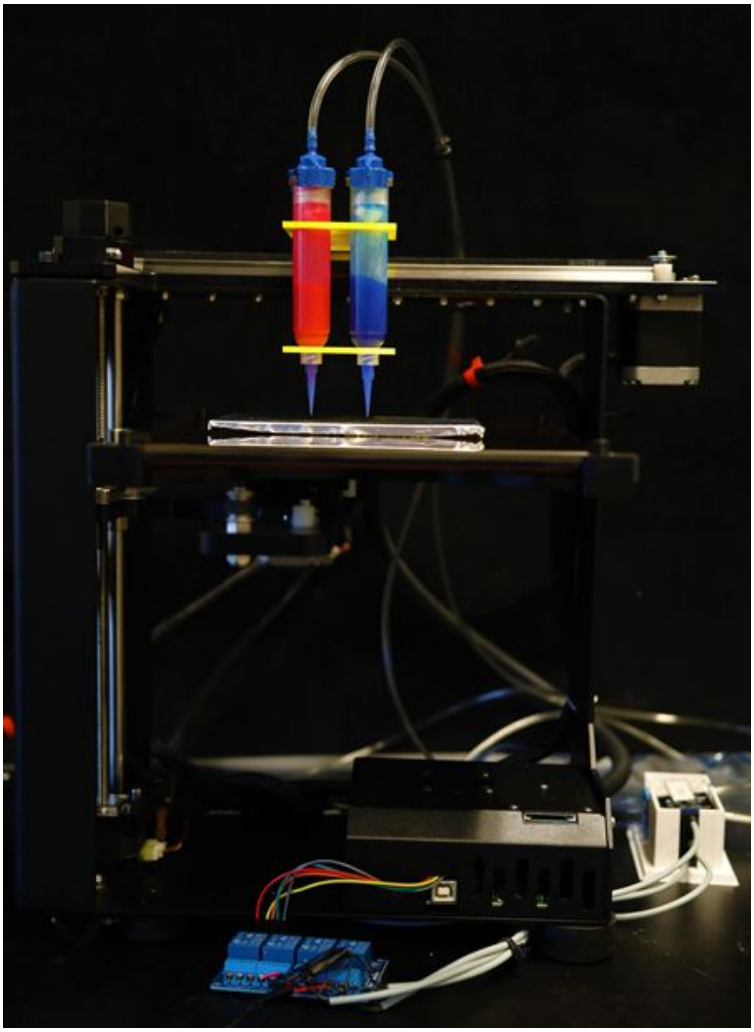
User Guide for the Makergear M2 Pneumatic Control System [M2PCS]



Version 1.2

December 13th, 2018

Architected Materials Lab
University of Pennsylvania
Matthew Sorna

TABLE OF CONTENTS

INTRODUCTION

Natural materials combine desirable mechanical and physical properties (such as high stiffness, strength, and toughness, but low density) which are difficult to simultaneously achieve in synthetic materials. In recent years, much has been learned about how these properties are attained in natural materials, linking performance directly to complex, heterogeneous, multi-material microstructures.

Our current fabrication techniques in the Architected Materials Laboratory are limited by our necessity to design continuous print paths, as well as only being able to extrude a single material. Thus, we can see a need to have control of the actuation of multiple material channels, giving us the much needed start-stop functionality. Included is this user guide is the instructions for how to operate the *Makergear M2 Pneumatic Control System (M2PCS)*, designed to address the barriers found in our current fabrication techniques.

INITIALIZATION AND SETUP

TURNING ON THE M2

To begin printing with the M2PCS, you first need to make sure the printer itself is turned on and initialized. Next to the PC controlling the ShopBot is a 24V Power Supply. This is what powers the M2. At the back of the power supply is an I/O switch. Make sure this is set to on. They'll be some ways you can verify the printer is on. Upon power-up, the fan attached to the electronics housing (located under the print bed) will begin to whir. Additionally, red and green lights should be visible from within the electronics housing.

SETTING UP THE SERIAL CONNECTION TO THE M2

The M2 receives GCode commands serially via a USB connection from a host PC. Located at the front of the electronics housing is a USB printer cable connection. Connect a USB to printer connector cable from the M2 to any USB port on your host PC.

DETERMINING THE CORRECT SERIAL PORT

USING WINDOWS AS A HOST PC

With the USB cable connected from the M2 to your Windows PC, navigate to the *Device Manager*. Under the section labeled *Ports (COM & LPT),* make note of any USB devices listed. Then, disconnect the USB connection to the M2, and again, make note of any changes to the COM list. After connecting the USB cable back to your PC, you should see the creation of COM device, with some name followed by (COM#), with some #, like COM3, COM6 etc.

Once this COM port has been properly identified, make note of this. This will be the “address” used when looking to communicate with the printer using the M2PY module in Python.

You’ll also want to right-click on the COM port and select *Properties* to adjust a few settings. Under the tab *Port Settings,* you want to change *Bits per second* to 115200, and hit OK to save these changes.

USING MAC AS A HOST PC

The majority of the information found in this document is making the assumption that you’ll be connecting to the M2PCS from a Windows PC. Some of the only differences you’ll have to keep in mind when using is Mac will be outlined below.

You’ll want to know the name of the serial port assigned to the Makergear. Instead of a name like COM6, the serial ports of a Mac will look something like: `/dev/tty.usbserial-A6004byf` with everything after the (tty.) is unique, and used to that specific port.

In order to determine the name of your connected serial ports, open up Terminal and enter the following command:

```
ls /dev/tty.*
```

You’ll want to remember the entire string, including `’/dev/tty.’`
This will be what you use for your string for the COM parameter in mp.mopen, described in detail in later sections. A baud speed of **115200** was still found to work just fine on a Mac as well.

