

[Home](#) / [Tutorials](#) / How to build an 2-axis Arduino CNC Gcode Interpreter

Posted on [August 27, 2013](#)

How to build a 2-axis Arduino CNC Gcode Interpreter



Written by

Dan

Posted in

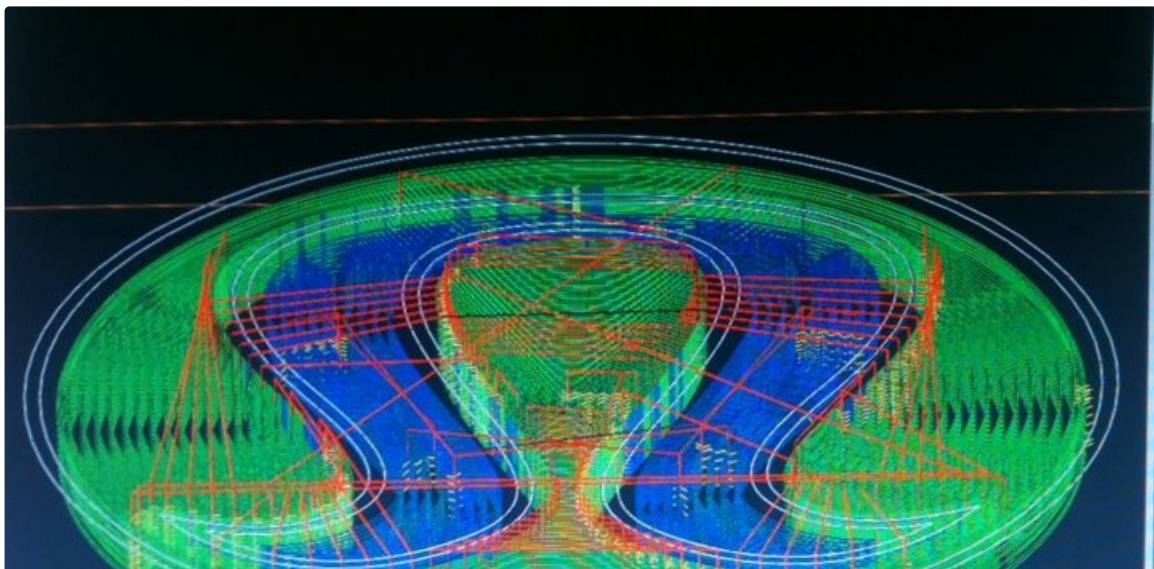
Tutorials

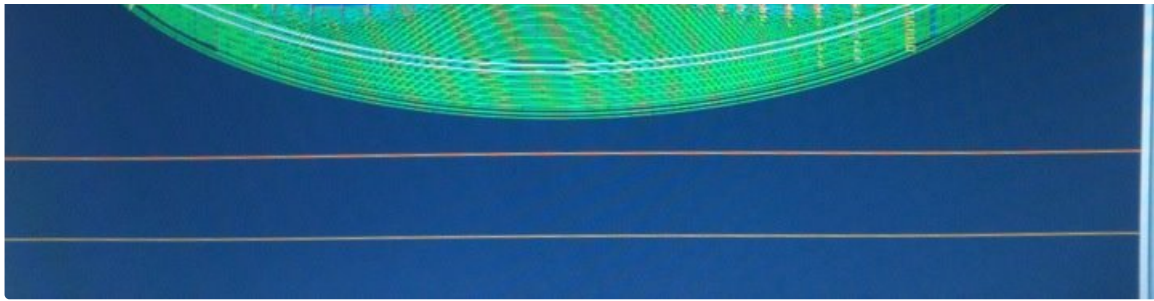
Tagged

arduino, cnc, gcode, interpreter

Comments

19 Comments





Purpose

3D printers, laser cutters, water jet cutters, robot arms, delta robots, stewart platforms, the Makelangelo: all of these are examples of *computer numerical control* (CNC) machines. CNC machines need to move accurately and on command. Stepper motors are a great way to move accurately – they move a predictable amount and then stay where you put them. To command the stepper motors we need a way to easily turn our human desires into machine instructions into stepper motor steps. In effect we need our robot brain to be an *interpreter*. I'm going to show you a simple interpreter written for Arduino that lets you move stepper motors for your robots.

