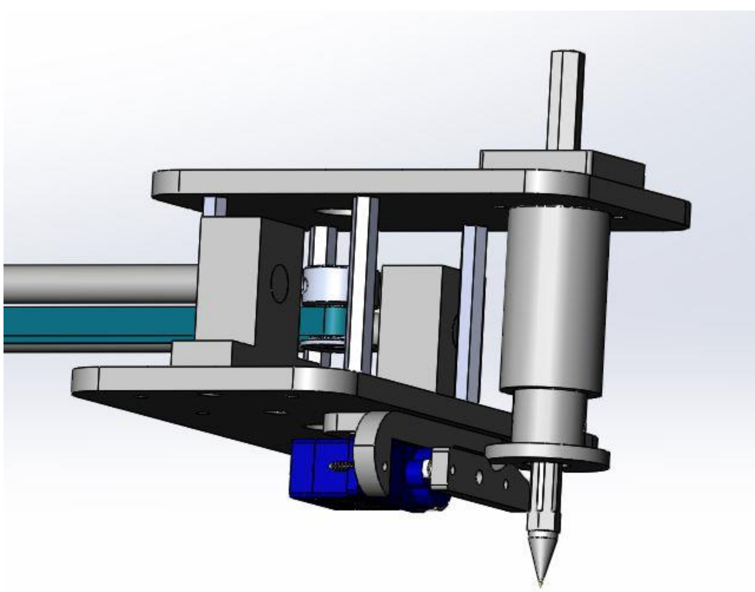


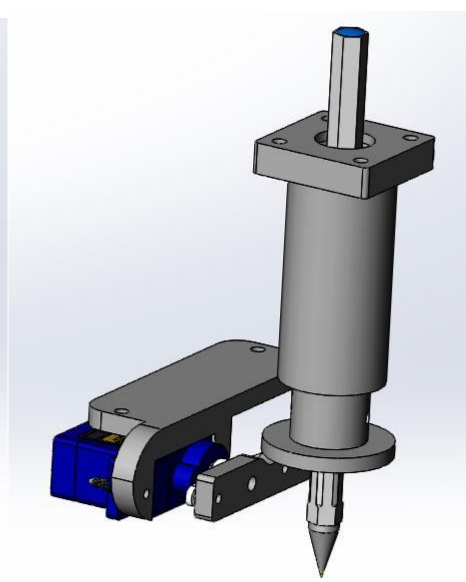
X axis drive consists of NEMA 17 mounted on the middle drive assembly having GT2 Timing belt pulley which drives the two-block system. A belt clamp is connected at one of the ends to provide rack and pinion type mechanism as a resultant. Belt is clamped between two acrylic t joints. The belt must be clamped to the body of the X axis, so that there is no relative motion between the belt and the axis body. Unlike Y axis, only servo motor actuator mover with the axis. This assembly consist of guideway bearings for both the axis. There are two linear bearings for each of the axis at 2 different plates.

However, due to play in the bearing itself, it has observed that even slight deviation in the linear bearing results into a major deflection at the end. And when Bearings gives deviation, 4 bearing per axis system is the most suitable one. Two Bearings in a single row negates the deflection effect of bearings. Therefore, this type of system Total 8 Linear support bearing has to used, 4 for each axis.



3) Z axis Actuation for pen assembly and 3D printed Pen Holder Assembly:





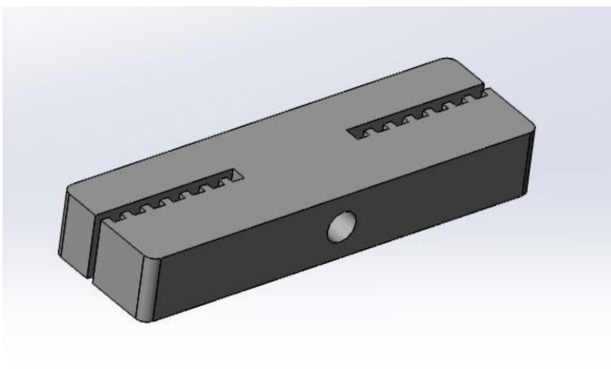
A servo motor is used for the actuation. The motor has a servo horn to which a lever link having suitable profile is connected. Servo when actuated provides a lift to the pen assembly. Now the robot is ready to move the assembly towards the next letter.

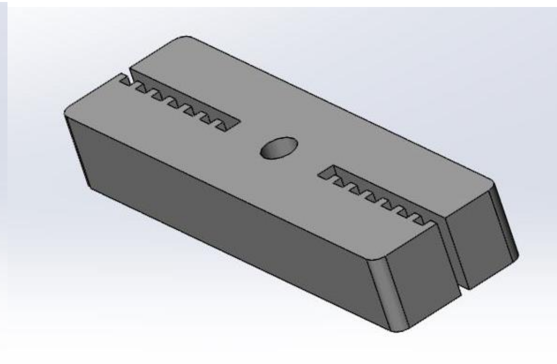
3D printing resolved the problem of wall thickness, gave compactness and it was possible to build complex shaped components for weight optimization. Additionally, the servo clamp is also made by 3d printing.

There is an outer covering and inner covering. The in-between portion of of the assembly consists of a spring. The spring provides the pen appropriate force for writing. Pen fits inside the assembly with press-fit.

4) 3D Printed belt holder:

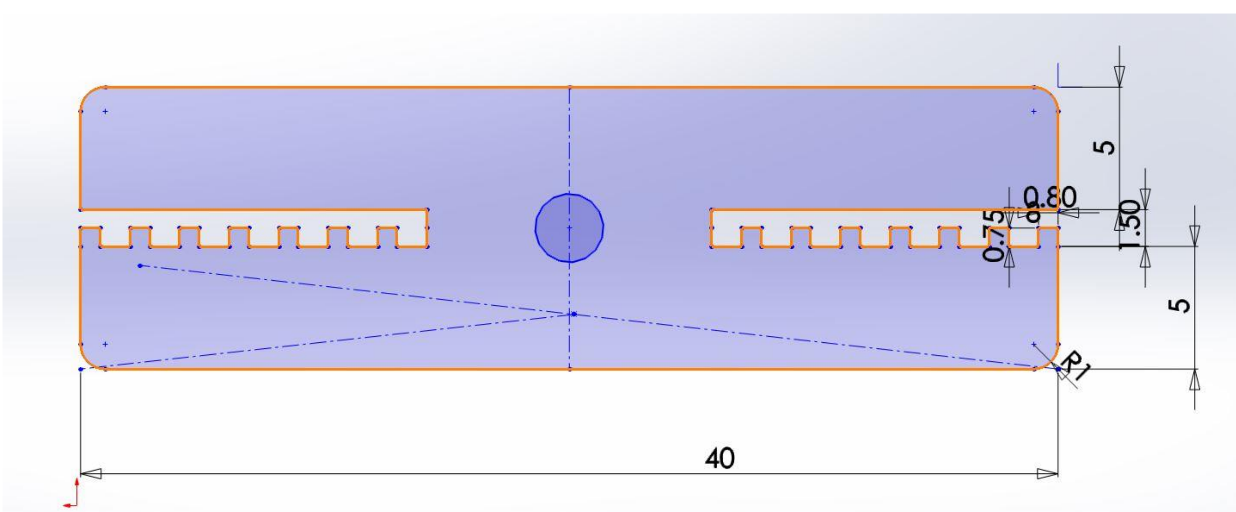
There are a lot of options in the market for GT2 belt holders. However, to learn in depth about the Prusa 3D printer available at the R&D department, I designed It myself and produced it on the Pusa 3D printer.





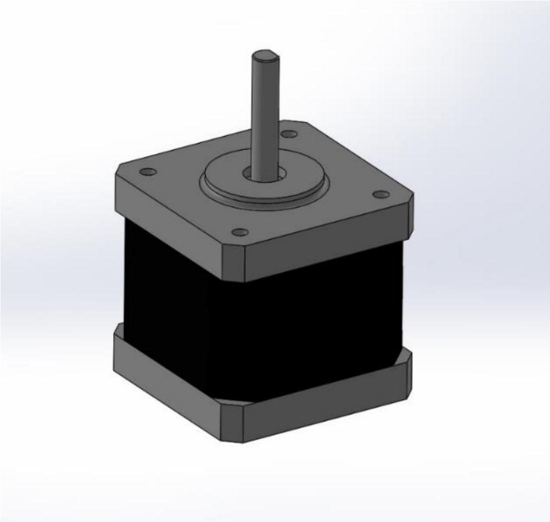
Two different belt holders are required as per the application of bolting one horizontally and other vertically. The connectors holds both the end of the belt with strength. 3D printing makes the assembly of belt easier. The profile for both the belts is created from some sample holders design reference and dimensions diagrams of those belt holders.

Sketch:



2.4 Components Selection:

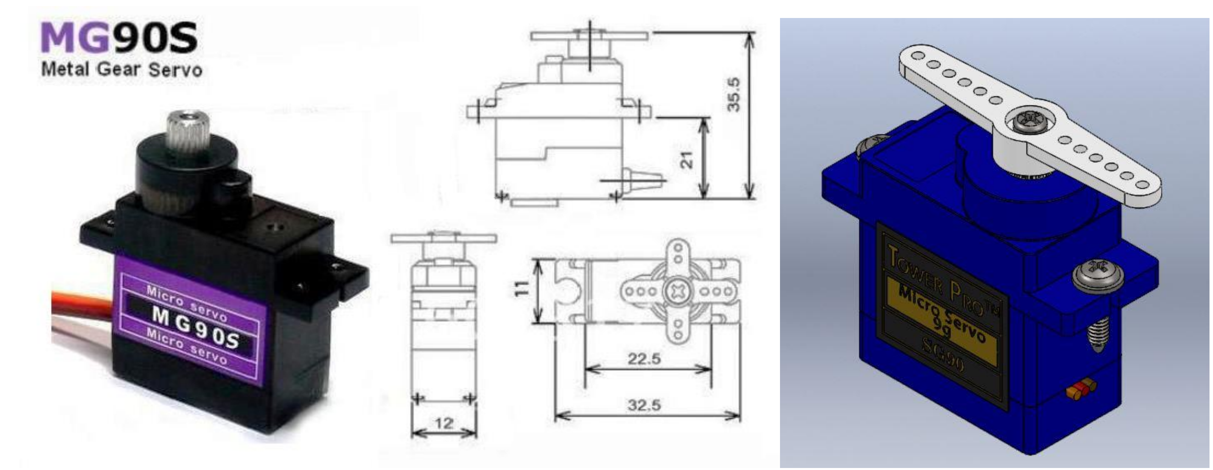
a) Stepper Motors:



- 200 steps per revolution, 1.8 degrees
- Coil 1: Red & Yellow wire pair. Coil 2 Green & Brown/Gray wire pair.
- Bipolar stepper requires 2 full H-bridges
- 4-wire, 8-inch leads
- 42mm/1.65" square body
- 31mm/1.22" square mounting holes, 3mm metric screws (M3)
- 5mm diameter drive shaft, 24mm long, with a machined flat
- 12V rated voltage (you can drive it at a lower voltage, but the torque will drop) at 350mA max current
- 28 oz*in, 20 N*cm, 2 Kg*cm holding torque per phase
- 35 ohms per winding

Furthermore, the main functionality of the robot is to write, draw and sketch. The pen assembly needs to me actuating and de-actuating at various instances. For example, if the robot is given a task to write the word “COEP”, it needs to lift the pen/marker after it has completed writing each word, (after C, After O, After E). Also, this actuation is needed to be at the end of one of the axes. This engenders the cantilever forces on the assembly. As a result, the system requires some light weight actuator which is adequate in providing the lifting force for the pen and is light enough that it does not produce any cantilever forces. Therefore, the best option is to go for Sg90 small servo motor.

b) Servo Motor:

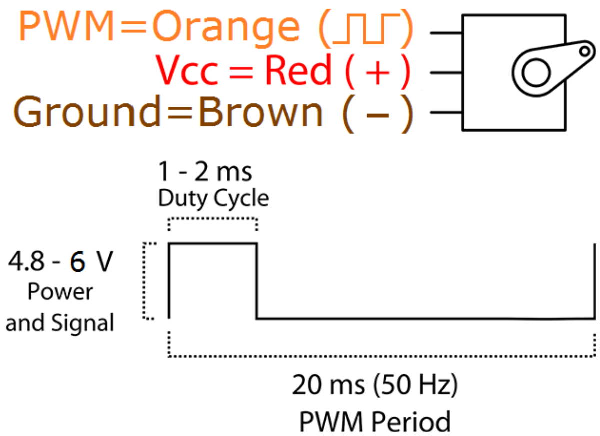


MG90S servo, Metal gear with one bearing

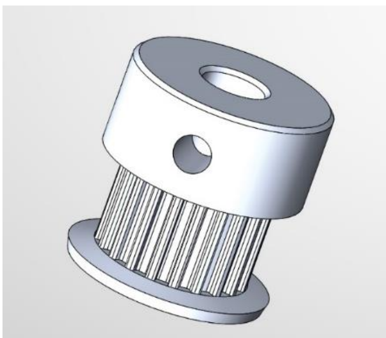
Tiny and lightweight with high output power, this tiny servo is perfect for RC Airplane, Helicopter, Quadcopter or Robot. This servo has *metal gears* for durability. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware. Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.

Specifications:

- Weight: 13.4 g
- Dimension: 22.5 x 12 x 35.5 mm approx.
- Stall torque: 1.8 kgf·cm (4.8V), 2.2 kgf·cm (6 V)
- Operating speed: 0.1 s/60 degree (4.8 V), 0.08 s/60 degree (6 V)
- Operating voltage: 4.8 V - 6.0 V
- Dead band width: 5 µs



c) Timing Pulleys:



- Number of Teeth : 20 teeth, Teeth width: 7mm, Pitch: 2mm
- Bore of Timing pulley : 5mm
- Fixed mode : 2pcs X M4 setscrews
- Suitable for : GT2 6mm width belt
- Material : Aluminum Alloy

Timing Belt: gt2 timing belt

Quick Questions:

Why Stepper Motor?

Stepper motors are an important part of draw-bots as well as 3D printers. They are used in a variety of applications depending on the type of printer. For example, stepper motors are used to move the drawing assembly along the x and y axis. Stepper motors are unique in that they can move to a known interval and then hold that position. Because they are a good motor to move an object to a repeatable position, they are often used in robotics and in printers. Stepper motors come in a variety of sizes. The most popular sizes used in 3D printers are NEMA 14, NEMA 17 and NEMA 23 and NEMA 24. NEMA is a measurement standardized by the National Electrical Manufacturers Association (NEMA) and refers to the frame size of the motor. Just because a motor has a larger frame doesn't mean it has more torque.