INITIALIZING THE PYTHON ENVIRONMENT

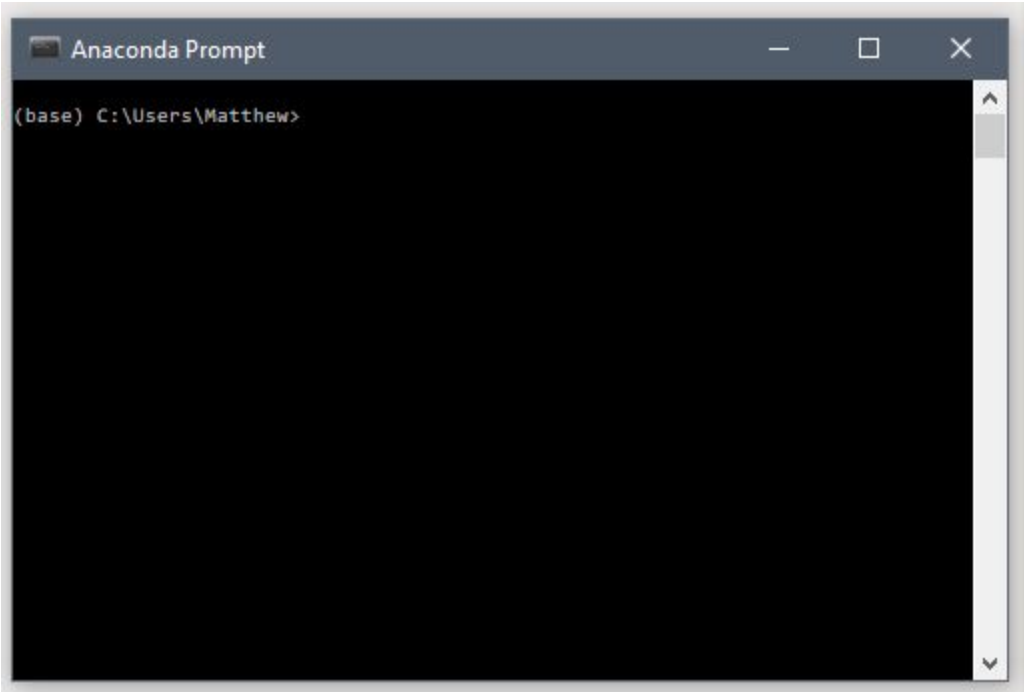
Control of the printer via serial commands using the M2PY Python module is done through *Spyder*, an IDE (integrated development environment) housed under the free-to-use software distribution, *Anaconda (https://www.anaconda.com/download/).*

You'll need to install the Python 3.6 version, found at the link above (specific for your operating system). All standard installation options are fine to use.

IMPORTING THE PYSERIAL MODULE TO ANACONDA

With Anaconda installed, we'll need to import the Python module, PySerial (more info at https://pythonhosted.org/pyserial/).

To do this, assuming Anaconda has been properly installed, use the Start menu and open up the program *Anaconda Prompt*.



With this prompt open, type or copy in the following text:

```
conda install -c anaconda pyserial
```

And hit Enter. Follow the onscreen instructions to complete the installation. Once the installation process is taking place, you'll know it's done when the prompt returns to its initial state:

```
(base) C:\(your dir)>
```

and is awaiting your input. You can now close the prompt.

INSTALLING THE CURRENT VERSION OF M2PY

The most recent version of the M2PY Python module is found at https://github.com/sorna-matthew/m2-python

Using either an SVN client (described in the section, *Versioning*), or just by simply downloading a ZIP file of the working version of M2PY, put the resulting folder somewhere in your documents folder, or any other location you would be storing any of your work that will be utilizing this system (**just so you don't lose it!**)

Navigate to *m2-python>trunk>m2py* and find the file *m2py.py*. Copy this file.

Next, navigate to the following location. (It may be slightly different depending on if you chose to alter the default setting for installation directory)

C:\Users\<username>\Anaconda3\Lib

In this directory, paste a copy of *m2py.py*. This will then allow Python to properly import the m2py module.

You will need to **manually update this file** when changes are pushed to Github. Keep an eye out for when this file is updated to ensure you're utilizing all new features. I'll do my best to send out an email or something to remind everyone that something major has been changed.

COMMUNICATING WITH THE M2 OVER A SERIAL PROMPT

With the printer turned on, and the COM port setup, we can now communicate with the printer, sending simple GCode commands one at a time. To do this, we will utilize the M2PY function, `prompt()` (assuming M2PY has been installed properly as described earlier).

First, open Spyder.

In the bottom left corner of the interface is the IPython console, shown below:

```
In [1]:
```

To import the M2PY module into your current kernel of IPython, type:

```
import m2py as mp
```

and hit Enter.

If nothing happens, that means you installed M2PY correctly! Otherwise, make sure your working version of m2py.py is located in the Libs folder as previously described.

[For those unfamiliar with Python, using the syntax *import * as ** allows you to import a module and shorten its name. For example, m2py can be shortened to just mp. That way when you call functions in the IPython console, or in your script, you can preface your commands with just (mp.) For example:

```
mp.move(x = 10, y = 20, z = -100)]
```

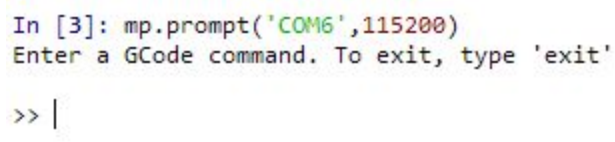
With the M2PY library properly imported, open the mp.prompt using the following command:

```
mp.prompt('COM#',115200)
```

Where # is the number corresponding to the COM port you identified earlier. For example:

```
mp.prompt('COM6',115200)
```

Which will look like the following:



You can now type in GCode commands line by line after the >>, hitting `Enter` when you want to send them.

After each command you send, the absolute coordinates of the tool head are displayed.

However, it is important to note that the M2 has no sense of its global coordinates, and therefore, this prompt will take the current position of the tool head when the prompt is initialized as (0,0,0).