Audience

This tutorial is meant for people who have an Arduino and understand the basics of programming in C. That means variables, methods, while(), if(), switch(), and manipulating strings. There's even a few ?: style if statements.

History

In the very beginnings of computers the programmers talked to the machines in machine languages – the ones and zeros of binary. They memorized the binary machine language and wrote programs in that machine language. Each program probably did one small job because they were so hard to write.

Then somebody got smart and wrote the first interpreter – a binary program that could turn a more human language (assembly) into binary. Every language since then has been a different flavor of the same ice cream: trying to find easier ways of letting humans tell machines what to do. Most programs today are written using interpreters that were built using interpreters that were built using the original interpreters. Ours will be, too. So Meta!

Goal

I can't build something without a clearly defined goal. I need to see the target in order to hit it.

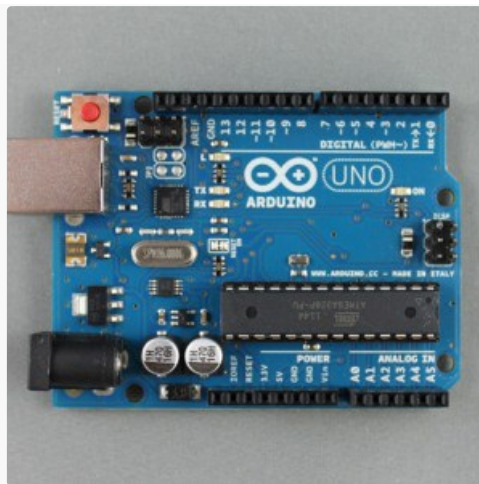
I could write a 10-line program that does one pattern of motor movements but then – just like the binary

programmers of yore – I would have to write a new program for every pattern. Compiling, uploading, and testing a program is a time-consuming process. My goal is to write an interpreter that can listen and respond in real time to any pattern.

Before you can run an Arduino program you have to compile and upload it. When our program is done you won't have to compile anything. You will send the gcode to the Arduino through the serial connection and the Arduino will listen, understand, and obey.

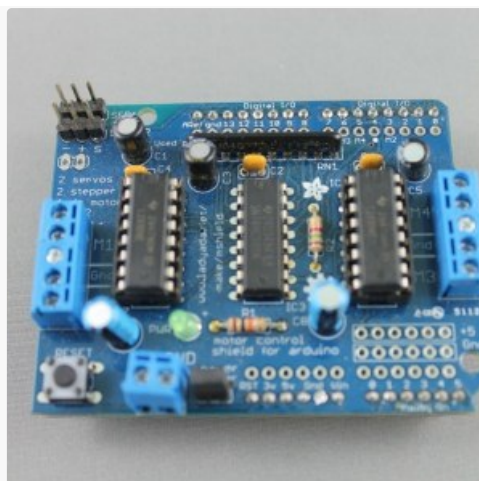
Hardware

My **NEMA 17 stepper motors** are controlled by an **L293 motor shield** riding on top of an **Arduino UNO**. On the motor shield I attached a **female power plug** so I could easily plug in a **12v power supply**. I put a piece of tape on the motor shaft of each stepper so I can easily see it moving. Later I can replace that with pulleys and belts.



CAD \$32.99

Add to cart



NRSC 100

CAD \$18.33

Add to cart



NRSC 100

CAD \$12.45

Add to cart



NRSC 100

CAD \$24.67

Add to cart

For the sake of simplicity I'm going to assume that our motors are moving *orthogonal* to each other. That means on a grid that one motor moves along the X axis and one moves on the Y axis, also known as a **cartesian coordinate system**. (Delta robots, stewart platforms, and robot arms all use much more complicated systems to arrive at the same effect.)

Because we're moving in a grid I can use Bresenham's Line Algorithm to move each motor at same time with different relative speeds. That means in our cartesian system I can draw lines from anywhere to anywhere and they'll be nice and straight. I can draw curves by chopping the curve into lots of tiny lines that approximate the curve shape.