Why Tower pro?

Tower pro SG90 is one of the only servo motors available in the market which has less wait and higher precision. The motor is easily available in the market. Tower Pro SG90 is also one of the cheapest there in the market. The torque provided by the motor suffice the mechanism requirement, that is to lift the pen assembly.

Why mechanical limit switches?

There are some different criteria that we should use to select a switch type:

- Precision / repeatability: does the switch trigger at the same place every time? How much spread is there in the trigger position? Do environmental changes or machine setting changes affect the trigger position?
- Contact distance: does the switch register with enough clearance to its hard-stop that the homing axis can stop before colliding with something?
- Noise-rejection: does the switch ONLY trigger when it is supposed to?

  In the case of mechanical limit switches, Precision/repeatability depends on the switch quality, length of lever arm attached (longer increases contact distance but is worse for precision), and impact speed of the carriage with the switch. It's possible to have a good mechanical switch or a bad mechanical switch. This is typically a reasonable default choice because it is simple and cheap.

Why this mechanism?

The center drive mechanism reduces the overall weight of the robot. Mechanism usually used for Cartesian Robots involves greater number of pulleys and complex type of assembly.

Why Timing belt and pulley?

When dealing with CCR, accuracy is crucial for obtaining good results. If something comes loose in the middle of the print, or doesn’t move exactly the way it should, it will be clearly visible in the print. Thus, when concerning yourself with a CCR belt, it’s very important to make sure that its movements are as controlled and accurate as possible. The use of stepper motors can help provide more advanced control, but it’s useless if the belt slips.

d) Guide Rods Selection:

Justification:

Type	Hollow	Solid
Cantilever	7.787e-001mm	3.462e-001mm
Center Load	6.580e-002mm	2.892e-002mm
No Load (Self Weight)	1.557e-003mm	6.924e-004mm

Hollow rod Performs better when it comes to canti-lever force than that of Solid rod. However, when it comes to self-weight, the solid rod performs better showing less deflection, As it can be determined from the table above that the hollow rod is better till a critical amount of load, and after this amount of critical load, the solid rods are the best suit for the current application, Additionally, the load on rods is greater than that of the critical load. Therefore, it is advisable to use solid rods in this type of system.

SOM Calculations and formulae: **********

With Mateial

3. Manufacturing and Assembly

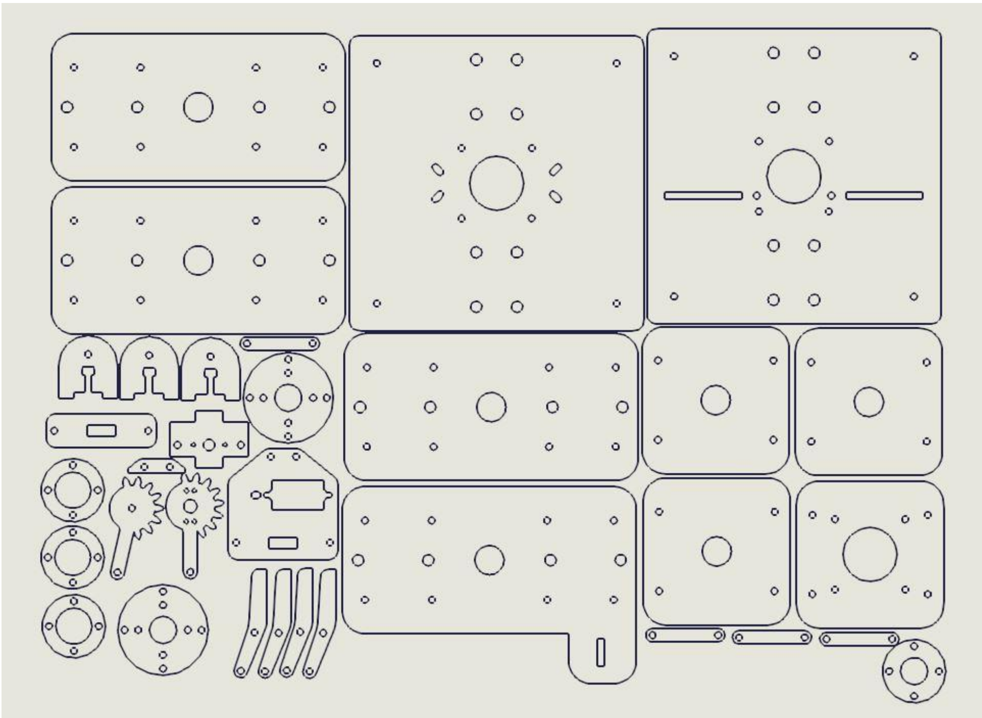
3.1 Bill of Material:

Sr. No.	Part Name	Type	Qty.
Mechanical Parts			
1	Servo Motor	MG996R	1
2	Stepper Motor	NEMA17 - 5.5kgcm T	2
3	Limit Switch	Micro	2
4	GT2 Timing Belt	2500mm, 6mm width	1
5	Tube holder 8mm	for 8mm OD	8
6	Solid Rod	8mmOD	12ft
7	Linear Bearing	for 8mm OD	4
8	Collared Ball_bearing_ID 5mm	ID=5, OD=13	7
9	GT2 timing pulley	bore=5, OD=16	5
10	Acrylic Sheet 5mm	5mm - 2*2 ft	1
11	Acrylic Sheet 3mm	3mm - 1*2 ft	1

Electronics Parts			
1	Arduino Uno	Controller	1
2	A4988 stepper motor driver module	A4988	2
3	Haiyin 11.1V 2200mah LiPo		1
4	DPDT w/o spring NRS 1610 switch		1

3.2 3D to 2D files conversion:

3D files are saved as DWG, DXF, CDR and PDF format by selecting each individual part and saving it differently. Now all the separate .cdr files are combined together on Coral Draw Software according to the sheet size and thickness. Each file is placed so that minimum area becomes waste and the sheet area is optimized. Vender then cuts the material using the files provided.



3.3 Assembly Manual:*****************

4. Electronics, actuation and programming

Actuators Used: 2 NEMA 17 Stepper Motors, 1 MG 90S Servo Motor.

Sensors Used: 2 physical limit switches.

Power Source: 2200 mah Lipo battery.

Microcontroller Used: Arduino UNO + CNC Shield.

4.1 Wiring and Soldering:

a) Block Diagram**********

Actuators wires are extended as per requirement. Wires has to reach to the microcontroller, which will be situated at one of the ends of Y axis assembly. Therefore, one stepper motor (Y axis stepper motor) will have smaller wire length, X axis will have longer length of wiring as the moving motor assembly has to move along the Y axis. Same with the wiring of the servo motor. The servo motor has to move along the X axis as well as along the Y axis. Therefore, the length of the wiring for the servo motor has to be greater than the diagonal of both the axis.

b) Nomenclature of stepper motors wiring:

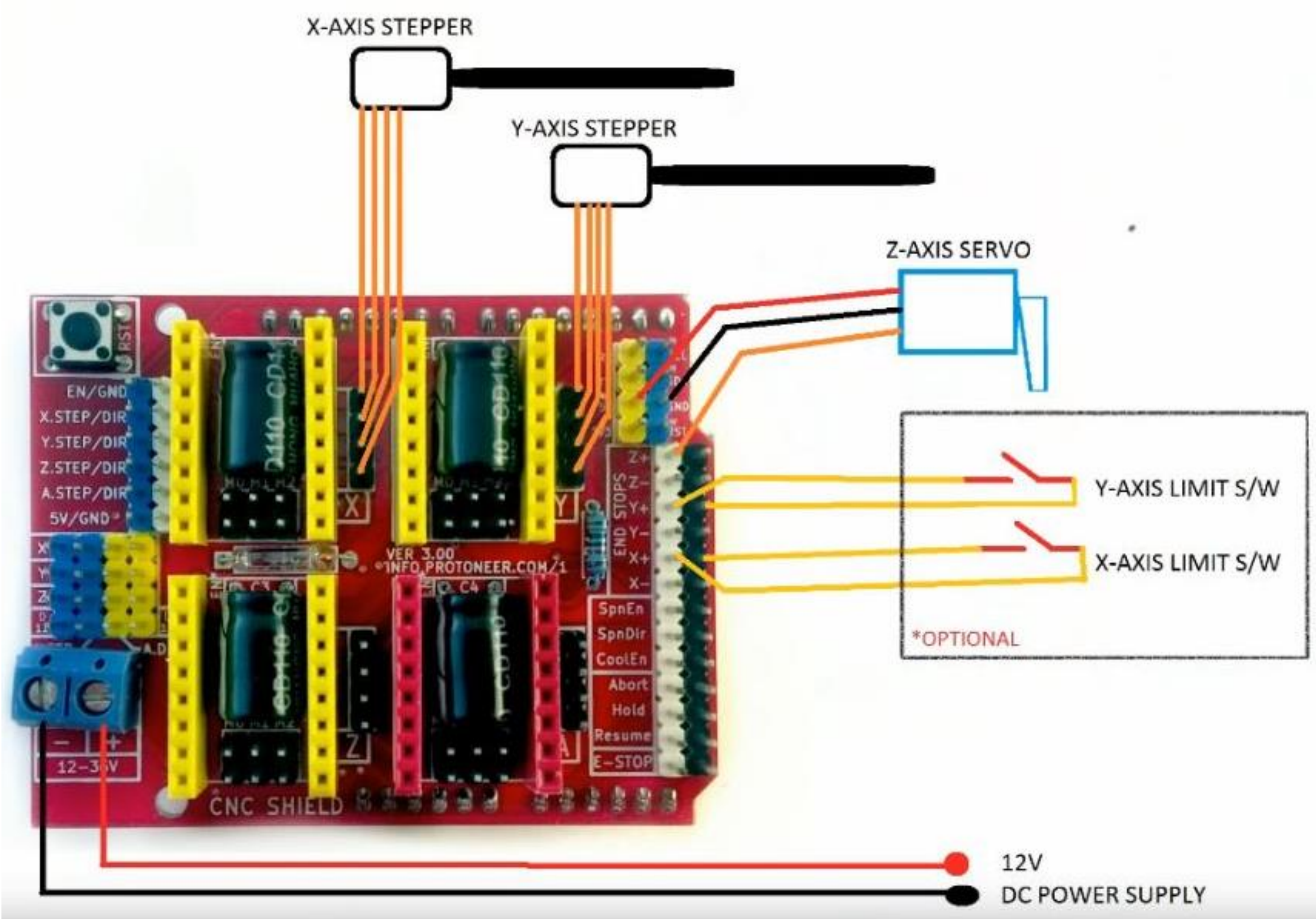
4 LEAD WIRES	1	2	3	4
Color Code 1	Red	Blue	Green	Black
Color Code 2	Brown	Orange	Red	Yellow
Color Code 3	Red	Red White Stripe	Green	Green White Stripe
Bipolar Driver	A	$\bar{A}$	B	$\bar{B}$

6 LEAD WIRES	1	2	3	4	5	6
Color Code 1	Red	White	Blue	Green	Yellow	Black
Color Code 2	Brown	Black	Orange	Red	White	Yellow
Color Code 3	Red	Black	Red White Stripe	Green	White	Green White Stripe
Bipolar Drive Half Coil Connection	A	$\bar{A}$		B	$\bar{B}$	
		$\bar{A}$	A		$\bar{B}$	B
Bipolar Drive Series Connection	A		$\bar{A}$	B		$\bar{B}$
Unipolar Drive	A	A/C Comm	C	B	B/D Comm	D

8 LEAD WIRES	1	2	3	4	5	6	7	8
Color Code 1	Blue White Stripe	Red White Stripe	Blue	Red	Green White Stripe	Black White Stripe	Green	Black
Color Code 2	Red	Yellow White Stripe	Red White Stripe	Yellow	Orange	Black White Stripe	Orange White Stripe	Black
Color Code 3	Red	Black White Stripe	Red White Stripe	Black	Green	Yellow White Stripe	Green White Stripe	Yellow
Bipolar Drive Parallel Connection	A		$\bar{A}$		B		$\bar{B}$	
Bipolar Drive Series Connection	A			$\bar{A}$	B			$\bar{B}$
Unipolar Drive	A	A/C Comm		C	B	B/D Comm		D

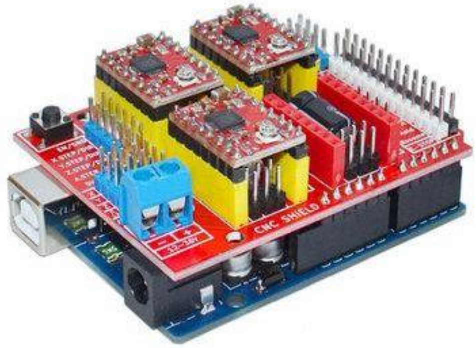
BLDC Connections	Hall Sensor Connections					Winding Connections		
	Vcc	Hall A	Hall B	Hall C	GND	Phase A	Phase B	Phase C
Color Code 1	Red / White	Blue	Green	White	Black / White	Yellow	Red	Black

c) Follow the following wiring diagram:



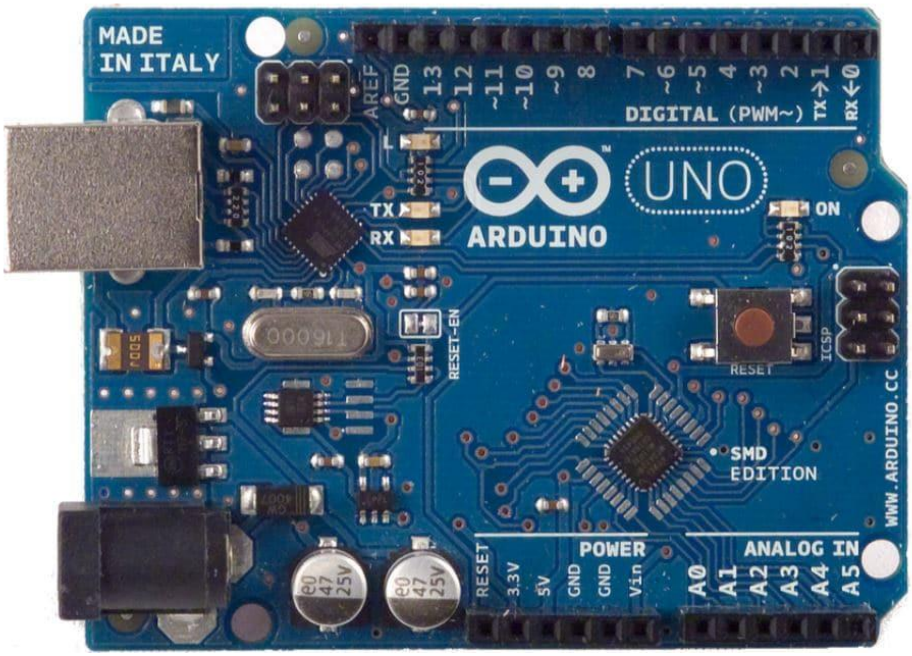
At the yellow positions on the diagram, stepper motor driver needs to be mounted. For stepper motors, relimate male connectors are required. However, for limit switches and servo motor, female relimate connectors are required.