Any subsequent movement commands `(G1 X10 Y20 Z-30)` will be relative from this point.

Instead, it would be more beneficial to send the following commands to get your printer into a well-known state before sending any manual movement commands.

```
G91
G28
```

These commands would set the coordinate system of your printer to relative, and home your X Y and Z axes. Every G28 command will also set the X Y & Z coordinates of your tracking point back to zero.

```
In [5]: mp.prompt('COM6',115200)
Enter a GCode command. To exit, type 'exit'

>> G91
Currently at (0, 0, 0)

>> G28
Currently at (0, 0, 0)

>> G1 X10 Y20 Z-30
Currently at (10.0, 20.0, -30.0)

>> G28
Currently at (0, 0, 0)

>> G1 X20
Currently at (20.0, 0, 0)

>> exit
Serial port disconnected
```

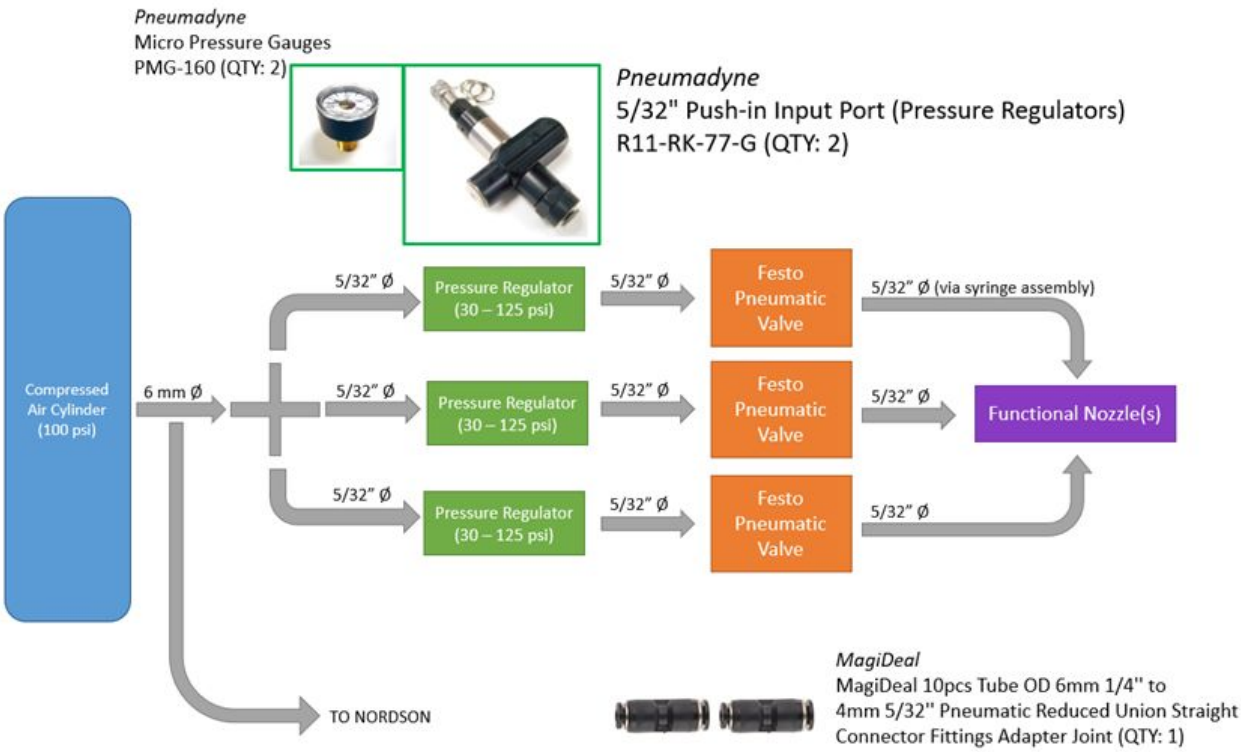
To exit out of this mode, and to release the COM port of the printer, simply type `exit` in the command prompt.

ADDITIONAL SYSTEM INFORMATION

The following section outlines some additional information that may be more relevant for those who are interested in learning about the specifics of how the Makergear was modified to accept new GCode commands to allow for the control of pneumatic valves. This section will also be useful for those who think they'd like to make additional changes to the Makergear firmware.

UNDERSTANDING THE PNEUMATIC CONTROL SYSTEM

Besides the Makergear M2 itself, there is an entire system of pneumatic tubing, regulators and switches which allow for the rapid control of material dispensing, outlined in the schematic below. This section will give an overview of the various components involved and how they interact to provide quick actuation of these three independent pneumatic channels