Method

The Arduino will need to:

- Setup:
 - start listening to the serial connection
 - set up the stepper motors
 - Tell whoever is listening that we're ready for more instructions.
- Loop forever:
 - Wait for a message
 - Read the message
 - Interpret the meaning
 - Act on that meaning
 - Tell whoever is listening that we're ready for more instructions.

Code

The messages I send to the Arduino could be in any language. I could even make up a language! To keep life simple I'm going to use a language other people know and understand, called *gcode*.

The rules of gcode – the punctuation, syntax, grammar, and vocabulary – are very easy to explain to a machine. Gcode commands all consist of an uppercase letter followed by a number. Here are the codes I'm going to build into the interpreter.

COMMAND	MEANING
G00 [X(number)] [Y(number)] [F(number)] G01 [X(number)] [Y(number)] [F(number)]	Absolute mode: Move in a line to (X,Y) at speed F Relative Mode: Move (X,Y) amount at speed F

G04 P(number) COMMAND	Do nothing for P seconds MEANING
G90	absolute mode
G91	relative mode
G92 [X(number)] [Y(number)]	change logical position
M18	turn off power to motors
M100	print out instructions for the human
M114	report position and feedrate

- Every (*number*) is assumed to be a *float* – a number that might have a decimal place and an exponent. 2.015e-5 is a float.
- Anything in [brackets] is optional.
- G and M commands cannot be combined in a single instruction.
- Arduino software has a nice way to send messages to the PCB through the *serial interface window*. It's the magnifying glass on the right hand side of the Arduino window. Unfortunately the serial interface window doesn't send the return key (\n) to the PCB. Instead of return I'm going to use semicolon (;) to tell the machine "this is the end of an instruction".

Step

```
#define BAUD (57600) // How fast is the Arduino talking?
#define MAX_BUF (64) // What is the longest message Arduino can store?

char buffer[MAX_BUF]; // where we store the message until we get a ';'
int sof far; // how much is in the buffer

/**
 * First thing this machine does on startup. Runs only once.
 */
void setup() {
  Serial.begin(BAUD); // open coms
  help(); // say hello
  set_feedrate(200); // set default speed
  ready();
}
```



```

/**
 * display helpful information
 */
void help() {
  Serial.print(F("CNC Robot "));
  Serial.println(VERSION);
  Serial.println(F("Commands:"));
  Serial.println(F("G00 [X(steps)] [Y(steps)] [F(feedrate)]; - linear move"));
};
  Serial.println(F("G01 [X(steps)] [Y(steps)] [F(feedrate)]; - linear move"));
};
  Serial.println(F("G04 P[seconds]; - delay"));
  Serial.println(F("G90; - absolute mode"));
  Serial.println(F("G91; - relative mode"));
  Serial.println(F("G92 [X(steps)] [Y(steps)]; - change logical position"));
  Serial.println(F("M18; - disable motors"));
  Serial.println(F("M100; - this help message"));
  Serial.println(F("M114; - report position and feedrate"));
}

/**
 * prepares the input buffer to receive a new message and
 * tells the serial connected device it is ready for more.
 */
void ready() {
  sofars=0; // clear input buffer
  Serial.print(F("> ")); // signal ready to receive input
}

```

The only mystery here should be `F()`, a special Arduino-only macro. It tells the compiler to put the string in program memory instead of RAM, which can sometimes be the difference between a program that fits on an Arduino and a program that doesn't.

loop

```

/**
 * After setup() this machine will repeat loop() forever.
 */
void loop() {
  // listen for commands
  if( Serial.available() ) { // if something is available

```

```

char c = Serial.read(); // get it
Serial.print(c); // optional: repeat back what I got for debugging

// store the byte as long as there's room in the buffer.
// if the buffer is full some data might get lost
if(sofar < MAX_BUF) buffer[sofar++]=c;
// if we got a return character (\n) the message is done.
if(c=='\n') {
    Serial.print(F("\r\n")); // optional: send back a return for debugging

    // strings must end with a \0.
    buffer[sofar]=0;
    processCommand(); // do something with the command
    ready();
}
}
}

```

I tend to write my comments first, then I write the code between the comments. That way I get my ideas organized and the code is better documented when I'm done.