The shield is placed on the Arduino UNO as shown in the figure.



4.2 Getting Started with Arduino UNO: (Arduino Codes)

a) Running 1 Stepper Motor:




```
/ defines pins numbers
const int stepPin = 3;
const int dirPin = 4;

void setup() {
  // Sets the two pins as Outputs
  pinMode(stepPin,OUTPUT);
  pinMode(dirPin,OUTPUT);
}
void loop() {
  digitalWrite(dirPin,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 200; x++) {
```

```
  digitalWrite(stepPin,HIGH);
  delayMicroseconds(500);
  digitalWrite(stepPin,LOW);
  delayMicroseconds(500);
}
delay(1000); // One second delay

digitalWrite(dirPin,LOW); //Changes the rotations direction
// Makes 400 pulses for making two full cycle rotation
for(int x = 0; x < 400; x++) {
  digitalWrite(stepPin,HIGH);
  delayMicroseconds(500);
  digitalWrite(stepPin,LOW);
  delayMicroseconds(500);
}
```

b) Running 2 Stepper motors:

```
const int stepPin1 = 3;
const int dirPin1 = 4;
const int stepPin2 = 5;
const int dirPin2 = 6;

void setup() {
pinMode(stepPin1,OUTPUT);
pinMode(dirPin1,OUTPUT);
pinMode(stepPin2,OUTPUT);
pinMode(dirPin2,OUTPUT);
}

void loop() {
stepper1_forward();
delay(3000);
//stepper2_forward();
//delay(3000);
stepper1_reverse();
delay(3000);
//stepper2_reverse();
//delay(3000);
}

void stepper1_forward(){
  digitalWrite(dirPin1,HIGH);
  for(int x = 0; x < 400; x++) {
```

```
    digitalWrite(stepPin1,HIGH);
    digitalWrite(stepPin1,LOW);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(2000);
  }
}

  void stepper1_reverse(){
  digitalWrite(dirPin1,LOW);
for(int x = 0; x < 400; x++) {
digitalWrite(stepPin1,HIGH);
delayMicroseconds(2000);
digitalWrite(stepPin1,LOW);
delayMicroseconds(2000);
 }
}

void stepper2_forward(){
  digitalWrite(dirPin2,HIGH);
for(int y = 0; y < 400; y++) {
digitalWrite(stepPin2,HIGH);
delayMicroseconds(2000);
digitalWrite(stepPin2,LOW);
delayMicroseconds(2000);
  }
  }

  void stepper2_reverse(){
```

```
  digitalWrite(dirPin2,LOW);
for(int y = 0; y < 400; y++) {
digitalWrite(stepPin2,HIGH);
delayMicroseconds(2000);
digitalWrite(stepPin2,LOW);
delayMicroseconds(2000);
}
  }
```

c) Running two Stepper Motors Simultaneously:

```
const int stepPin1 = 3;
const int dirPin1 = 4;
const int stepPin2 = 5;
const int dirPin2 = 6;

void setup() {
pinMode(stepPin1,OUTPUT);
pinMode(dirPin1,OUTPUT);
pinMode(stepPin2,OUTPUT);
pinMode(dirPin2,OUTPUT);
}

void loop() {
stepper1_2_forward();
//delay(3000);
stepper1_forward_2_reverse();
delay(3000);
stepper1_2_reverse();
delay(3000);
stepper1_reverse_2_forward();
delay(3000);
}

void stepper1_2_forward(){
  digitalWrite(dirPin1,HIGH);
  digitalWrite(dirPin2,HIGH);
  for(int x = 0; x < 50; x++) {
```

```
    digitalWrite(stepPin1,HIGH);
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(1000);
    digitalWrite(stepPin1,LOW);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(1000);
  }
}

  void stepper1_2_reverse(){
  digitalWrite(dirPin1,LOW);
  digitalWrite(dirPin2,LOW);
for(int x = 0; x < 400; x++) {
digitalWrite(stepPin1,HIGH);
digitalWrite(stepPin2,HIGH);
delayMicroseconds(1000);
digitalWrite(stepPin1,LOW);
digitalWrite(stepPin2,LOW);
delayMicroseconds(1000);
 }
}

void stepper1_forward_2_reverse(){
  digitalWrite(dirPin1,HIGH);
  digitalWrite(dirPin2,LOW);
for(int y = 0; y < 400; y++) {
    digitalWrite(stepPin1,HIGH);
    digitalWrite(stepPin2,HIGH);
```

```
    delayMicroseconds(1000);
    digitalWrite(stepPin1,LOW);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(1000);
  }
  }

  void stepper1_reverse_2_forward(){
  digitalWrite(dirPin1,LOW);
  digitalWrite(dirPin2,HIGH);
for(int y = 0; y < 400; y++) {
    digitalWrite(stepPin1,HIGH);
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(1000);
    digitalWrite(stepPin1,LOW);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(1000);
  }
  }
```

4.3 Base Code for the Robot:

```
#include <Servo.h>
Servo servo1;

const int stepPin1 = 3;
const int dirPin1 = 4;
const int stepPin2 = 5;
const int dirPin2 = 6;
const int limitPin1 = 2;
const int limitPin2 = 7;
int Steppe_Speed = 1500;

// in cm

int a1 = 10;
int b1 = 10;

int a2 = 4;
int b2 = 2;

int a3 = 4;
int b3 = 2;

int a4 = 2;
int b4 = 4;
```

```
void setup() {
servo1.attach(A0);
pinMode(stepPin1,OUTPUT);
pinMode(dirPin1,OUTPUT);
pinMode(stepPin2,OUTPUT);
pinMode(dirPin2,OUTPUT);
pinMode(limitPin2,INPUT);
pinMode(limitPin1,INPUT);
}

void loop() {
stepper2_caliberation();
delay(1000);
stepper1_caliberation();
delay(3000);
stepper1_2_forward();
delay(1000);
stepper1_forward_2_reverse();
delay(1000);
stepper1_2_reverse();
delay(1000);
stepper1_reverse_2_forward();
delay(1000);
Up();
//delay(1000);
Down();
//delay(1000);
delay(10000);
```

```
}

//33.615013mm peri of pulley, 0.168075065mm step length, 59.49722524278066 steps=1cm

void Up(){
  for (int i=15; i<=65; i++){
  servo1.write(i);
  delay(20);


  }
}
void Down(){
  for (int i=65; i>=15 ; i--){
    servo1.write(i);
    delay(20);
    }
}

void stepper2_caliberation(){
   digitalWrite(dirPin2,HIGH);
   for(int c = 0; digitalRead(limitPin2) == HIGH; c++){
      digitalWrite(stepPin2,HIGH);
      delayMicroseconds(3000);
      digitalWrite(stepPin2,LOW);
      delayMicroseconds(3000);
}
delay(500);
   digitalWrite(dirPin2,LOW);
```

```
  for(int c = 0; c < 50 ; c++){
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(1500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(1500);
}
}

void stepper1_caliberation(){
  digitalWrite(dirPin1,LOW);
  for(int c = 0; digitalRead(limitPin1) == HIGH; c++){
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(1500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(1500);
}
delay(500);
  digitalWrite(dirPin1,HIGH);
  for(int c = 0; c < 50 ; c++){
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(1500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(1500);
}
}
```

```
void stepper1_2_forward(){


 int a = 59.49722524278066*a1;
 int b = 59.49722524278066*b1;


 if(a >= b) {
   digitalWrite(dirPin1,HIGH);
   digitalWrite(dirPin2,LOW);
   for(int y = 0; y < b; y++) {
     digitalWrite(stepPin1,HIGH);
     digitalWrite(stepPin2,HIGH);
     delayMicroseconds(Steppe_Speed);
     digitalWrite(stepPin1,LOW);
     digitalWrite(stepPin2,LOW);
     delayMicroseconds(Steppe_Speed);
     }
 digitalWrite(dirPin1,HIGH);
 for(int y = 0; y < a-b; y++){
   digitalWrite(stepPin1,HIGH);
   delayMicroseconds(Steppe_Speed);
   digitalWrite(stepPin1,LOW);
   delayMicroseconds(Steppe_Speed);
   }
 } else if (b > a){
   {
     digitalWrite(dirPin1,HIGH);
     digitalWrite(dirPin2,LOW);
     for(int y = 0; y < a; y++) {
```

```
        digitalWrite(stepPin1,HIGH);
        digitalWrite(stepPin2,HIGH);
        delayMicroseconds(Steppe_Speed);
        digitalWrite(stepPin1,LOW);
        digitalWrite(stepPin2,LOW);
        delayMicroseconds(Steppe_Speed);
        }
    }
    {
      digitalWrite(dirPin2,LOW);
      for(int y = 0; y < b-a; y++){
        digitalWrite(stepPin2,HIGH);
        delayMicroseconds(Steppe_Speed);
        digitalWrite(stepPin2,LOW);
        delayMicroseconds(Steppe_Speed);
        }
     }
   }
}


void stepper1_2_reverse(){
int a = 59.49722524278066*a2;
int b = 59.49722524278066*b2;
  if(a >= b) {
    digitalWrite(dirPin1,LOW);
    digitalWrite(dirPin2,HIGH);
    for(int y = 0; y < b; y++) {
```

```
    digitalWrite(stepPin1,HIGH);
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(Steppe_Speed);
    digitalWrite(stepPin1,LOW);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(Steppe_Speed);
    }
digitalWrite(dirPin1,LOW);
for(int y = 0; y < a-b; y++){
  digitalWrite(stepPin1,HIGH);
  delayMicroseconds(Steppe_Speed);
  digitalWrite(stepPin1,LOW);
  delayMicroseconds(Steppe_Speed);
  }
} else if (b > a){
  {
    digitalWrite(dirPin1,LOW);
    digitalWrite(dirPin2,HIGH);
    for(int y = 0; y < a; y++) {
      digitalWrite(stepPin1,HIGH);
      digitalWrite(stepPin2,HIGH);
      delayMicroseconds(Steppe_Speed);
      digitalWrite(stepPin1,LOW);
      digitalWrite(stepPin2,LOW);
      delayMicroseconds(Steppe_Speed);
      }
  }
  {
```

```
    digitalWrite(dirPin2,HIGH);
    for(int y = 0; y < b-a; y++){
      digitalWrite(stepPin2,HIGH);
      delayMicroseconds(Steppe_Speed);
      digitalWrite(stepPin2,LOW);
      delayMicroseconds(Steppe_Speed);
      }
    }
  }
}


void stepper1_forward_2_reverse(){
int a = 59.49722524278066*a3;
int b = 59.49722524278066*b3;
  if(a >= b) {
    digitalWrite(dirPin1,HIGH);
    digitalWrite(dirPin2,HIGH);
    for(int y = 0; y < b; y++) {
      digitalWrite(stepPin1,HIGH);
      digitalWrite(stepPin2,HIGH);
      delayMicroseconds(Steppe_Speed);
      digitalWrite(stepPin1,LOW);
      digitalWrite(stepPin2,LOW);
      delayMicroseconds(Steppe_Speed);
      }
  digitalWrite(dirPin1,HIGH);
  for(int y = 0; y < a-b; y++){
```

```
  digitalWrite(stepPin1,HIGH);
  delayMicroseconds(Steppe_Speed);
  digitalWrite(stepPin1,LOW);
  delayMicroseconds(Steppe_Speed);
  }
} else if (b > a){
  {
    digitalWrite(dirPin1,HIGH);
    digitalWrite(dirPin2,HIGH);
    for(int y = 0; y < a; y++) {
      digitalWrite(stepPin1,HIGH);
      digitalWrite(stepPin2,HIGH);
      delayMicroseconds(Steppe_Speed);
      digitalWrite(stepPin1,LOW);
      digitalWrite(stepPin2,LOW);
      delayMicroseconds(Steppe_Speed);
      }
  }
  {
    digitalWrite(dirPin2,HIGH);
    for(int y = 0; y < b-a; y++){
      digitalWrite(stepPin2,HIGH);
      delayMicroseconds(Steppe_Speed);
      digitalWrite(stepPin2,LOW);
      delayMicroseconds(Steppe_Speed);
      }
   }
 }
```

```
}

  void stepper1_reverse_2_forward(){
int a = 59.49722524278066*a4;
int b = 59.49722524278066*b4;
  if(a >= b) {
    digitalWrite(dirPin1,LOW);
    digitalWrite(dirPin2,LOW);
    for(int y = 0; y < b; y++) {
      digitalWrite(stepPin1,HIGH);
      digitalWrite(stepPin2,HIGH);
      delayMicroseconds(Steppe_Speed);
      digitalWrite(stepPin1,LOW);
      digitalWrite(stepPin2,LOW);
      delayMicroseconds(Steppe_Speed);
      }
  digitalWrite(dirPin1,LOW);
  for(int y = 0; y < a-b; y++){
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(Steppe_Speed);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(Steppe_Speed);
    }
  } else if (b > a){
    {
      digitalWrite(dirPin1,LOW);
      digitalWrite(dirPin2,LOW);
      for(int y = 0; y < a; y++) {
```

```
      digitalWrite(stepPin1,HIGH);
      digitalWrite(stepPin2,HIGH);
      delayMicroseconds(Steppe_Speed);
      digitalWrite(stepPin1,LOW);
      digitalWrite(stepPin2,LOW);
      delayMicroseconds(Steppe_Speed);
      }
   }
   {
     digitalWrite(dirPin2,LOW);
     for(int y = 0; y < b-a; y++){
      digitalWrite(stepPin2,HIGH);
      delayMicroseconds(Steppe_Speed);
      digitalWrite(stepPin2,LOW);
      delayMicroseconds(Steppe_Speed);
      }
    }
  }
}
```

4.4 G-code Generation from the Drawing file:

a) What is G-code-:

G-Code is one of a number of computer code languages that are used to instruct CNC machining devices what motions they need to perform such as work coordinates, canned cycles, and multiple repetitive cycles. Industry has standardized on G-Code as its basic set of CNC machine codes.