HOW TO MANUALLY CONTROL THE PNEUMATIC VALVES

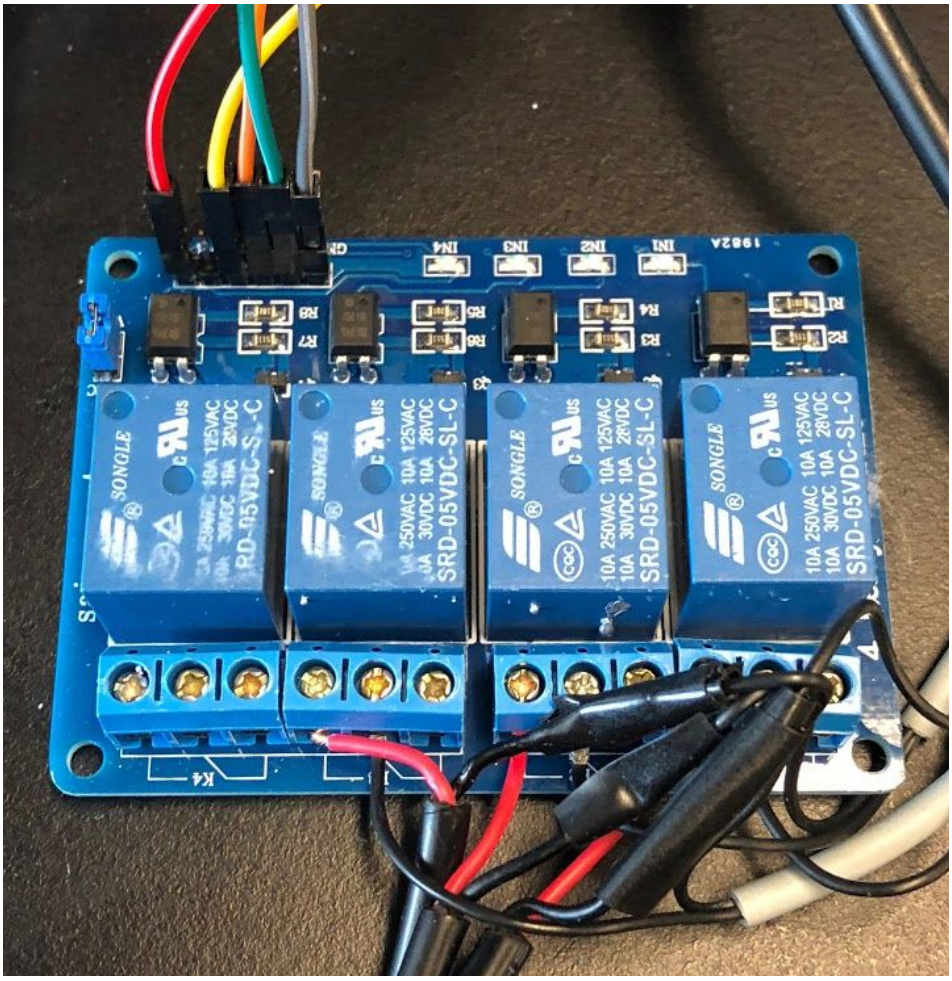
The current iteration of the Pneumatic Control System allows for the individual control of three pneumatic switches via GCode commands sent to the Makergear M2.

Six custom GCode commands were added to the Marlin firmware of the Makergear M2 (http://marlinfw.org/).

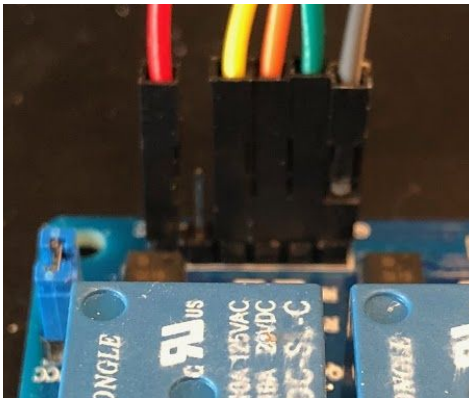
```
M3 - Channel 1 ON
M4 - Channel 1 OFF
M5 - Channel 2 ON
M6 - Channel 2 OFF
M7 - Channel 3 ON
M8 - Channel 3 OFF
```

[A version of this customized Marlin / Makergear M2 firmware exists in the Github directory as well. This firmware most likely won't need to be re-uploaded unless control system capabilities change.]

When these commands are interpreted by the code parser on the M2, specific **Digital I/O** aboard the electronics board (**RAMBo Ver. 1.3L** http://reprap.org/wiki/Rambo_v1.3) are set either high (ON) or low (OFF). These pins are connected through jumpers to a mechanical relay which sits in front of the M2's electronics housing.



The pins on the mechanical relay are in the upper left-hand of the image above. Shown more clearly below, these pins are described, and their connection to the electronics board of the M2 is explained in greater detail.



Jumper Color	Function	Pin on Relay	Pin on RAMBo*	Digital I/O #
Red	VCC (+5V Power)	VCC	PWM_EXT 1	--
Yellow	Channel 3	CH3	PWM_EXT 4	2
Orange	Channel 2	CH2	PWM_EXT 5	5
Green	Channel 1	CH1	SERIAL_EXT TX3	14
Gray	GND (Ground)	GND	PWM_EXT 2	--

*(http://reprap.org/wiki/Rambo_development) & (http://reprap.org/wiki/Rambo_v1.3#Schematic)

These Digital I/O numbers correspond to the value of the pins being set high and low via software on the Arduino IDE.

Practically speaking, you can control these channels manually by sending the GCode commands defined above, or by using the Python wrapper module **M2PY**, which will be discussed in greater detail.

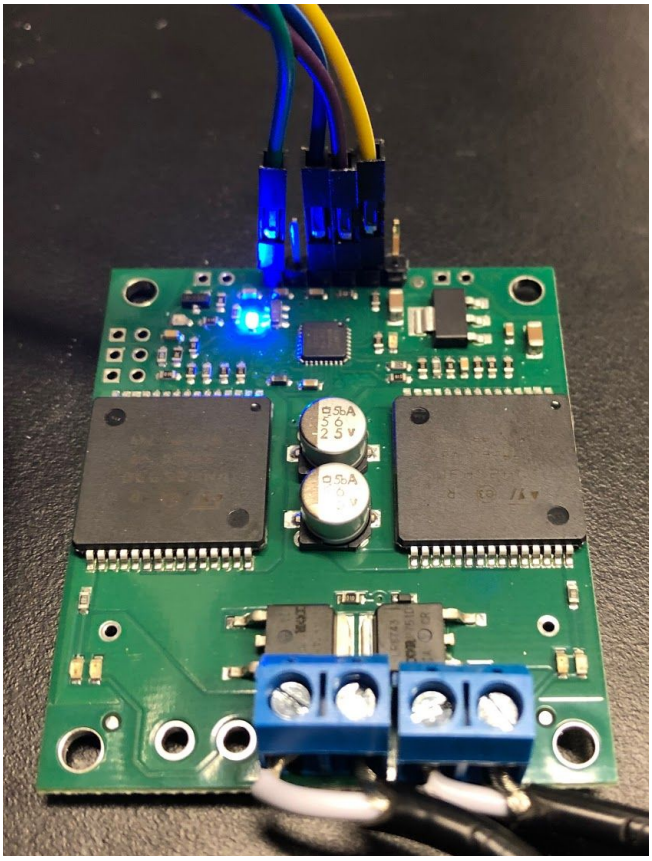
HOW TO MANUALLY CONTROL THE DC MOTOR

New to version 1.1 is the inclusion of the control of a DC motor. While the **rotate** and **ramp** functions will be explained in greater detail in the section explaining the **M2PY** library, this section will explain how the motor control board is wired up, and how the motor is controlled using a custom GCode command.