Handling Commands

```

/**
 * Read the input buffer and find any recognized commands. One G or M command per line.
 */
void processCommand() {
    // look for commands that start with 'G'
    int cmd=parsenumber('G',-1);
    switch(cmd) {
    case 0: // move in a line
    case 1: // move in a line
        set_feedrate(parsenumber('F',fr));
        line( parsnnumber('X',(mode_abs?px:0)) + (mode_abs?0:px),
            parsnnumber('Y',(mode_abs?py:0)) + (mode_abs?0:py) );
        break;
    // case 2: // clockwise arc
    // case 3: // counter-clockwise arc
    case 4: pause(parsenumber('P',0)*1000); break; // wait a while
    case 90: mode_abs=1; break; // absolute mode
    case 91: mode_abs=0; break; // relative mode

```



```

case 91: mode_abs=0; break; // relative mode
case 92: // set logical position
    position( parsenumber('X',0),
    parsenumber('Y',0) );
    break;
default: break;
}

// look for commands that start with 'M'
cmd=parsenumber('M',-1);
switch(cmd) {
case 18: // turns off power to steppers (releases the grip)
    m1.release();
    m2.release();
    break;
case 100: help(); break;
case 114: where(); break; // prints px, py, fr, and mode.
default: break;
}

// if the string has no G or M commands it will get here and the Arduino will
silently ignore it
}

```

parsenumber(key,default) searches for the letter 'key' in buffer. If it finds key it return the number that follows immediately after. If it doesn't find key it returns 'default'.

Drawings

I first learned about Bresenham's line algorithm from **André LaMothe** in one of his books back in the early 90's. I think it was "The Black Art of 3D Game Programming"? It's supposed to be used for drawing graphics on a computer screen. It works just as well for our purposes, and it can be extended to any number of motors all moving at once.

The slope of a line can be expressed as dx/dy . Let's pretend for a moment that dx/dy is $1/3$. Three steps on Y equals one step on X. Bresenham's algorithm starts stepping along Y and adds dx to a counter. when the counter $\geq Y$, step once on X. Here's the best part: by using the slope of the entire line we can do all the math with whole numbers, which means we shouldn't get any rounding errors and our Arduino can do the math extra fast. Whole number math nearly always faster than floating point math.

```

/**
 * Uses Bresenham's line algorithm to move both motors
 * @input newx the destination x position

```

```

* @input newy the destination y position
**/

void line(float newx,float newy) {
    long dx=newx-px; // distance to move (delta)
    long dy=newy-py;
    int dirx=dx > 0?1:-1; // direction to move
    int diry=dy > 0?1:-1;
    dx=abs(dx); // absolute delta
    dy=abs(dy);

    long i;
    long over=0;

    if(dx > dy) {
        for(i=0;i < dx;++i) {
            m1.onestep(dirx);
            over+=dy;
            if(over>=dx) {
                over-=dx;
                m2.onestep(diry);
            }
            pause(step_delay); // step_delay is a global connected to feed rate.
            // test limits and/or e-stop here
        }
    } else {
        for(i=0;i < dy;++i) { m2.onestep(diry); over+=dx; if(over>=dy) { over-=d
y; m1.onestep(dirx); } pause(step_delay); // step_delay is a global connecte
d to feed rate. // test limits and/or e-stop here } } // update the logical
position. We don't just = newx because // px + dx * dirx == newx could be fa
lse by a tiny margin and we don't want rounding errors. px+= dx*dirx; py+= d
y*diry; } /** * delay for the appropriate number of microseconds * @input ms
how many milliseconds to wait */ void pause(long ms) { delay(ms/1000); dela
yMicroseconds(ms%1000); // delayMicroseconds doesn't work for values > ~16k.
} /** * Set the feedrate (speed motors will move) * @input nfr the new spee
d in steps/second */ void set_feedrate(float nfr) { if(fr==nfr) return; // s
ame as last time? quit now. if(nfr > MAX_FEEDRATE || nfr < MIN_FEEDRATE) { /
/ don't allow crazy feed rates

    Serial.print(F("New feedrate must be greater than "));
    Serial.print(MIN_FEEDRATE);
    Serial.print(F("steps/s and less than "));
    Serial.print(MAX_FEEDRATE);
    Serial.println(F("steps/s.));
    return;
}

step_delay = 1000000.0/nfr;

```

```
fr=nfr;  
}
```

Sure

Want the entire source in one file, ready to compile? [Here you go](#). I use a version of this code in nearly all Marginally Clever robots.