G-Code is the most popular programming language used for programming CNC machinery. Some G words alter the state of the machine so that it changes from cutting straight lines to cutting arcs. Other G words cause the interpretation of numbers as millimeters rather than inches. Some G words set or remove tool length or diameter offsets.

G-Code	Description
G00	Rapid Linear Interpolation
G01	Linear Interpolation
G02	Clockwise Circular Interpolation
G03	Counter Clockwise Circular Interpolation
G04	Dwell
G05	High Speed Machining Mode
G10	Offset Input By Program
G12	Clockwise Circle With Entrance And Exit Arcs
G13	Counter Clockwise Circle With Entrance And Exit Arcs
G17	X-Y Plane Selection
G18	Z-X Plane Selection
G19	Y-Z Plane Selection
G28	Return To Reference Point
G34	Special Fixed Cycle (Bolt Hole Circle)
G35	Special Fixed Cycle (Line At Angle)
G36	Special Fixed Cycle (Arc)
G37	Special Fixed Cycle (Grid)
G40	Tool Radius Compensation Cancel
G41	Tool Radius Compensation Left
G42	Tool Radius Compensation Right
G43	Tool Length Compensation
G44	Tool Length Compensation Cancel

G45	Tool Offset Increase
G46	Tool Offset Decrease
G50.1	Programmed Mirror Image Cancel
G51.1	Programmed Mirror Image On
G52	Local Coordinate Setting
G54 - G59	Work Coordinate Registers 1 Thru 6
G60	Unidirectional Positioning
G61	Exact Stop Check Mode
G65	Macro Call (Non-Modal)
G66	Macro Call (Modal)
G68	Programmed Coordinate Rotation
G69	Coordinate Rotation Cancel
G73	Fixed Cycle (Step)
G74	Fixed Cycle (Reverse Tapping)
G76	Fixed Cycle (Fine Boring)
G80	Fixed Cycle Cancel
G81	Fixed Cycle (Drilling / Spot Drilling)
G82	Fixed Cycle (Drilling / Counter Boring)
G83	Fixed Cycle (Deep Hole Drilling)
G84	Fixed Cycle (Tapping)
G85	Fixed Cycle (Boring)
G86	Fixed Cycle (Boring)
G87	Fixed Cycle (Back Boring)
G88	Fixed Cycle (Boring)
G89	Fixed Cycle (Boring)
G90	Absolute Value Command
G91	Incremental Value Command
G92	Work Offset Set
G101	User macro 1 (substitution) =
G102	User macro 1 (addition) +
G103	User macro 1 (subtraction) -
G104	User macro 1 (multiplication) *
G105	User macro 1 (division) /
G106	User macro 1 (square root)

G107	User macro 1 (sine) sin
G108	User macro 1 (cosine) cos
G109	User macro 1 (arc tangent) tan
G110	User macro (square root)
G200	User macro 1 (unconditional branch)
G201	User macro 1 (zero condition branch)
G202	User macro (negative condition branch)

Examples of G-codes:

Circle:

M03 S0 (servo up)

G90 (To go into absolute distance mode, program G90.) ??

G21 (Unit selection of mm)

G1 F3000 (set the feed rate to 3000)

G1 X194.8819 Y106.2992 (linear move towards start point of the drawing)

G4 P0 (Dwell 0)

M05 S8 (Servo Down)

G4 P0 (Dwell 0) (Till this point, pen comes from origin to the point where it starts drawing and downs the servo)

Circle Drawing begins now:

G1 F500.000000 (set the feed rate to 500)

G2 X168.9366 Y43.6618 I-88.5827 J0. (Circular arm moves in clockwise)

G2 X106.2992 Y17.7165 I-62.6374 J62.6374

G2 X43.6618 Y43.6618 I-0. J88.5827

G2 X17.7165 Y106.2992 I62.6374 J62.6374

G2 X24.4595 Y140.1983 I88.5827 J0.

G2 X43.6618 Y168.9366 I81.8397 J-33.8991

G2 X72.4001 Y188.1389 I62.6374 J-62.6374

G2 X106.2992 Y194.8818 I33.8991 J-81.8397

G2 X140.1983 Y188.1389 I-0. J-88.5827

G2 X168.9366 Y168.9366 I-33.8991 J-81.8397

G2 X188.1389 Y140.1983 I-62.6374 J-62.6374

G2 X194.8819 Y106.2992 I-81.8397 J-33.8991

G1 X194.8819 Y106.2992 (linear move towards 0 0)

G4 P0

M03 S0

G1 F3000

G1 X0 Y0

Command M05- Servo down (in writing condition)

M03- Servo Up (not in writing condition)

Square:

M05 S0

G90

G21

G1 F3000

G1 X35.9303 Y212.1012

G4 P0

M03 S100

G4 P5

G1 F1000.000000

G1 X212.1012 Y212.1012

G1 X212.1012 Y35.9302

G1 X35.9303 Y35.9302

G1 X35.9303 Y212.1012

G4 P0

M05 S0

G1 F3000

G1 X0 Y0

Name: (Robolab)

M05 S0

G90

G21

G1 F3000

G1 X311.9379 Y36.3327

G4 P0

M03 S100

G4 P5

G1 F1000.000000

G1 X304.7091 Y36.3327

G1 X304.7091 Y81.4959

G1 X311.9379 Y81.4959

G1 X311.9379 Y36.3327

G1 X311.9379 Y36.3327

G4 P0

M05 S0

G1 F3000

G1 X290.5589 Y36.3327

G4 P0

```
M03 S100
G4 P5
G1 F1000.000000
G1  X283.33 Y36.3327
G1  X283.33 Y68.7539
G1  X290.5589 Y68.7539
G1  X290.5589 Y36.3327
G1  X290.5589 Y36.3327
G4 P0
M05 S0
G1 F3000
G1  X274.4862 Y36.6229
G4 P0
M03 S100
G4 P5
G1 F1000.000000
G2 X272.4406 Y36.2662 I-8.4143 J42.1999
G2 X270.0258 Y35.9554 I-8.7567 J58.5128
G2 X267.6386 Y35.7517 I-5.4359 J49.6032
G2 X265.7577 Y35.6942 I-1.881 J30.7297
G2 X259.4665 Y36.5199 I0. J24.3792
G2 X255.7603 Y38.3645 I2.6781 J10.0265
G2 X253.3746 Y41.5407 I4.6829 J6.0014
G2 X252.3381 Y46.9269 I13.4764 J5.3862
G1  X252.3381 Y64.1679
G1  X247.4547 Y64.1679
G1  X247.4547 Y68.7539
G1  X252.3381 Y68.7539
```

```
G1  X252.3381 Y78.071
G1  X259.567 Y78.071
G1  X259.567 Y68.7539
G1  X274.4862 Y68.7539
G1  X274.4862 Y64.1679
G1  X259.567 Y64.1679
G1  X259.567 Y49.3941
G3 X259.6159 Y46.8424 I66.5013 J-0.
G3 X259.7208 Y45.3886 I21.6773 J0.8324
G3 X260.059 Y44.0384 I5.0385 J0.5449
G3 X260.7974 Y42.7182 I5.5509 J2.2381
G3 X261.7758 Y41.7311 I3.7265 J2.7152
G3 X263.1045 Y41.0058 I3.26 J4.392
G3 X264.6333 Y40.6559 I2.3438 J6.7266
G3 X267.6418 Y40.4833 I3.0084 J26.1242
G3 X269.4018 Y40.573 I-0. J17.308
G3 X271.3331 Y40.8606 I-2.1606 J21.1364
G3 X273.2487 Y41.294 I-14.3594 J67.9251
G3 X274.1016 Y41.5282 I-3.301 J13.6889
G1  X274.4862 Y41.5282
G1  X274.4862 Y36.6229
G1  X274.4862 Y36.6229
G4 P0
M05 S0
G1 F3000
G1  X238.1879 Y36.3327
G4 P0
M03 S100
```

```
G4 P5
G1 F1000.000000
G1  X230.9975 Y36.3327
G1  X230.9975 Y39.7868
G3 X230.0401 Y39.286 I32.2528 J-62.8265
G3 X228.3828 Y38.3935 I100.7862 J-189.1456
G2 X226.7391 Y37.5888 I-11.0799 J20.5502
G2 X225.2298 Y37.0003 I-7.3933 J16.7317
G2 X223.3984 Y36.4203 I-7.0658 J19.1298
G2 X221.077 Y35.8974 I-8.0324 J30.2449
G2 X218.7185 Y35.5653 I-4.162 J21.0175
G2 X215.5784 Y35.4329 I-3.14 J37.1729
G2 X210.0303 Y36.2339 I-0. J19.6167
G2 X205.7348 Y38.3355 I4.0375 J13.6926
G2 X202.712 Y41.7581 I5.9981 J8.3435
G2 X201.6974 Y45.7369 I7.2939 J3.9787
G2 X202.2835 Y49.1951 I10.496 J-0.
G2 X203.7738 Y51.6871 I6.7916 J-2.37
G2 X206.1313 Y53.6056 I7.8564 J-7.2463
G2 X209.7722 Y55.2862 I9.5714 J-15.9522
G2 X213.7508 Y56.3174 I9.5775 J-28.7599
G2 X219.1929 Y57.0567 I9.9969 J-53.1928
G2 X224.6957 Y57.4514 I18.3354 J-217.0601
G2 X230.9975 Y57.7533 I16.7821 J-284.3806
G1  X230.9975 Y58.595
G3 X230.7472 Y60.3714 I-6.4302 J0.
G3 X230.1131 Y61.6717 I-4.0187 J-1.1552
G3 X229.1243 Y62.7162 I-3.9184 J-2.7189
```

```
G3 X227.6522 Y63.5874 I-4.1437 J-5.323
G3 X226.0776 Y64.1167 I-3.8862 J-8.9533
G3 X223.9609 Y64.4872 I-3.8681 J-15.8677
G3 X221.8043 Y64.6589 I-4.1533 J-38.5235
G3 X219.462 Y64.7193 I-2.3423 J-45.3914
G3 X216.6273 Y64.5899 I0. J-31.1136
G3 X213.1175 Y64.1389 I4.408 J-48.1804
G3 X209.6433 Y63.4752 I8.4956 J-53.899
G3 X205.8886 Y62.5135 I14.3731 J-63.924
G1  X205.5041 Y62.5135
G1  X205.5041 Y68.0573
G2 X207.6275 Y68.4419 I9.1361 J-44.3792
G2 X211.6179 Y69.0151 I24.4755 J-156.2326
G2 X215.6252 Y69.4093 I8.1867 J-62.6555
G2 X219.5005 Y69.5376 I3.8753 J-58.4619
G2 X224.0234 Y69.3712 I0. J-61.5782
G2 X227.383 Y68.957 I-2.53 J-34.3517
G2 X230.6793 Y68.1534 I-3.5202 J-21.6004
G2 X233.2277 Y67.0414 I-4.5778 J-13.9668
G2 X235.4098 Y65.4571 I-5.2519 J-9.5291
G2 X236.919 Y63.5874 I-5.903 J-6.3086
G2 X237.8266 Y61.3686 I-6.8223 J-4.0858
G2 X238.1879 Y58.3338 I-12.5647 J-3.0348
G1  X238.1879 Y36.3327
G1  X238.1879 Y36.3327
G4 P0
M05 S0
G1 F3000
```

G1 X230.9975 Y44.3146

G4 P0

M03 S100

G4 P5

G1 F1000.000000

G1 X230.9975 Y53.3415

G3 X227.6917 Y53.1768 I12.2581 J-279.2088

G3 X223.1918 Y52.9061 I28.8713 J-517.6313

G3 X218.7477 Y52.4975 I5.4254 J-83.3675

G3 X216.1167 Y52.0644 I3.5229 J-29.6121

G3 X213.1222 Y51.1308 I3.6147 J-16.8635

G3 X211.0796 Y49.9746 I3.701 J-8.9204

G3 X209.6752 Y48.3169 I2.6728 J-3.6882

G3 X209.157 Y46.1142 I4.4228 J-2.2027

G3 X209.7666 Y43.662 I5.2371 J-0.