As with the channels, this motor is controlled using a custom GCode, M9, with the following format:

```
M9 S100
```

Where the integer after “S” is the setting of the motor speed, from -127 - 127, with 127 being the maximum rated speed of the attached motor in the ccw direction, and -127 in the max cw direction. As of Version 1.1, this motor has a maximum rotation speed with zero added torque of roughly 500 RPM. The conversion between S value and true rotation speed in RPM would need to be measured and calibrated.



Like the mechanical relay for the channels, the motor is controlled from an external control board which takes commands from the printer controller and in turn actuates the motor. The pinout and connection diagram is shown below:

Jumper Color	**Function**	**Pin on Controller**	**Pin on RAMBo***	**Digital I/O #**
Green	Ground	GND	SERIAL_EXT GND	--
Blue	Receive	RX	SERIAL_EXT RX1	19
Purple	Transmit	TX	SERIAL_EXT TX1	18
Yellow	Reset	RST	SERIAL_EXT RX3	15

The motor has a three-prong power cord connected to the power brick with the label “Motor Power” beside the M2. When this is plugged in, the blue LED on the control board will turn on. This LED will also turn on when the M2 is powered on. If both the M2 and the Motor Power brick are plugged in, the blue LED will light up as well as a green flashing heartbeat. Additionally, when motor M0 (the board also has the possibility of driving two motors, M0 and M1, we only have M0 attached and coded internally with M9) is rotating with speed S>0, an additional green indicator LED will light. For S<0, a red indicator LED will light.

POSSIBILITIES FOR FUTURE CONTROL SYSTEM EXPANSION

If there were ever a need to expand the functionality of this printer, or one of a similar design to allow for additional synchronous control signals, there is plenty of unused pins to take advantage of, namely any of the TX/RX 2 Pins in the serial expansion header on the RAMBo 1.3L. TX/RX 1&3 are in use for the channels and motor control, but there is still free space in the serial expansion header.

Some unused pins for future expansion

Pin on RAMBo*	**Digital I/O #**
SERIAL_EXT TX2	16
SERIAL_EXT RX2	17

If you’re interested in expanding the functionality of this system, and would like to alter the firmware of the Makergear, follow these steps.

Download and install the software, Arduino IDE
(**BETA 1.5.5**, which is the latest stable version that works with the M2 Electronics Board)

(https://www.arduino.cc/en/Main/OldSoftwareReleases#1.5.x)

Once that’s installed, in the customized Marlin firmware found on the Github, open the file `m2pcs_marlin_firmware.ino`

Navigate to the file: `Marlin_main.cpp`

Do a **CTRL-F (CMD-F)** search for *M2PCS*, which should bring you to the modified sections of the code written specifically for this system.

```
// [M2PCS] CHANNEL/FUNCTION PIN DEFINITIONS
int CH1 = 14;
int CH2 = 5;
int CH3 = 2;

// [M2PCS] PIN DIRECTION SETTING AND INTITIAL STATE CONFIGURATION
pinMode(CH1, OUTPUT);
pinMode(CH2, OUTPUT);
pinMode(CH3, OUTPUT);

digitalWrite(CH1, HIGH); // RELAY INITIALLY IN OFF POSITION
digitalWrite(CH2, HIGH); // RELAY INITIALLY IN OFF POSITION
digitalWrite(CH3, HIGH); // RELAY INITIALLY IN OFF POSITION

//[M2PCS] Motor Control
#include <SoftwareSerial.h>
#include "PololuQik.h"
PololuQik2s12v10 qik(19, 18, 15);

//[M2PCS] Motor Control
qik.init();
```

It would then be possible to write self similar case/switch statements and add them to the firmware in order to add additional synchronous command functions in GCode, which are shown on the following page. You just have to make sure that no custom M code you write is already defined by the printer. As of Version 1.1, M10-M19 are still available for custom use.