Edit 2014-09-29: I've since added examples for Adafruit Motor Shield v1, Adafruit Motor Shield v2, RUMBA, and RAMPs.

Vide

This is an updated version that drives 4 steppers at once. It could be done with even more.

Noty

- Add a `parsenumber('N')` for line numbers. Then your machine can detect if it receives lines in the wrong order, or request specific lines from the computer. You may also need to understand M110 N[number] so the computer can set the line number.
- Add a ';' on the end of every command, followed by a checksum. Then the arduino can tell if the message is correctly received.
- Add G02 and G03 for curved lines.

Filaments

So there you have it. In 293 lines of code we've built a really simple CNC machine Gcode interpreter that handles six G commands and three M commands.

If you'd like me to go into more detail about **how to make arcs**, or something else then please make a comment below and I'll post about it.

Please share videos & pictures of your creations to **the forums**.

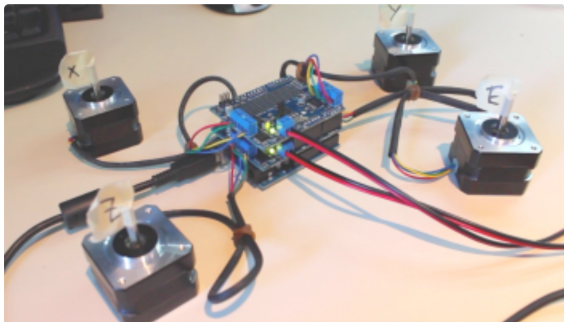
SoAb

How to choose your controller electronics to build your first CNC.

erds



tal eR



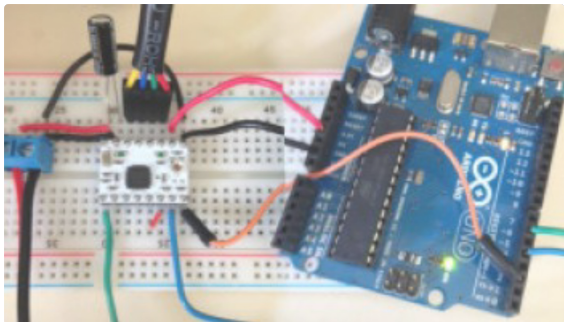
How to Build a 4-axis CNC Gcode Interpreter for Arduino

September 3, 2013
In "Tutorials"

How do I choose controller electronics for my first CNC?

Tony asks "How do I choose controller electronics for my first 2D DIY CNC?" What's a CNC? The Makelangelo, Delta robots, Stewart platforms, robot arms, CoreXY CNCs, and traditional CNCs are all fundamentally the same. A CNC is a Computer Numerical Control device, a machine for moving precisely in one...

January 7, 2015
In "Tutorials"



How To move a Stepper Motor with an A4988 driver and an Arduino

February 29, 2016
In "Tutorials"

“
The only
Cooler”

How to add
coordinates

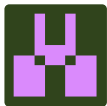


adnankhan

December 13, 2015

how we add the coordinates in this program?

Reply ↩

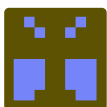


Muhammad Atif

January 9, 2016

Write G code like
G00 X100 Y200 F200

Reply ↩



Lucas

January 23, 2016

I do not understand where parsenumber is to be declared.

Reply ↩

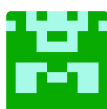


Mali

February 2, 2016

Hello, thanks for supporting the cheap AMS v1 for Gcode. Your code is the only one I could find working. However it seems like when gcode has lxxx and Jxxx values for curved lines it seems not responding. I dont know if this is my problem or line drawing is the only mod software can parse. Thanks in advance.