G3 X211.3872 Y41.9055 I4.1398 J2.1937

G3 X213.7465 Y40.9628 I3.7963 J6.0771

G3 X218.1931 Y40.5124 I4.4466 J21.7227

G3 X221.9496 Y40.8082 I0. J24.0009

G3 X225.1529 Y41.6153 I-2.887 J18.2187

G3 X228.2436 Y42.881 I-12.8314 J35.7404

G3 X230.9975 Y44.3146 I-13.7593 J29.7917

G1 X230.9975 Y44.3146

G4 P0

M05 S0

G1 F3000

G1 X197.1602 Y66.4899

G4 P0

```
M03 S100
G4 P5
G1 F1000.000000
G2 X196.8146 Y63.7045 I-11.3991 J0.
G2 X195.8143 Y61.1493 I-10.5649 J2.6623
G2 X194.2705 Y58.9137 I-10.6921 J5.7333
G2 X192.123 Y56.9116 I-10.684 J9.3066
G2 X188.9987 Y55.0067 I-11.1427 J14.7613
G2 X185.1248 Y53.5446 I-10.2669 J21.3398
G2 X181.0432 Y52.7713 I-6.8988 J25.255
G2 X174.9352 Y52.4418 I-6.108 J56.4327
G1  X167.3218 Y52.4418
G1  X167.3218 Y36.3327
G1  X159.7084 Y36.3327
G1  X159.7084 Y79.5513
G1  X175.2428 Y79.5513
G2 X180.3771 Y79.3552 I0. J-67.3246
G2 X183.9713 Y78.8837 I-2.5555 J-33.412
G2 X187.4707 Y78.015 I-4.7619 J-26.6661
G2 X190.3158 Y76.8519 I-6.0036 J-18.7475
G2 X193.3235 Y74.8935 I-7.4783 J-14.7737
G2 X195.3529 Y72.7303 I-7.1614 J-8.7521
G2 X196.6485 Y70.0959 I-6.7949 J-4.9775
G2 X197.1602 Y66.4899 I-12.4514 J-3.6059
G1  X197.1602 Y66.4899
G4 P0
M05 S0
G1 F3000
```

```
G1  X189.2391 Y66.3448
G4 P0
M03 S100
G4 P5
G1 F1000.000000
G3 X188.9567 Y68.4984 I-8.3504 J-0.
G3 X188.2009 Y70.2342 I-6.1292 J-1.6361
G3 X186.964 Y71.6686 I-5.3265 J-3.3425
G3 X185.0479 Y72.9335 I-6.0093 J-7.0193
G3 X183.1527 Y73.6761 I-5.568 J-11.4218
G3 X180.8183 Y74.2106 I-5.0843 J-16.839
G3 X178.4622 Y74.4942 I-3.9169 J-22.6095
G3 X174.8583 Y74.617 I-3.6039 J-52.859
G1  X167.3218 Y74.617
G1  X167.3218 Y57.347
G1  X173.7432 Y57.347
G3 X178.3379 Y57.5334 I0. J56.7042
G3 X181.2412 Y57.9565 I-1.8705 J23.0081
G3 X184.0096 Y58.8454 I-3.3911 J15.3166
G3 X185.9323 Y59.9592 I-3.5975 J8.4266
G3 X187.5601 Y61.4972 I-6.8307 J8.8602
G3 X188.4701 Y62.8908 I-4.6725 J4.0449
G3 X189.0338 Y64.4776 I-6.1758 J3.0873
G3 X189.2391 Y66.3448 I-8.3863 J1.8672
G1  X189.2391 Y66.3448
G4 P0
M05 S0
G1 F3000
```

```
G1  X117.181 Y47.5655
G4 P0
M03 S100
G4 P5
G1 F1000.000000
G2 X115.9393 Y42.541 I-10.7868 J-0.
G2 X112.5668 Y38.7999 I-9.04 J4.7586
G2 X107.7114 Y36.6404 I-8.5897 J12.7748
G2 X100.2623 Y35.7232 I-7.4491 J29.791
G2 X98.4187 Y35.7721 I-0. J34.7806
G2 X95.3405 Y35.9844 I5.1604 J97.2168
G2 X92.2713 Y36.273 I6.2947 J83.4087
G2 X90.188 Y36.5649 I4.36 J38.7017
G1  X90.188 Y41.9346
G1  X90.611 Y41.9346
G3 X92.1944 Y41.5591 I10.4618 J40.5885
G3 X94.4946 Y41.0929 I18.4678 J85.2089
G3 X96.8121 Y40.7683 I4.5829 J24.2856
G3 X99.2241 Y40.6575 I2.4121 J26.2033
G3 X102.7344 Y40.8475 I0. J32.5182
G3 X104.8381 Y41.2671 I-1.3071 J12.0378
G3 X106.8164 Y42.0923 I-2.5951 J9.0048
G3 X107.9526 Y43.0085 I-2.1205 J3.7924
G3 X108.7795 Y44.27 I-3.7773 J3.3777
G3 X109.26 Y45.853 I-5.5374 J2.5451
G3 X109.4808 Y47.5443 I-16.4897 J3.013
G3 X109.5676 Y49.7424 I-27.7926 J2.1981
G1  X109.5676 Y74.9653
```

```
G1  X97.4554 Y74.9653
G1  X97.4554 Y79.5513
G1  X117.181 Y79.5513
G1  X117.181 Y47.5655
G1  X117.181 Y47.5655
G4 P0
M05 S0
G1 F3000
G1  X83.8051 Y57.8985
G4 P0
M03 S100
G4 P5
G1 F1000.000000
G2 X82.9282 Y52.2874 I-18.3901 J0.
G2 X80.3829 Y47.2172 I-17.4766 J5.5993
G2 X76.4936 Y43.0327 I-17.4944 J12.3606
G2 X71.3468 Y39.7868 I-15.4347 J18.77
G2 X67.3324 Y38.2553 I-12.6504 J27.1329
G2 X62.5798 Y37.1454 I-11.1425 J36.984
G2 X57.7523 Y36.5842 I-7.829 J46.3024
G2 X49.8908 Y36.3327 I-7.8614 J122.7444
G1  X35.4331 Y36.3327
G1  X35.4331 Y79.5513
G1  X49.737 Y79.5513
G2 X58.0922 Y79.26 I0. J-119.9643
G2 X63.0413 Y78.6224 I-2.9765 J-42.6331
G2 X67.9173 Y77.4579 I-8.1398 J-44.874
G2 X71.4237 Y76.1263 I-7.2859 J-24.4673
```

```
G2 X76.7728 Y72.776 I-11.3679 J-24.0958
G2 X80.5367 Y68.7829 I-12.6361 J-15.6812
G2 X82.9214 Y63.9142 I-13.5418 J-9.6511
G2 X83.8051 Y57.8985 I-20.0338 J-6.0158
G1  X83.8051 Y57.8985
G4 P0
M05 S0
G1 F3000
G1  X75.8456 Y57.9855
G4 P0
M03 S100
G4 P5
G1 F1000.000000
G3 X75.1894 Y62.8646 I-18.4672 J-0.
G3 X73.5001 Y66.548 I-12.068 J-3.3058
G3 X70.7329 Y69.5224 I-10.8 J-7.273
G3 X66.5019 Y72.0337 I-11.7814 J-15.0295
G3 X63.0514 Y73.2481 I-10.8071 J-25.1973
G3 X59.3115 Y74.0365 I-8.0322 J-28.8371
G3 X55.492 Y74.4485 I-6.6046 J-43.3134
G3 X50.1985 Y74.617 I-5.2935 J-83.1067
G1  X43.0465 Y74.617
G1  X43.0465 Y41.2671
G1  X50.1985 Y41.2671
G3 X55.6841 Y41.4415 I-0. J86.3488
G3 X59.7729 Y41.8765 I-3.0791 J48.3703
G3 X63.7984 Y42.7744 I-4.18 J28.2129
G3 X67.3094 Y44.1405 I-6.8462 J22.79
```

```
G3 X71.1711 Y46.6555 I-7.8829 J16.3266
G3 X73.6923 Y49.5682 I-8.2069 J9.6512
G3 X75.2324 Y53.1145 I-10.2685 J6.5671
G3 X75.8456 Y57.9855 I-19.0413 J4.8711
G1  X75.8456 Y57.9855
G4 P0
M05 S0
G1 F3000
G1  X291.0203 Y74.1816
G4 P0
M03 S100
G4 P5
G1 F1000.000000
G1  X282.8686 Y74.1816
G1  X282.8686 Y79.8415
G1  X291.0203 Y79.8415
G1  X291.0203 Y74.1816
G1  X291.0203 Y74.1816
G4 P0
M05 S0
G1 F3000
G1 X0 Y0
```

b) Inkscape 0.91:

Inkscape is professional quality vector graphics software which runs on Windows, Mac OS X and GNU/Linux. It is used by design professionals and hobbyists worldwide, for creating a wide variety of graphics such as illustrations, icons, logos, diagrams, maps and web graphics. Inkscape uses the W3C open standard SVG (Scalable Vector Graphics) as its native format, and is free and open-source software.

Object Creation:

- Drawing: pencil tool (freehand drawing with simple paths), pen tool (creating Bézier curves and straight lines), calligraphy tool (freehand drawing using filled paths representing calligraphic strokes)
- Shape tools: rectangles (may have rounded corners), ellipses (includes circles, arcs, segments), stars/polygons (can be rounded and/or randomized), spirals
- Text tool (multi-line text, full on-canvas editing)
- Embedded bitmaps (with a command to create and embed bitmaps of selected objects)
- Clones ("live" linked copies of objects), including a tool to create patterns and arrangements of clones

Object Manipulation:

- Transformations (moving, scaling, rotating, skewing), both interactively and by specifying exact numeric values
- Z-order operations (raising and lowering)
- Grouping objects ("select in group" without ungrouping, or "enter the group" making it a temporary layer)
- Layers (lock and/or hide individual layers, rearrange them, etc; layers can form a hierarchical tree)
- Alignment and distribution commands

Fill and Stroke:

- Color selector (RGB, HSL, CMYK, color wheel, CMS)
- Color picker tool
- Copy/paste style
- A gradient editor capable of multi-stop gradients
- Dashed strokes, with many predefined dash patterns

- Path markers (ending, middle and/or beginning marks, e.g. arrowheads)

Operations on Path:

- Node editing: moving nodes and Bezier handles, node alignment and distribution, etc.
- Converting to path (for text objects or shapes), including converting stroke to path
- Boolean operations
- Path simplification, with variable threshold
- Path insetting and outsetting, including dynamic and linked offset objects
- Bitmap tracing (both color and monochrome paths)

Text Support:

- Multi-line text
- Uses any installed outline fonts, including right-to-left scripts
- Kerning, letterspacing, linespacing adjustments
- Text on path (both text and path remain editable)
- Text in shape (fill shape following stroke)

Rendering:

- Fully anti-aliased display
- Alpha transparency support for display and PNG export
- Complete "as you drag" rendering of objects during interactive transformations

File Formats:

- Perfectly compliant SVG format file generation and editing
- Live watching and editing the document tree in the XML editor
- PNG, OpenDocument Drawing, DXF, sk1, PDF, EPS and PostScript export formats and more.

c) Make Inkscape a G code compatible tool:

Software for Laser Engraving and Cutting

There are a few programs we use here at J Tech Photonics, Inc. with our laser upgrade kits to generate the G Code needed to run a program on your 3D printer or CNC machine. We support the vector graphics editing open source software from Inkscape for use with the Laser Tool plug-in. For photos, we support PicLaser from picengrave.com for use with PWM inputs for variable intensity photo engraving. Here you will find how to download and install these programs as well as tutorials on how to use then to generate G Code programs for your laser upgraded machine.

How to Give add the J tech extension?

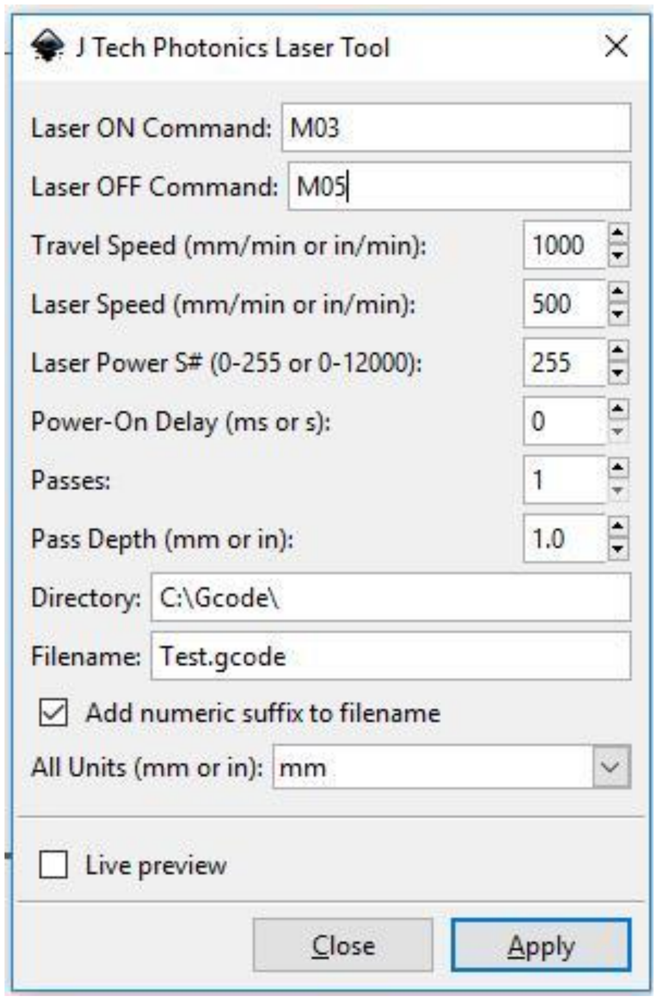
Step 1: Download J Tech Laser tool extension from the link below: http://jtechphotonics.com/?page_id=1980

Step 2: Put the contents of this .zip folder into the "inkscape\share\extensions" folder. Once it is there it will show up under the "extensions" tab in Inkscape.

d) J Tech Laser tool extension properties:

1) What is J Tech Laser tool?

This open source program can transform a picture or, better, a SVG picture in GCODE. This is possible thanks to a plug-in: J Tech Photonic Laser Tool. Below an example of the usage. One can use M03 to lower the pen (drawing) and M05 to upper the pen (no drawing). Put a value on the field Power-On-Delay (i.e. 150ms) because the servo needs a little time to perform the movement.

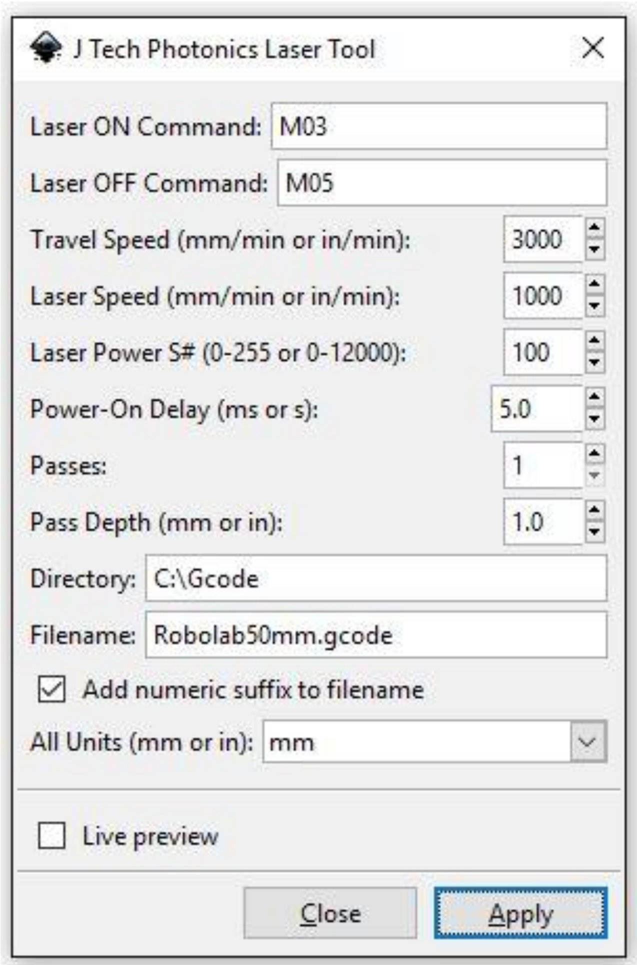


- **Laser ON Command:** The command for turning ON the laser. For example, M03 or M106.
- **Laser OFF Command:** The command for turning OFF the laser. For example, M05 or M107.
- **Travel Speed:** The speed of the machine when the laser is OFF in mm/min.
- **Laser Speed:** The speed of the machine when the laser is ON in mm/min.
- **Laser Power:** If you have PWM control, then you can adjust this. For J Tech firmware and most 3D printers use a number between 0 and 255 (255 being full power). For GRBL 0.9 and 1 standard,

use a number between 0 and 12000 (12000 being full power). If you don't have PWM, keep at max power (either 255 or 12000).