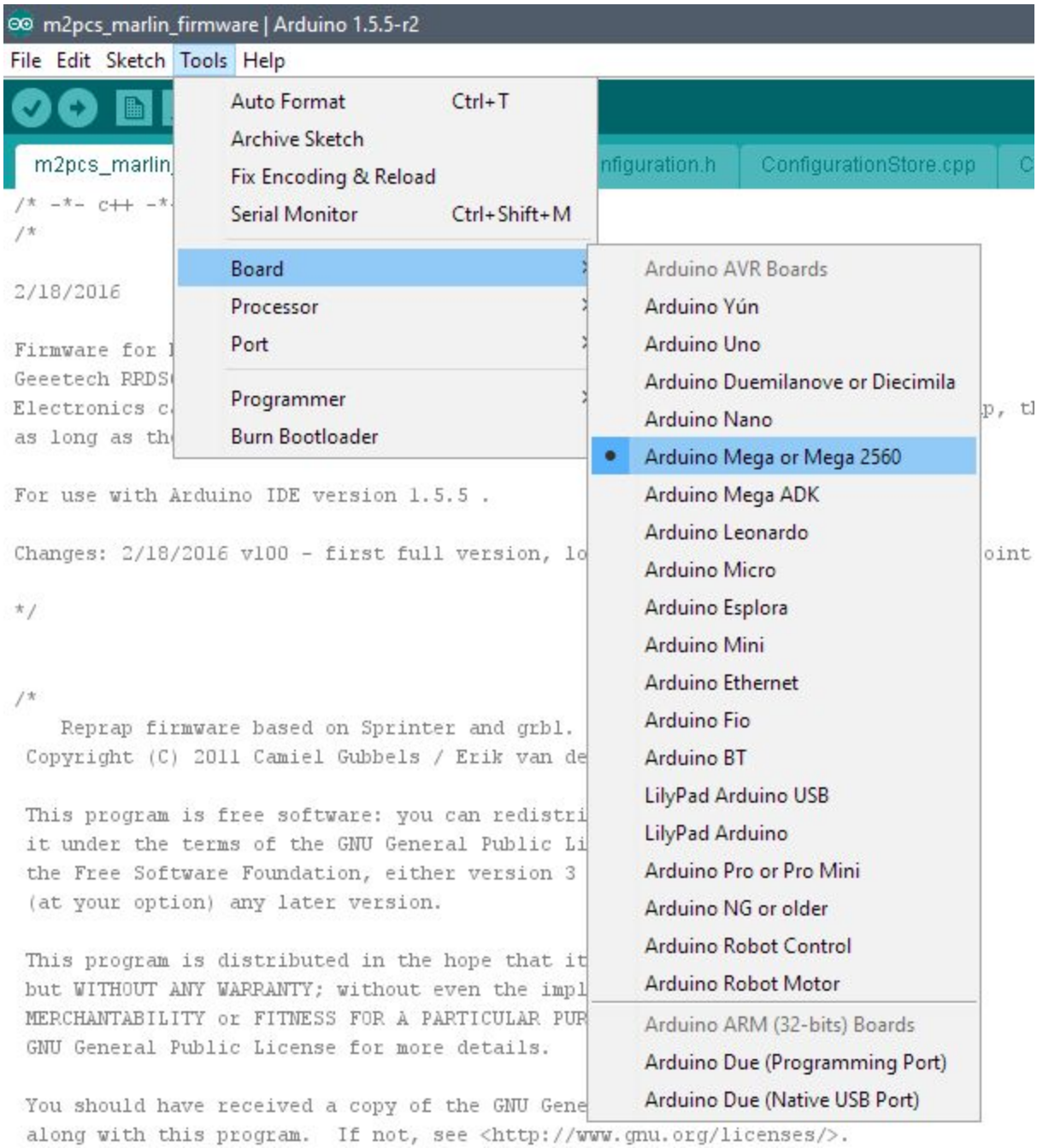
```
  // [M2PCS] CUSTOM GCODE DEFINTIONS
  //(check to make sure that any additional M
  // st_synchronize() requires printer to be
  // Without it, printer will act on command

case 3: // M3 CH1 ON
    st_synchronize();
    digitalWrite(CH1, LOW);
    delay(CH_ON_DELAY_TIME);
  break;

case 4: // M4 CH1 OFF
    st_synchronize();
    digitalWrite(CH1, HIGH);
  break;

case 5: // M5 CH2 ON
    st_synchronize();
    digitalWrite(CH2, LOW);
    delay(CH_ON_DELAY_TIME);
  break;

case 6: // M6 CH2 OFF
    st_synchronize();
    digitalWrite(CH2, HIGH);
  break;

case 7: // M7 CH3 ON
    st_synchronize();
    digitalWrite(CH3, LOW);
    delay(CH_ON_DELAY_TIME);
  break;

case 8: // M8 CH3 OFF
    st_synchronize();
    digitalWrite(CH3, HIGH);
  break;

case 9: // M9 MOTOR SET SPEED
    st_synchronize();
    if (code_seen('S')) {
      MOTOR_SPEED = code_value();
      qik.setM0Speed(MOTOR_SPEED);
    }
  break;
```

UPLOADING FIRMWARE CHANGES TO THE M2

Assuming you made any necessary changes you needed to the firmware, you can follow these steps to upload the firmware to the Makergear, using the Arduino IDE.

With the firmware properly saved, make sure you have the proper target **board** type selected:

Navigate to Tools > Board > and select **Arduino Mega or Mega 2560**



Similarly, go to Tools > Processor and select **ATmega2560 (Mega 2560)**

And lastly, check to make sure the target COM port is selected:

Tools > Port > and select the COM port assigned to the M2 via Device Manager

Again, assuming you have the proper Arduino IDE version installed (**Arduino 1.5.5-r2**) you should have no issues in compiling and sending your firmware to the Arduino onboard the M2 electronics control board.

In the upper left-hand portion of the screen, click the RIGHT ARROW button, and upload the firmware to the M2. Below in the console you'll be told if the firmware uploaded properly. If not, it's typically due to the fact that you messed something up in the syntax. Remember that this code is written in C (occasionally C++), so make sure all lines of code end with a ";" (common mistake)

You should also see the yellow lights from under the electronics housing of the M2 flash during the writing process of the new firmware and will stop flashing rapidly when done. Please, DO NOT POWER OFF during this time, as the firmware of the board could become unstable, and unable to receive any additional uploads, effectively frying the Arduino.

PRINTING WITH THE M2PCS

HOW TO PRESSURIZE THE PNEUMATIC CHANNELS

Pressurized air from the supply cylinders makes its way to the valve beside the ShopBot. From there, the line tees off to both the Nordson pressure regulators, and to the three inline regulators pressurizing the pneumatic valves attached to the M2.