Reply ↩



Marcus Wolschon

March 29, 2016

thanks.
I was looking fot super-simple g-code parsing code for a robot arm (to avoid the evening of coding my own)
here I just need to remove these Besenham code since my SPI stepper controllers accept target-positions and not just step+dir

Reply ↩



Nikodem

February 2, 2017

Hi,
Nice tutorial. I am writing my own 3D printer program I am just not sure about number rounding. I can't find it in your code, I also can't find place where you calculate from mm to actual steps. Thanks in advance for

help.
Best
Nikodem

Reply ↩



Dan 

February 5, 2017

I believe I only use floor() in a few places.

Reply ↩

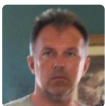


Márcio Pessoa

February 2, 2017

Congratulations Dan! Very usefull.

Reply ↩



Rick

February 7, 2017

A tutorial on arcs and spirals would be nice.

Reply ↩



Dan 

February 14, 2017

<https://www.marginallyclever.com/2014/03/how-to-improve-the-2-axis-cnc-gcode-interpreter-to-understand-arcs/> ?

Reply 



shan

February 16, 2017

thanks for this sir its very helpful for students like me thanks again

Reply 



Deli DD

March 5, 2017

Very useful, but may I ask, is there any literature on how to add acceleration to this code? I am making open source-hobby stepper library. I actually got implemented trapezoidal acceleration on linear movement, but I cannot find out how to implement it for arcs. Thanks for any ideas

Reply 



Dan 

March 6, 2017

this conversation could be long, and should be done in the forums, please.

Reply 



Daniel R

March 10, 2017

Deli DD, may I ask how you managed to implement trapezoidal acceleration?

Also, I am having trouble trying to understand feed rates on the bresenham algorithm. Can anyone recommend any kind of literature on that?

Thanks!

Reply 



Dan 

March 13, 2017

It's a great question that's probably better handled in the forums, because it could be a long conversation not relevant to this tutorial.

Reply 



Jesus

March 10, 2017

I would like to thank you for this post it is very clear to understand! Great job!!
Just one question: what can I use to send the GCode? Is it compatible with Universal GCode Sender?
Thank in advance!

Reply ↩



Dan 

March 13, 2017

It might be compatible. I haven't tested it recently. Please post your results to the forums and I'll update the post with your results.

Reply ↩



Ross

May 9, 2017

I am able to get this program to work on my arduino Mega, but i would like to run it on an Arduino DUE, so i can combine it with my existing program calculating non cartesian coordinates.

When i load it onto my arduino DUE the serial monitor only responds to the first command, then stops...
What can i do?

Reply ↩



Dan 

May 12, 2017

Discuss it in the support forums? We don't officially support DUE because we have none in stock. If you can get it working and submit a github pull request, we'll include your changes in the official firmware along with your name.

Reply ↩

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment



Please upgrade to a [supported browser](#) to get a reCAPTCHA challenge.

Alternatively if you think you are getting this page in error, please check your internet connection and reload.

[Why is this happening to me?](#)

Name *

Email *

Website






Post Comment

☐ Notify me of follow-up comments by email.






☐ Notify me of new posts by email.

 Search ...

Recent Posts

-  [Where to Start with Robotics](#)
-  [Build your own falling block game like Tetris](#)
-  [How to use an 8×8 LED grid with Arduino](#)
-  [Shift registers and seven segment displays in Arduino](#)
-  [How to setup NodeMCU drivers and Arduino IDE](#)

Recent Comments

-  [Dan](#) on [An Easier Way to Align Laser Cutter Mirrors](#)
-  [Jeff](#) on [An Easier Way to Align Laser Cutter Mirrors](#)
-  [Dan](#) on [How to build an 2-axis Arduino CNC Gcode Interpreter](#)
-  [Ross](#) on [How to build an 2-axis Arduino CNC Gcode Interpreter](#)
-  [Dan](#) on [How to control a linear actuator with an Arduino](#)

In stock orders placed before 1pm PST are shipped the next business day.

Join our newsletter

You give email, we give news.

Email

Subscribe

Follow Us









Follow us secretly

RSS - Posts

RSS - Comments

More info

-  [My Account](#)
-  [Order Tracking](#)
-  [Privacy](#)
-  [Frequently Asked Questions](#)
-  [Terms of Service](#)
-  [Jobs Now Hiring](#)

© Marginally Clever Robots 2017

Storefront designed by **WooCommerce**.

