Arduino GCode Interpreter

From RepRapWiki

This page describes something which is no longer the most recent version. For the replacement version see: ***G-code***

This page has been flagged as containing duplicate material that Darwin/Arduino GCode Interpreter also attempts to cover.
These pages should be merged such that both pages do not attempt to cover the duplicate topics.

Contents

Arduino G-Code Interpreter

This page has now been superseded. For the up-to-date one, see here.

Introduction

G-Code is a commonly use language to control CNC machines. G-Code is a light-weight, simple to parse format, designed to be able to run on modest hardware, such as the Arduino loaded with this firmware (in

this case). It is a format that is prepared in advance and created from the digital design files (CAD files).

Several software packages can generate G-Code, so using this firmware allows you a degree of flexibility.

Files

The G-Code firmware source is available from SourceForge as part of the RepRap Arduino firmware package (http://sourceforge.net/project/showfiles.php?group_id=159590&package_id=256565) . Make sure you get the latest version.

Installation

Once you download the proper files, there are a couple steps you need to do:

1. Copy the folders in **reprap-arduino-firmware-x.y/library** to **arduino-00xx/hardware/libraries**
2. Open the GCode_Interpreter sketch in **reprap-arduino-firmware-x.y/gcode/GCode_Interpreter /GCode_Interpreter.pde** with the Arduino software.
3. Configure the firmware (see below)
4. Upload the sketch to your Arduino.

Ubuntu users will also have to upgrade their avr-gcc; the standard one contains a bug. See this link (http://blog.coldtobi.de/1_coldtobis_blog/archive/21_updating_avr-gcc_binutils_to_get_the_devices_update_now_also_includes_deb_files.html) for details.

Usage

Firmware Configuration

In order for the firmware to operate properly, you must configure the proper variables in the firmware and then upload the firmware to your Arduino. The values are stored in the file _init.pde. An example version is included in the distribution called _init.pde.dist.

We'll cover each of the variables below:

X_STEPS_PER_INCH

This variable stores how many steps to take to move the X axis 1 inch. You will need to set this as accurately as possible if you want your machine to be accurate. There are two ways to set it:

1. Move and Measure - slap a pen or marker on as a toolhead and draw a 1000 step line. Measure it and divide 1000 by the length in inches.
2. Calculate Step Size - this one is the preferred way of doing things. Its rather easy to calculate step size based on your drive mechanism.

For threaded rod drive systems:

Find your TPI (threads per inch). for example, 1/4"-20 threaded rod means that there are 20 threads per inch (aka 20 turns = 1 inch.) Simply take that number and multiply it by the steps in a revolution. With a 400 step motor, it would be 8000 steps per inch.

For belt/pulley systems:

1. Find the circumference of your drive pulley. (remember circumference = 2*pi*r) (say: 2.75")
2. Calculate step size (ie: circumference / steps per revolution) (say: 2.75" / 400 = 0.00625")
3. Divide 1 inch by step size (1" / 0.00625" = 160 steps/inch)

X_STEPS_PER_MM

This variable stores how many steps to take to move the X axis 1mm. You can either calculate it independently, or take the number above and divide by 25.4.

X_MAX_SPEED

This variable stores the maximum speed of the stepper in RPM (revolutions per minute). This is important as it determines the fastest speed to move the stepper. Start low and work your way up if you are unsure of the proper speed.

X_MOTOR_STEPS

This variable stores the number of steps per revolution. This is important as it factors into how many steps to take for a line, as well as how fast your stepper will move. Your stepper will have a fixed number of steps per revolution and should say it on the datasheet.

If you are driving your stepper in half-stepping mode, double the number of steps. Similarly, if you are using a microstepper driver, multiply the steps by the appropriate factor. Make sure you use this new step number when you are calculating steps per inch/mm.

Y_*, Z_*

These variables are the same as the variables above, but allow for different drive systems on the Y and Z axes respectively. You must still fill them out, even if all drive systems are the same.

FAST_XY_FEEDRATE and FAST_Z_FEEDRATE

These define the maximum feedrates used for the G0 command. The feedrates are given in mm/minute.

Sending Commands

The Arduino firmware waits for commands on the serial port and will start processing a command after either encountering a newline, or when there are no more characters to read.

There are several ways to control the machine through the G-Code Firmware:

- You can either write your own custom host software to send commands,
- Use Zach Smith's [ReplicatorG|http://replicat.org/ ReplicatorG (http://replicat.org/)]
- use the RepRap host software to send commands,
- or use the Processing app that we've bundled to send commands.

Implementation

Due to the limited size and processing power available on the Arduino, only a limited subset of G-Code has been implemented. However, the vast majority of uses of G-Code are limited to these commands. Below, we cover the implemented commands as well as limitations.

G-Codes

G0 - Rapid Motion

Implemented - only supports X, Y, and Z axes.

G1 - Coordinated Motion

Implemented - only supports X, Y, and Z axes.

G2 - Arc - Clockwise

Implemented

G3 - Arc - Counter Clockwise

Implemented

G4 - Dwell

Impemented.

G20 - Inches as units

Implemented.

G21 - Millimeters as units

Implemented.

G28 - Go Home

Implemented.

G30 - Go Home via Intermediate Point

Implemented.

G90 - Absolute Positioning

Implemented.

G91 - Incremental Positioning

Implemented.

G92 - Set current as home

Implemented. =)

M Codes

See: MCodeReference

TODO

- Explore implementing offsets.

Bugs

- None so far, but they are lurking.