- **Power On Delay:** This will turn on the laser and wait to move until the delay is complete. It is used to heat up the material and initiate the burning process. Delay in ms for 3D printers and seconds for GRBL.
- **Passes:** If cutting, this will repeat the entire path by the number of passes. If engraving leave as 1.
- **Pass Depth:** This will move Z axis down by this amount for each pass. For example, 3mm piece of material with 3 passes might use 1mm per pass to cut all the way through.
- **Directory:** The directory to store the file.
- **Filename:** Name of the file.
- **Add numeric suffix to filename:** Adds a number to the name in case there already is a file with the same name in the directory.
- **All Units:** Change the units to either mm or inches. This will make everything in inches or mm.
- **Live preview:** Shows the path being generated.
- **Apply:** Click to run the converter.

2) Current System Compatibility:



These parameters are best suited for the current system. However, the laser power does not matter in this system and can be neglected. Above values are a result of different iterations.

3) Generate your G-code:

Step 1: Draw in Inkscape.
Step 2: Go to tools command, select J Tech Laser Tool
Step 3: Keep the settings as mentioned above and save the file as .gcode at required location.

4.5 Arduino UNO GRBL Firmware upload:

a. What is Arduino UNO GRBL Firmware?

GRBL is a firmware for Arduino boards(uno, nano, Duemillanove) that controls stepper motors and spindles/lasers. GRBL uses gcode as input and outputs signals via the Arduino pins. Most industrial cnc machines uses parallel port controller that requires Those big purple connectors. Because GRBL arduino boards you just hook it up to a free USB port.

b. Steps to upload the firmware:

Step 1: Download the firmware from https://github.com/arnabdasbwn/grbl-coreXY-servo the ZIP File Unzip the download and you'll have a folder called grbl-coreXY-servo. The firmware's source code has been changed to be compatible with the z axis servo motor.

Step 2: Copy MIGRBL folder, open Arduino folder, paste it in the library. Now the firmware is added.

Step 3: Open Arduino IDE, open MIGRBL from file>example, and compile it to the Arduino.

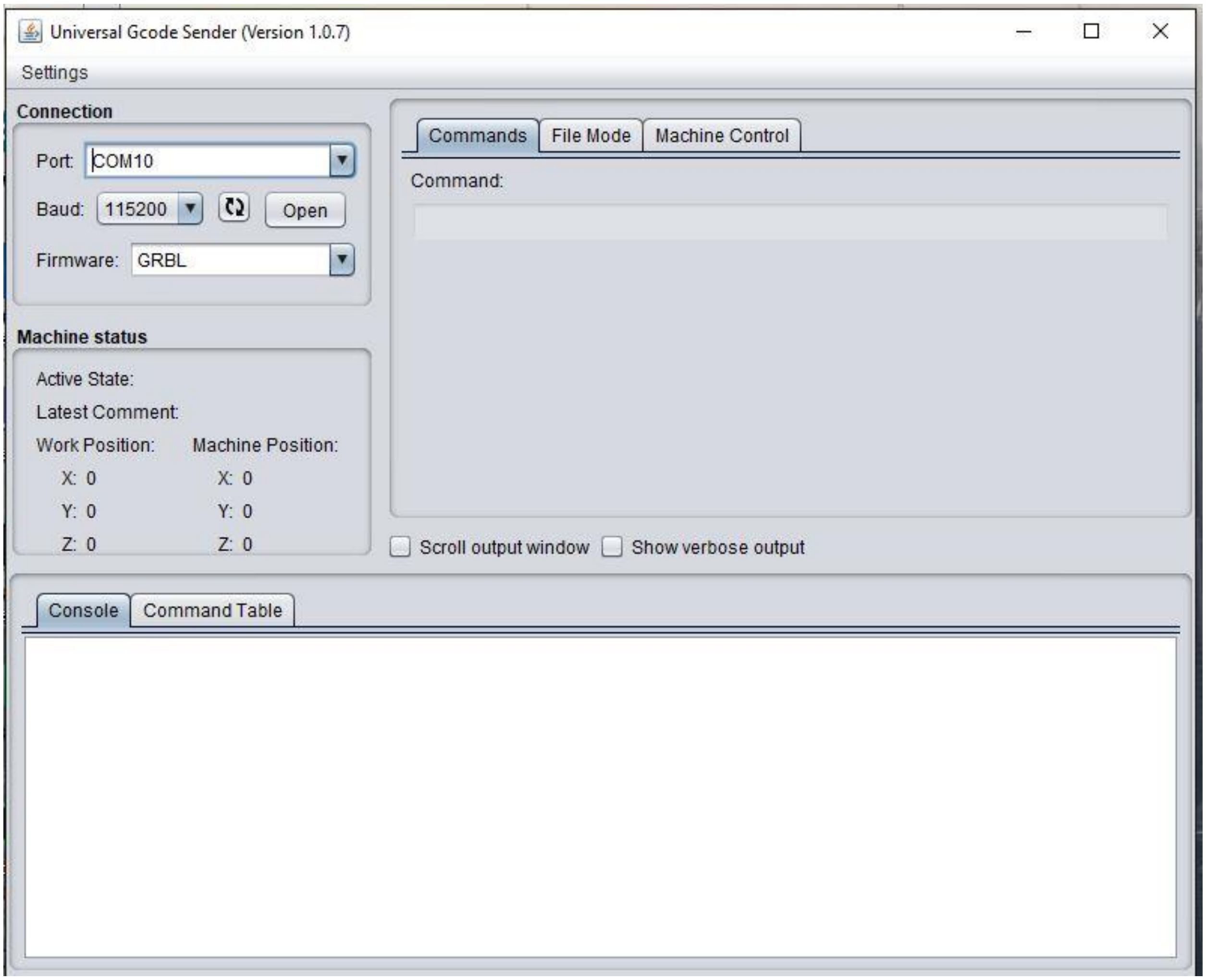
Step 4: Download the .hex file from https://www.youtube.com/redirect?q=https%3A%2F%2Fgithub.com%2Fdownloads%2Fgrbl%2Fgrbl%2Fgrbl_v0_8a_edge_328p_16mhz_9600_build20120310.hex&redir_token=doZjMxUQIyWDc7LUPxwAQP4lsnF8MTU1NDg4ODU2NkAxNTU0ODAyMTY2&v=rUPKBf2vILg&event=video_description

Step 5: If above procedure 1,2 & 3 does not work, follow the procedure below.

Step 6: Download and install Xloader from https://www.youtube.com/redirect?q=http%3A%2F%2Frussemotto.com%2Fxloader%2F&redir_token=doZjMxUQIyWDc7LUPxwAQP4lsnF8MTU1NDg4ODU2NkAxNTU0ODAyMTY2&v=rUPKBf2vILg&event=video_description

Step 7: Browse the .hex file from the Xloader, select Arduino UNO, select the COM port of the connected Arduino, set baud rate = 115200 and say upload.

4.6 Universal G-code Sender:



a) What is Universal G-code Sender?

A full featured gcode platform used for interfacing with advanced CNC controllers like GRBL and TinyG. Universal Gcode Sender is a self-contained Java application which includes all external dependencies, that means if you have the Java Runtime Environment setup UGS provides the rest.

Features:

- Executable Cross platform, tested on Windows, OSX, Linux, and Raspberry Pi.
- All-In-One JAR file - if you have java there is nothing to install. The JAR file includes native dependencies for all supported operating systems.
- 3D Gcode Visualizer with color coded line segments and real time tool position feedback.
- Duration estimates.
- Over 3000 lines of unit test code, and another 1000 lines of comments documenting the tests.
- Configuratble gcode optimization:

- Remove comments
- Truncate decimal precision to configurable amount
- Convert arcs (G2/G3) to line segments
- Remove whitespace

b) Steps for hoe to use UGS

Step 1: Download Universal G-code sender from https://www.youtube.com/redirect?q=http%3A%2F%2Fwww.mediafire.com%2Ffile%2F2l16dvqivwgc9el%2FUGCS.zip&event=video_description&v=kcz1ygh20c0&redir_token=sPJVVLjk6AZFHvid7r8R2qoMFHp8MTU1NDg4ODk5OEAxNTU0ODAyNTk4

Step 2: Set baud rate = 115200, Select the Arduino COM port. Open the port.

Step 3: Go to machine control and give $X command.

Step 4: Now, go to Commands and give $$ command to the command box. Set all the 31 parameters as per the system according to the system calibration. After calibration, the system is ready to process the .gcode file.

4.7 What are GRBL Settings:

Type $ and press enter to have Grbl print a help message. You should not see any local echo of the $ and enter. Grbl should respond with:

```
$$ (view Grbl settings)
$# (view # parameters)
$G (view parser state)
$I (view build info)
$N (view startup blocks)
$x=value (save Grbl setting)
$Nx=line (save startup block)
$C (check gcode mode)
$X (kill alarm lock)
$H (run homing cycle)
~ (cycle start)
! (feed hold)
? (current status)
ctrl-x (reset Grbl)
```

The ‘$’-commands are Grbl system commands used to tweak the settings, view or change Grbl's states and running modes, and start a homing cycle. The last four **non**-'$' commands are Real time control commands that can be sent at any time, no matter what Grbl is doing. These either immediately change Grbl's running behavior or immediately print a report of the important real-time data like current position (aka DRO).

$$ - View Grbl settings

To view the settings, type $$ and press enter after connecting to Grbl. Grbl should respond with a list of the current system settings, as shown in the example below. All of these settings are persistent and kept in EEPROM, so if you power down, these will be loaded back up the next time you power up your Arduino.

```
$0=10 (step pulse, usec)
$1=25 (step idle delay, msec)
$2=0 (step port invert mask:00000000)
$3=6 (dir port invert mask:00000110)
$4=0 (step enable invert, bool)
$5=0 (limit pins invert, bool)
$6=0 (probe pin invert, bool)
$10=3 (status report mask:00000011)
$11=0.020 (junction deviation, mm)
$12=0.002 (arc tolerance, mm)
$13=0 (report inches, bool)
```

```
$20=0 (soft limits, bool)
$21=0 (hard limits, bool)
$22=0 (homing cycle, bool)
$23=1 (homing dir invert mask:00000001)
$24=50.000 (homing feed, mm/min)
$25=635.000 (homing seeks, mm/min)
$26=250 (homing debounce, msec)
$27=1.000 (homing pull-off, mm)
$100=314.961 (x, step/mm)
$101=314.961 (y, step/mm)
$102=314.961 (z, step/mm)
$110=635.000 (x max rate, mm/min)
$111=635.000 (y max rate, mm/min)
$112=635.000 (z max rate, mm/min)
$120=50.000 (x accel, mm/sec^2)
$121=50.000 (y accel, mm/sec^2)
$122=50.000 (z accel, mm/sec^2)
$130=225.000 (x max travel, mm)
$131=125.000 (y max travel, mm)
$132=170.000 (z max travel, mm)
```

$x=val - Save Grbl setting

The $x=val command saves or alters a Grbl setting, which can be done manually by sending this command when connected to Grbl through a serial terminal program, but most Grbl GUIs will do this for you as a user-friendly feature.
To manually change e.g. the microseconds step pulse option to 10us you would type this, followed by an enter:

```
$0=10
```

If everything went well, Grbl will respond with an 'ok' and this setting is stored in EEPROM and will be retained forever or until you change them. You can check if Grbl has received and stored your setting correctly by typing $$ to view the system settings again.

Grbl's $x=val settings and what they mean

NOTE: Settings numbering has changed since v0.8c for future-proofing purposes.

$0 – Step pulse, microseconds

Stepper drivers are rated for a certain minimum step pulse length. Check the data sheet or just try some numbers. You want the shortest pulses the stepper drivers can reliably recognize. If the pulses are too long, you might run into trouble when running the system at very high feed and pulse rates, because the step pulses can begin to overlap each other. We recommend something around 10 microseconds, which is the default value.

$1 - Step idle delay, msec

Every time your steppers complete a motion and come to a stop, Grbl will delay disabling the steppers by this value. **OR**, you can always keep your axes enabled (powered so as to hold position) by setting this value to the maximum 255 milliseconds. Again, just to repeat, you can keep all axes always enabled by setting $1=255.
The stepper idle lock time is the time length Grbl will keep the steppers locked before disabling. Depending on the system, you can set this to zero and disable it. On others, you may need 25-50 milliseconds to make sure your axes come to a complete stop before disabling. This is to help account for machine motors that do not like to be left on for long periods of time without doing something. Also, keep in mind that some stepper drivers don't remember which micro step they stopped on, so when you re-enable, you may witness some 'lost' steps due to this. In this case, just keep your steppers enabled via $1=255.

$2 – Step port invert mask:binary

This setting inverts the step pulse signal. By default, a step signal starts at normal-low and goes high upon a step pulse event. After a step pulse time set by $0, the pin resets to low, until the next step pulse event. When inverted, the step pulse behavior switches from normal-high, to low during the pulse, and back to high. Most users will not need to use this setting, but this can be useful for certain CNC-stepper drivers that have peculiar requirements. For example, an artificial delay between the direction pin and step pulse can be created by inverting the step pin.
This invert mask setting is a value which stores the axes to invert as bit flags. You really don't need to completely understand how it works. You simply need to enter the settings value for the axes you want to invert. For example, if you want to invert the X and Z axes, you'd send $2=5 to Grbl and the setting should now read $2=5 (step port invert mask:00000101).

Setting Value	Mask	Invert X	Invert Y	Invert Z
0	00000000	N	N	N
1	00000001	Y	N	N
2	00000010	N	Y	N
3	00000011	Y	Y	N
4	00000100	N	N	Y
5	00000101	Y	N	Y

Setting Value	Mask	Invert X	Invert Y	Invert Z
6	00000110	N	Y	Y
7	00000111	Y	Y	Y

$3 – Direction port invert mask:binary

This setting inverts the direction signal for each axis. By default, Grbl assumes that the axes move in a positive direction when the direction pin signal is low, and a negative direction when the pin is high. Often, axes don't move this way with some machines. This setting will invert the direction pin signal for those axes that move the opposite way.