Assuming you are ready to go, and have syringes prepared:

1. Attach the appropriate syringe assembly (3/10/30cc) to PORT 2 on the Festo pneumatic valves. All fitting in this system are push-in and should be fairly simple to swap fittings in and out.
2. Mount syringe(s) to the mounting plate / back plate on the M2.
3. Attach syringe assembly to the syringes.
    a. Make sure that each syringe is hooked up to the proper valve (CH1, CH2, CH3)
4. With these fittings attached, you can open up the air cylinder, and then the valve beside the ShopBot. At this point, lines should be all pressurize up to the valves, but no air should be escaping. If so, check push in fittings for leaks.
5. Now is this time to adjust the in-line pressure regulators
    a. Each regulator (P1 P2 P3) corresponds to a pneumatic channel.
    b. Dials on the gauges are controlled by the knurled nuts on each regulator, screwing in raises pressure, unscrewing lowers pressure
        i. Adjust the valves slowly, as the gauges take some time to accurately read the new pressure, especially when unscrewing to lower pressures. Let the gauges sit for ~ 30 seconds after adjusting to make sure the set value you wanted is reflected on the gauge
        ii. It's also important to note that if you vary the max output from the valve on the air cylinder, **that will change the relative pressure output of each channel.** So don't assume that pressures are the same every time you start up the system, and adjust accordingly.

The last, **and most important step**, is to power the switches controlling the channels!

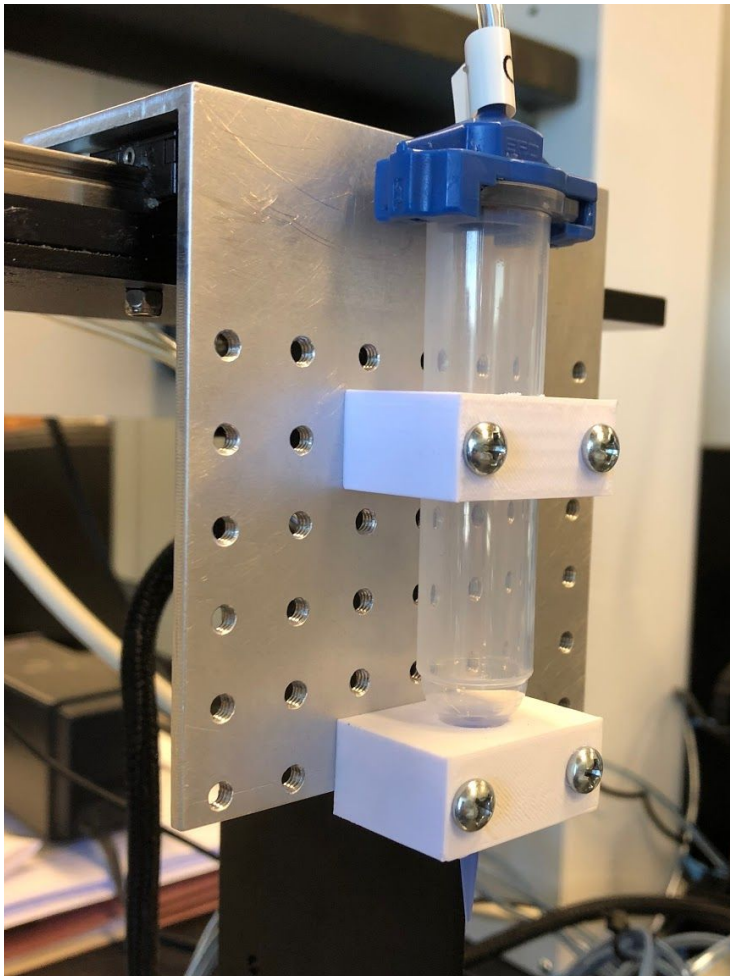
To the left of the Makergear is a power brick, **"Channel Power"**

These pneumatic valves are **normally closed** and will only open when powered. All that you need to do to provide these power is plug the power brick into an outlet. The green led on-board the power brick should light up when powered.

Make sure to unplug the power brick when finished with printing.

DESIGNING STATIONARY NOZZLE MOUNTS USING THE M2 FACEPLATE

Although a simple set of acrylic laser cut mounts exist from back when the system was first built, it was made clear that a mounting faceplate similar to the one found on the ShopBot needed to be developed to allow for modular nozzle mount designs.



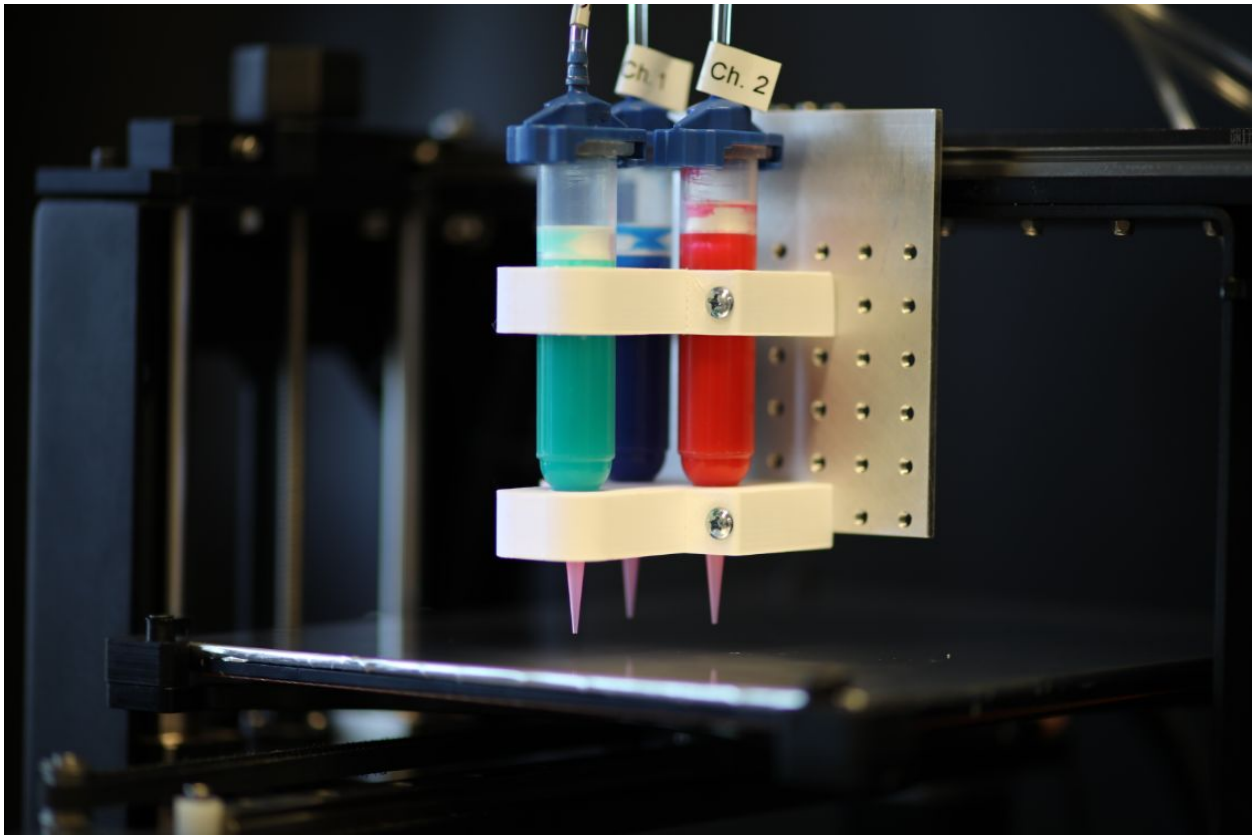
The faceplate is shown above.
Its main face is 3.5x4”, with 42 8-32 threaded screw holes, at 0.5” spacing (6 x 7)