This invert mask setting works exactly like the step port invert mask and stores which axes to invert as bit flags. To configure this setting, you simply need to send the value for the axes you want to invert. Use the table above. For example, if want to invert the Y axis direction only, you'd send $3=2 to Grbl and the setting should now read $3=2 (dir port invert mask:00000010)

$4 - Step enable invert, bool

By default, the stepper enable pin is high to disable and low to enable. If your setup needs the opposite, just invert the stepper enable pin by typing $4=1. Disable with $4=0. (May need a power cycle to load the change.)

$5 - Limit pins invert, bool

By default, the limit pins are held normally-high with the Arduino's internal pull-up resistor. When a limit pin is low, Grbl interprets this as triggered. For the opposite behavior, just invert the limit pins by typing $5=1. Disable with $5=0. You may need a power cycle to load the change.
NOTE: If you invert your limit pins, you will need an external pull-down resistor wired in to all of the limit pins to prevent overloading the pins with current and frying them.

$6 - Probe pin invert, bool

By default, the probe pin is held normally-high with the Arduino's internal pull-up resistor. When the probe pin is low, Grbl interprets this as triggered. For the opposite behavior, just invert the probe pin by typing $6=1. Disable with $6=0. You may need a power cycle to load the change.

$10 - Status report mask:binary

This setting determines what Grbl real-time data it reports back to the user when a '?' status report is sent. By default, Grbl will send back its running state (can't be turned off), machine position, and work position (machine position with coordinate offsets and other offsets applied). Three additional reporting features are available that are useful for interfaces or users setting up their machines, which include the serial RX buffer, planner block buffer usage, and limit pin states (as high or low, shown in the order ZYX).

To set them, use the table below to determine what data you'd like Grbl to send back. Select the report types you'd like to see in the status reports and add their values together. This is the value you use to send to Grbl. For example, if you need machine and work positions, add the values 1 and 2 and send Grbl $10=3 to set it. Or, if you need machine position only and limit pin state, add the values 1 and 16 and send Grbl $10=17.
In general, keep this real-time status data to a minimum, since it takes resources to print and send this data back at a high rate. For example, limit pins reporting is generally only needed when users are setting up their machine. Afterwards, it's recommended to disable it, as it isn't very useful once you've got everything figured out.

Report Type	**Value**
Machine Position	1
Work Position	2
Planner Buffer	4
RX Buffer	8
Limit Pins	16

$11 - Junction deviation, mm

Junction deviation is used by the acceleration manager to determine how fast it can move through line segment junctions of a G-code program path. For example, if the G-code path has a sharp 10-degree turn coming up and the machine is moving at full speed, this setting helps determine how much the machine needs to slow down to safely go through the corner without losing steps.

How we calculate it is a bit complicated, but, in general, higher values gives faster motion through corners, while increasing the risk of losing steps and positioning. Lower values make the acceleration manager more careful and will lead to careful and slower cornering. So if you run into problems where your machine tries to take a corner too fast, *decrease* this value to make it slow down when entering corners. If you want your machine to move faster through junctions, *increase* this value to speed it up. For curious people, hit this link to read about Grbl's cornering algorithm, which accounts for both velocity and junction angle with a very simple, efficient, and robust method.

$12 – Arc tolerance, mm

Grbl renders G2/G3 circles, arcs, and helices by subdividing them into teeny tiny lines, such that the arc tracing accuracy is never below this value. You will probably never need to adjust this setting, since 0.002mm is well below the accuracy of most all CNC machines. But if you find that your circles are too crude or arc tracing is performing slowly, adjust this setting. Lower values give higher precision but may lead to performance issues by overloading Grbl with too many tiny lines. Alternately, higher values trace to a lower precision, but can speed up arc performance since Grbl has fewer lines to deal with.
For the curious, arc tolerance is defined as the maximum perpendicular distance from a line segment with its end points lying on the arc, aka a chord. With some basic geometry, we solve for the length of the line segments to trace the arc that satisfies this setting. Modeling arcs in this way is great, because the arc line segments automatically adjust and scale with length to ensure optimum arc tracing performance, while never losing accuracy.

$13 - Report inches, bool

Grbl has a real-time positioning reporting feature to provide a user feedback on where the machine is exactly at that time, as well as, parameters for coordinate offsets and probing. By default, it is set to report in mm, but by sending a $13=1command, you send this boolean flag to true and these reporting features will now report in inches. $13=0 to set back to mm.

$20 - Soft limits, bool

Soft limits is a safety feature to help prevent your machine from traveling too far and beyond the limits of travel, crashing or breaking something expensive. It works by knowing the maximum travel limits for each axis and where Grbl is in machine coordinates. Whenever a new G-code motion is sent to Grbl, it checks whether or not you accidentally have exceeded your machine space. If you do, Grbl will issue an immediate feed hold wherever it is, shutdown the spindle and coolant, and then set the system alarm indicating the problem. Machine position will be retained afterwards, since it's not due to an immediate forced stop like hard limits.

NOTE: Soft limits requires homing to be enabled and accurate axis maximum travel settings, because Grbl needs to know where it is. $20=1 to enable, and $20=0 to disable.

$21 - Hard limits, bool

Hard limit work basically the same as soft limits, but use physical switches instead. Basically, you wire up some switches (mechanical, magnetic, or optical) near the end of travel of each axes, or where ever you feel that there might be trouble if your program moves too far to where it shouldn't. When the switch triggers, it will immediately halt all motion, shutdown the coolant and spindle (if connected), and go into alarm mode, which forces you to check your machine and reset everything.

To use hard limits with Grbl, the limit pins are held high with an internal pull-up resistor, so all you have to do is wire in a normally-open switch with the pin and ground and enable hard limits with $21=1. (Disable with $21=0.) We strongly advise taking electric interference prevention measures. If you want a limit for both ends of travel of one axes, just wire in two switches in parallel with the pin and ground, so if either one of them trips, it triggers the hard limit.
Keep in mind, that a hard limit event is considered to be critical event, where steppers immediately stop and will have likely have lost steps. Grbl doesn't have any feedback on position, so it can't guarantee it has any idea where it is. So, if a hard limit is triggered, Grbl will go into an infinite loop ALARM mode, giving you a chance to check your machine and forcing you to reset Grbl. Remember it's a purely a safety feature.

$22 - Homing cycle, bool

Ahh, homing. For those just initiated into CNC, the homing cycle is used to accurately and precisely locate a known and consistent position on a machine every time you start up your Grbl between sessions. In other words, you know exactly where you are at any given time, every time. Say you start machining something or are about to start the next step in a job and the power goes out, you re-start Grbl and Grbl has no idea where it is. You're left with the task of figuring out where you are. If you have homing, you always have the machine zero reference point to locate from, so all you have to do is run the homing cycle and resume where you left off.

To set up the homing cycle for Grbl, you need to have limit switches in a fixed position that won't get bumped or moved, or else your reference point gets messed up. Usually they are setup in the farthest point in +x, +y, +z of each axes. Wire your limit switches in with the limit pins and ground, just like with the hard limits, and enable homing. If you're curious, you can use your limit switches for both hard limits AND homing. They play nice with each other.

By default, Grbl's homing cycle moves the Z-axis positive first to clear the workspace and then moves both the X and Y-axes at the same time in the positive direction. To set up how your homing cycle behaves, there are more Grbl settings down the page describing what they do (and compile-time options as well.)

Also, one more thing to note, when homing is enabled. Grbl will lock out all G-code commands until you perform a homing cycle. Meaning no axes motions, unless the lock is disabled ($X) but more on that later. Most, if not all CNC controllers, do something similar, as it is mostly a safety feature to prevent users from making a positioning mistake, which is very easy to do and be saddened when a mistake ruins a part. If you find this annoying or find any weird bugs, please let us know and we'll try to work on it so everyone is happy. :)

NOTE: Check out config.h for more homing options for advanced users. You can disable the homing lockout at startup, configure which axes move first during a homing cycle and in what order, and more.

$23 - Homing dir invert mask, int:binary

By default, Grbl assumes your homing limit switches are in the positive direction, first moving the z-axis positive, then the x-y axes positive before trying to precisely locate machine zero by going back and forth slowly around the switch. If your machine has a limit switch in the negative direction, the homing direction mask can invert the axes' direction. It works just like the step port invert and direction port invert masks, where all you have to do is send the value in the table to indicate what axes you want to invert and search for in the opposite direction.

$24 - Homing feed, mm/min

The homing cycle first searches for the limit switches at a higher seek rate, and after it finds them, it moves at a slower feed rate to home into the precise location of machine zero. Homing feed rate is that slower feed rate. Set this to whatever rate value that provides repeatable and precise machine zero locating.

$25 - Homing seek, mm/min

Homing seek rate is the homing cycle search rate, or the rate at which it first tries to find the limit switches. Adjust to whatever rate gets to the limit switches in a short enough time without crashing into your limit switches if they come in too fast.

$26 - Homing debounce, ms

Whenever a switch triggers, some of them can have electrical/mechanical noise that actually 'bounce' the signal high and low for a few milliseconds before settling in. To solve this, you need to debounce the signal, either by hardware with some kind of signal conditioner or by software with a short delay to let the signal finish bouncing. Grbl performs a short delay, only homing when locating machine zero. Set this delay value to whatever your switch needs to get repeatable homing. In most cases, 5-25 milliseconds is fine.

$27 - Homing pull-off, mm

To play nice with the hard limits feature, where homing can share the same limit switches, the homing cycle will move off all of the limit switches by this pull-off travel after it completes. In other words, it helps to prevent accidental triggering of the hard limit after a homing cycle.

$100, $101 and $102 – [X,Y,Z] steps/mm

Grbl needs to know how far each step will take the tool in reality. To calculate steps/mm for an axis of your machine you need to know:

- The mm traveled per revolution of your stepper motor. This is dependent on your belt drive gears or lead screw pitch.
- The full steps per revolution of your steppers (typically 200)

- The microsteps per step of your controller (typically 1, 2, 4, 8, or 16). *Tip: Using high microstep values (e.g., 16) can reduce your stepper motor torque, so use the lowest that gives you the desired axis resolution and comfortable running properties.*

The steps/mm can then be calculated like this: steps_per_mm = (steps_per_revolution*microsteps)/mm_per_rev
Compute this value for every axis and write these settings to Grbl.

$110, $111 and $112 – [X,Y,Z] Max rate, mm/min

This sets the maximum rate each axis can move. Whenever Grbl plans a move, it checks whether or not the move causes any one of these individual axes to exceed their max rate. If so, it'll slow down the motion to ensure none of the axes exceed their max rate limits. This means that each axis has its own independent speed, which is extremely useful for limiting the typically slower Z-axis.

The simplest way to determine these values is to test each axis one at a time by slowly increasing max rate settings and moving it. For example, to test the X-axis, send Grbl something like G0 X50 with enough travel distance so that the axis accelerates to its max speed. You'll know you've hit the max rate threshold when your steppers stall. It'll make a bit of noise, but shouldn't hurt your motors. Enter a setting a 10-20% below this value, so you can account for wear, friction, and the mass of your workpiece/tool. Then, repeat for your other axes.
NOTE: This max rate setting also sets the G0 seek rates.

$120, $121, $122 – [X,Y,Z] Acceleration, mm/sec^2

This sets the axes acceleration parameters in mm/second/second. Simplistically, a lower value makes Grbl ease slower into motion, while a higher value yields tighter moves and reaches the desired feedrates much quicker. Much like the max rate setting, each axis has its own acceleration value and are independent of each other. This means that a multi-axis motion will only accelerate as quickly as the lowest contributing axis can.

Again, like the max rate setting, the simplest way to determine the values for this setting is to individually test each axis with slowly increasing values until the motor stalls. Then finalize your acceleration setting with a value 10-20% below this absolute max value. This should account for wear, friction, and mass inertia. We highly recommend that you dry test some G-code programs with your new settings before committing to them. Sometimes the loading on your machine is different when moving in all axes together.

$130, $131, $132 – [X,Y,Z] Max travel, mm

This sets the maximum travel from end to end for each axis in mm. This is only useful if you have soft limits (and homing) enabled, as this is only used by Grbl's soft limit feature to check if you have exceeded your machine limits with a motion command.

Grbl's Other '$' Commands

The other $ commands provide additional controls for the user, such as printing feedback on the current G-code parser modal state or running the homing cycle. This section explains what these commands are and how to use them.

$# - View gcode parameters

G-code parameters store the coordinate offset values for G54-G59 work coordinates, G28/G30 pre-defined positions, G92 coordinate offset, tool length offsets, and probing (not officially, but we added here anyway). Most of these parameters are directly written to EEPROM anytime they are changed and are persistent. Meaning that they will remain the same, regardless of power-down, until they are explicitly changed. The non-persistent parameters, which will are not retained when reset or power-cycled, are G92, G43.1 tool length offsets, and the G38.2 probing data.

G54-G59 work coordinates can be changed via the G10 L2 Px or G10 L20 Px-command defined by the NIST gcode standard and the EMC2 (linuxcnc.org) standard. G28/G30 pre-defined positions can be changed via the G28.1 and the G30.1 command, respectively.

When $# is called, Grbl will respond with the stored offsets from machine coordinates for each system as follows. TLO denotes tool length offset, and PRBdenotes the coordinates of the last probing cycle.

```
[G54:4.000,0.000,0.000]
[G55:4.000,6.000,7.000]
[G56:0.000,0.000,0.000]
[G57:0.000,0.000,0.000]
[G58:0.000,0.000,0.000]
[G59:0.000,0.000,0.000]
[G28:1.000,2.000,0.000]
[G30:4.000,6.000,0.000]
[G92:0.000,0.000,0.000]
[TLO:0.000,0.000,0.000]
[PRB:0.000,0.000,0.000]
```

$G - View gcode parser state

This command prints all of the active gcode modes in Grbl's G-code parser. When sending this command to Grbl, it will reply with something like:

```
[G0 G54 G17 G21 G90 G94 M0 M5 M9 T0 S0.0 F500.0]
```

These active modes determine how the next G-code block or command will be interpreted by Grbl's G-code parser. For those new to G-code and CNC machining, modes sets the parser into a particular state so you don't have to constantly tell the parser how to parse it. These modes are organized into sets called "modal groups" that cannot be logically active at the same time. For example, the units modal group sets whether your G-code program is interpreted in inches or in millimeters.

A short list of the modal groups, supported by Grbl, is shown below, but more complete and detailed descriptions can be found at LinuxCNC's website. The G-code commands in **bold** indicate the default modes upon powering-up Grbl or resetting it.

Modal Group Meaning	Member Words
Motion Mode	**G0**, G1, G2, G3, G38.2, G38.3, G38.4, G38.5, G80
Coordinate System Select	**G54**, G55, G56, G57, G58, G59
Plane Select	**G17**, G18, G19
Distance Mode	**G90**, G91
Arc IJK Distance Mode	**G91.1**
Feed Rate Mode	G93, **G94**
Units Mode	G20, **G21**
Cutter Radius Compensation	**G40**
Tool Length Offset	G43.1, **G49**
Program Mode	**M0**, M1, M2, M30
Spindle State	M3, M4, **M5**
Coolant State	M7, M8, **M9**

In addition to the G-code parser modes, Grbl will report the active T tool number, S spindle speed, and F feed rate, which all default to 0 upon a reset. For those that are curious, these don't quite fit into nice modal groups, but are just as important for determining the parser state.

$I - View build info

This prints feedback to the user the Grbl version and source code build date. Optionally, $I can also store a short string to help identify which CNC machine you are communicating with, if you have more than machine using Grbl. To set this string, send Grbl $I=xxx, where xxx is your customization string that is less than 80 characters. The next time you query Grbl with a $I view build info, Grbl will print this string after the version and build date.

$N - View startup blocks

$Nx are the startup blocks that Grbl runs every time you power on Grbl or reset Grbl. In other words, a startup block is a line of G-code that you can have Grbl auto-magically run to set your G-code modal defaults, or anything else you need Grbl to do everytime you start up your machine. Grbl can store two blocks of G-code as a system default.
So, when connected to Grbl, type $N and then enter. Grbl should respond with something short like:

```
$N0=
$N1=
ok
```

Not much to go on, but this just means that there is no G-code block stored in line $N0 for Grbl to run upon startup. $N1 is the next line to be run.

$Nx=line - Save startup block

(IMPORTANT: Be very careful when storing any motion (G0/1,G2/3,G28/30) commands in the startup blocks. These motion commands will run everytime you reset or power up Grbl, so if you have an emergency situation and have to e-stop and reset, a startup block move can and will likely make things worse quickly. Also, do not place any commands that save data to EEPROM, such as G10/G28.1/G30.1. This will cause Grbl to constantly re-write this data upon every startup and reset, which will eventually wear out your Arduino's EEPROM.

Typical usage for a startup block is simply to set your preferred modal states, such as G20 inches mode, always default to a different work coordinate system, or, to provide a way for a user to run some user-written unique feature that they need for their crazy project.)

To set a startup block, type $N0= followed by a valid G-code block and an enter. Grbl will run the block to check if it's valid and then reply with an ok or an error:to tell you if it's successful or something went wrong. If there is an error, Grbl will not save it.
For example, say that you want to use your first startup block $N0 to set your G-code parser modes like G54 work coordinate, G20 inches mode, G17 XY-plane. You would type $N0=G20 G54 G17 with an enter and you should see an 'ok' response. You can then check if it got stored by typing $N and you should now see a response like $N0=G20G54G17.
Once you have a startup block stored in Grbl's EEPROM, everytime you startup or reset you will see your startup block printed back to you and a response from Grbl to indicate if it ran okay. So for the previous example, you'll see:

```
Grbl 0.9i ['$' for help]
G20G54G17ok
```

If you have multiple G-code startup blocks, they will print back to you in order upon every startup. And if you'd like to clear one of the startup blocks, (e.g., block 0) type $N0= without anything following the equal sign.
Also, if you have homing enabled, the startup blocks will execute immediately after the homing cycle, not at startup.

$C - Check gcode mode
This toggles the Grbl's gcode parser to take all incoming blocks and process them completely, as it would in normal operation, but it does not move any of the axes, ignores dwells, and powers off the spindle and coolant. This is intended as a way to provide the user a way to check how their new G-code program fares with Grbl's parser and monitor for any errors (and checks for soft limit violations, if enabled).

When toggled off, Grbl will perform an automatic soft-reset (^X). This is for two purposes. It simplifies the code management a bit. But, it also prevents users from starting a job when their G-code modes are not what they think they are. A system reset always gives the user a fresh, consistent start.

$X - Kill alarm lock
Grbl's alarm mode is a state when something has gone critically wrong, such as a hard limit or an abort during a cycle, or if Grbl doesn't know its position. By default, if you have homing enabled and power-up the Arduino, Grbl enters the alarm state, because it does not know its position. The alarm mode will lock all G-code commands until the '$H' homing cycle has been performed. Or if a user needs to override the alarm lock to move their axes off their limit switches, for example, '$X' kill alarm lock will override the locks and allow G-code functions to work again.

But, tread carefully!! This should only be used in emergency situations. The position has likely been lost, and Grbl may not be where you think it is. So, it's advised to use G91 incremental mode to make short moves. Then, perform a homing cycle or reset immediately afterwards.

$H - Run homing cycle
This command is the only way to perform the homing cycle in Grbl. Some other motion controllers designate a special G-code command to run a homing cycle, but this is incorrect according to the G-code standards. Homing is a completely separate command handled by the controller.

TIP: After running a homing cycle, rather jogging manually all the time to a position in the middle of your workspace volume. You can set a G28 or G30 pre-defined position to be your post-homing position, closer to where you'll be machining. To set these, you'll first need to jog your machine to where you would want it to move to after homing. Type G28.1 (or G30.1) to have Grbl store that position. So then after '$H' homing, you could just enter 'G28' (or 'G30') and it'll move there auto-magically. In general, I would just move the XY axis to the center and leave the Z-axis up. This ensures that there isn't a chance the tool in the spindle will interfere and that it doesn't catch on anything.

$RST=$, $RST=#, and $RST=- Restore Grbl settings and data to defaults*

These commands are not listed in the main Grbl $ help message but are available to allow users to restore parts of or all of Grbl's EEPROM data. Note: Grbl will automatically reset after executing one of these commands to ensure the system is initialized correctly.

- $RST=$: Erases and restores the $$ Grbl settings back to defaults, which is defined by the default settings file used when compiling Grbl. Often OEMs will build their Grbl firmwares with their machine-specific recommended settings. This provides users and OEMs a quick way to get back to square-one, if something went awry or if a user wants to start over.
- $RST=# : Erases and zeros all G54-G59 work coordinate offsets and G28/30 positions stored in EEPROM. These are generally the values seen in the $#parameters printout. This provides an easy way to clear these without having to do it manually for each set with a G20 L2/20 or G28.1/30.1 command.
- $RST=* : This clears and restores all of the EEPROM data used by Grbl. This includes $$ settings, $# parameters, $N startup lines, and $I build info string. Note that this doesn't wipe the entire EEPROM, only the data areas Grbl uses. To do a complete wipe, please use the Arduino IDE's EEPROM clear example project.

Real-Time Commands: ~, !, ?, and Ctrl-X

The last four of Grbl's commands are real-time commands. This means that they can be sent at anytime, anywhere, and Grbl will immediately respond, no matter what it's doing. For those that are curious, these are special characters that are 'picked-off' from the incoming serial stream and will tell Grbl to execute them, usually within a few milliseconds.

~ - *Cycle start*

This is the cycle start or resume command that can be issued at any time, as it is a real-time command. When Grbl has motions queued in its buffer and is ready to go, the ~ cycle start command will start executing the buffer and Grbl will begin moving the axes. However, by default, auto-cycle start is enabled, so new users will not need this command unless a feed hold is performed. When a feed hold is executed, cycle start will resume the program. Cycle start will only be effective when there are motions in the buffer ready to go and will not work with any other process like homing.

! - Feed hold

The feed hold command will bring the active cycle to a stop via a controlled deceleration, so as not to lose position. It is also real-time and may be activated at any time. Once finished or paused, Grbl will wait until a cycle start command is issued to resume the program. Feed hold can only pause a cycle and will not affect homing or any other process.

If you need to stop a cycle mid-program and can't afford losing position, perform a feed hold to have Grbl bring everything to a controlled stop. Once finished, you can then issue a reset. Always try to execute a feed hold whenever the machine is running before hitting reset, except of course if there is some emergency situation.

? - Current status

The ? command immediately returns Grbl's active state and the real-time current position, both in machine coordinates and work coordinates. Optionally, you can also have Grbl respond back with the RX serial buffer and planner buffer usage via the status report mask setting. The ? command may be sent at any time and works asynchronously with all other processes that Grbl is doing. The $13 Grbl setting determines whether it reports millimeters or inches. When ? is pressed, Grbl will immediately reply with something like the following:

```
<Idle,MPos:5.529,0.560,7.000,WPos:1.529,-5.440,-0.000>
```

The active states Grbl can be in are: **Idle, Run, Hold, Door, Home, Alarm, Check**

- **Idle**: All systems are go, no motions queued, and it's ready for anything.
- **Run**: Indicates a cycle is running.
- **Hold**: A feed hold is in process of executing or slowing down to a stop. After the hold is complete, Grbl will remain in Hold and wait for a cycle start to resume the program.
- **Door**: (New in v0.9i) This compile-option causes Grbl to feed hold, shut-down the spindle and coolant, and wait until the door switch has been closed and the user has issued a cycle start. Useful for OEM that need safety doors.
- **Home**: In the middle of a homing cycle. NOTE: Positions are not updated live during the homing cycle, but they'll be set to the home position once done.
- **Alarm**: This indicates something has gone wrong or Grbl doesn't know its position. This state locks out all G-code commands but allows you to interact with Grbl's settings if you need to. '$X' kill alarm lock releases this state and puts Grbl in the Idle state, which will let you move things again. As said before, be cautious of what you are doing after an alarm.
- **Check**: Grbl is in check G-code mode. It will process and respond to all G-code commands, but not motion or turn on anything. Once toggled off with another '$C' command, Grbl will reset itself.

Ctrl-x - Reset Grbl

This is Grbl's *soft* reset command. It's real-time and can be sent at any time. As the name implies, it resets Grbl, but in a controlled way, *retains* your machine position, and all is done without powering down your Arduino. The only times a soft-reset could lose position is when problems arise, and the steppers were killed while they were moving. If so, it will report if Grbl's tracking of the machine position has been lost. This is because an uncontrolled deceleration can lead to lost steps, and Grbl has no feedback to how much it lost (this is the problem with steppers in general). Otherwise, Grbl will just re-initialize, run the startup lines, and continue on its merry way.

Please note that it's recommended to do a soft-reset before starting a job. This guarantees that there aren't any G-code modes active that from playing around or setting up your machine before running the job. So, your machine will always starts fresh and consistently, and your machine does what you expect it to.

4.7 GRBL Calibration and Final Values for CCR:

a) Magnification Calibration:

$100 parameter in Universal G-code sender- (x, step/mm)

$101 parameter in Universal G-code sender- (y, step/mm)

This defines the magnification factor of the image to be drawn.

$100 and $101 are the parameters defined and calculated by Diameter of the pulley, number of steps in one rotation of a stepper motor.

For the given practical instance, pulley diameter = 33.615013mm,

Number steps per revolution = 200 (equivalent to 360degree) = 33.615013mm

1step = 1.8degree = 33.615013*1.8/360 = 0.1681

1mm = 1/0.1681 steps = 5.9488 (which does not work)

However, after actual trial and error of drawing, drawing is perfect at factors,

$100=1.414 (x, step/mm)

$101=1.414 (y, step/mm)

G-code for a circle with Diameter = 100mm generated via J tech laser cutting tool extension in Inkscape.

M03 S0 (servo up)

G90 (To go into absolute distance mode, program G90.) ??

G21 (Unit selection of mm)

G1 F3000 (set the feed rate to 3000)

G1 X194.8819 Y106.2992 (linear move towards start point of the drawing)

G4 P0 (Dwell 0)

M05 S8 (Servo Down)

G4 P0 (Dwell 0) (Till this point, pen comes from origin to the point where it starts drawing and downs the servo)

Circle Drawing begins now:

G1 F500.000000 (set the feed rate to 500)

G2 X168.9366 Y43.6618 I-88.5827 J0. (Circular arm moves in clockwise)

G2 X106.2992 Y17.7165 I-62.6374 J62.6374

G2 X43.6618 Y43.6618 I-0. J88.5827

G2 X17.7165 Y106.2992 I62.6374 J62.6374

G2 X24.4595 Y140.1983 I88.5827 J0.

G2 X43.6618 Y168.9366 I81.8397 J-33.8991

G2 X72.4001 Y188.1389 I62.6374 J-62.6374

G2 X106.2992 Y194.8818 I33.8991 J-81.8397

G2 X140.1983 Y188.1389 I-0. J-88.5827

G2 X168.9366 Y168.9366 I-33.8991 J-81.8397

G2 X188.1389 Y140.1983 I-62.6374 J-62.6374

G2 X194.8819 Y106.2992 I-81.8397 J-33.8991

G1 X194.8819 Y106.2992 (linear move towards 0 0)

G4 P0

M03 S0

G1 F3000

G1 X0 Y0

Command M05- Servo down (in writing condition)

M03- Servo Up (not in writing condition)

Current System Compatibility:

$0=10 (step pulse, usec)
$1=0 (step idle delay, msec)
$2=0 (step port invert mask:00000000)
$3=0 (dir port invert mask:00000000)
$4=0 (step enable invert, bool)
$5=0 (limit pins invert, bool)
$6=0 (probe pin invert, bool)
$10=3 (status report mask:00000011)
$11=0.010 (junction deviation, mm)
$12=0.010 (arc tolerance, mm)
$13=0 (report inches, bool)
$20=0 (soft limits, bool)
$21=0 (hard limits, bool)
$22=1 (homing cycle, bool)
$23=0 (homing dir invert mask:00000000)
$24=25.000 (homing feed, mm/min)
$25=500.000 (homing seek, mm/min)
$26=250 (homing debounce, msec)
$27=1.000 (homing pull-off, mm)
$100=1.414 (x, step/mm)
$101=1.414 (y, step/mm)
$102=80.000 (z, step/mm)
$110=5000.000 (x max rate, mm/min)
$111=5000.000 (y max rate, mm/min)
$112=5000.000 (z max rate, mm/min)
$120=50.000 (x accel, mm/sec^2)
$121=50.000 (y accel, mm/sec^2)
$122=30.000 (z accel, mm/sec^2)
$130=300.000 (x max travel, mm)
$131=300.000 (y max travel, mm)
$132=200.000 (z max travel, mm)