When designing mounts compatible with regular 8-32 screws, I’ve found it best to make the holes **4 mm** in diameter. It ends up being a little tight, but you can still fasten the screws manually using a screwdriver, guaranteeing a nice snug fit, making the holes pseudo-threaded.

Also, for a 10 cc syringe that we typically use, the lower mount has a diameter of **11 mm**, and the upper mount has a diameter of **19 mm**. These too were found using particular print settings, so they may need to be adjusted depending on the application.

Three different syringe mounts have been designed and printed, and should be located near the M2. STEP files of each (including upper and lower syringe mounts) can be found in the Team Drive under **Makergear M2 Pneumatic Control System [M2PCS] > System CAD Files >> Stationary Syringe Mounts**

Feel free to take a look at these and create new mounts with improved or modified geometries as you see fit.

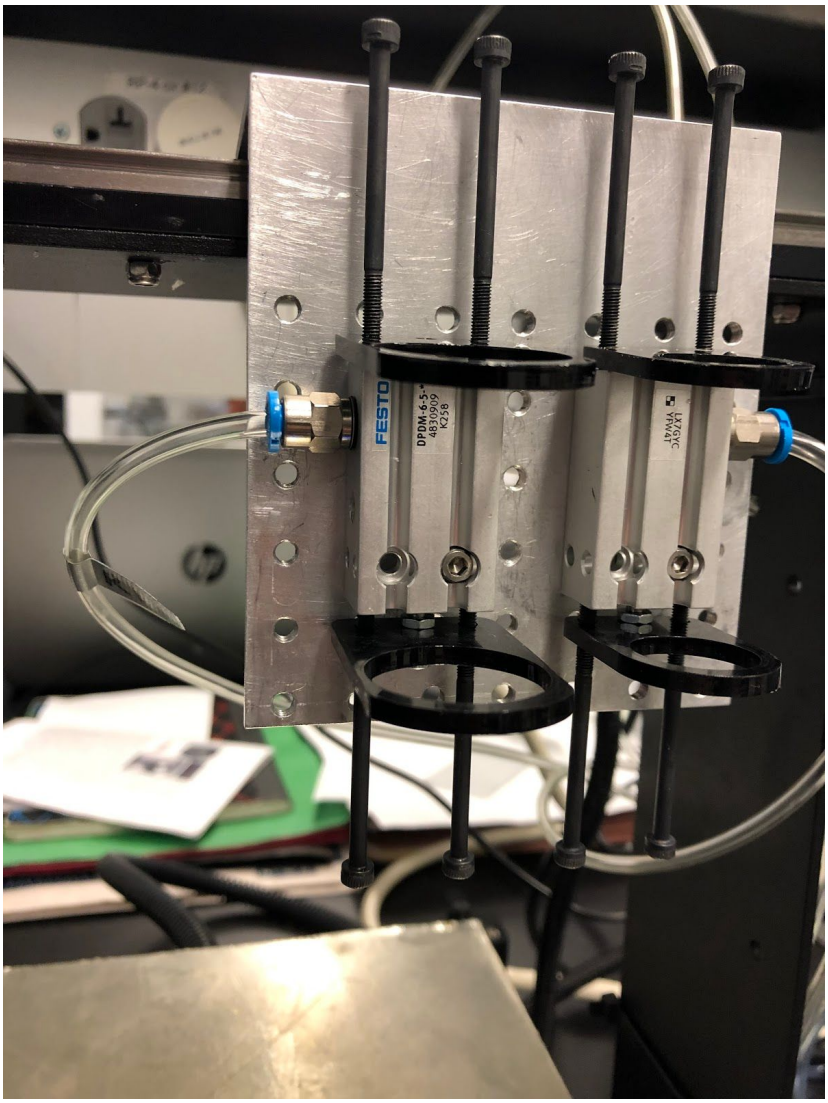


PRINTING USING RETRACTING NOZZLES

One of the major advancements in Version 1.2 was the addition of pneumatic linear actuators to the faceplate of the Makergear, allowing the retracting of unused nozzles away from the build plate when not dispensing.

To achieve this, pressurized air is tee'd off of each of the three pneumatic channels prior to being regulated and actuates three simple on/off pneumatic linear actuators. The other half of the line is regulated down to a set pressure using the manual valves and used to dispense material through the syringe barrels.

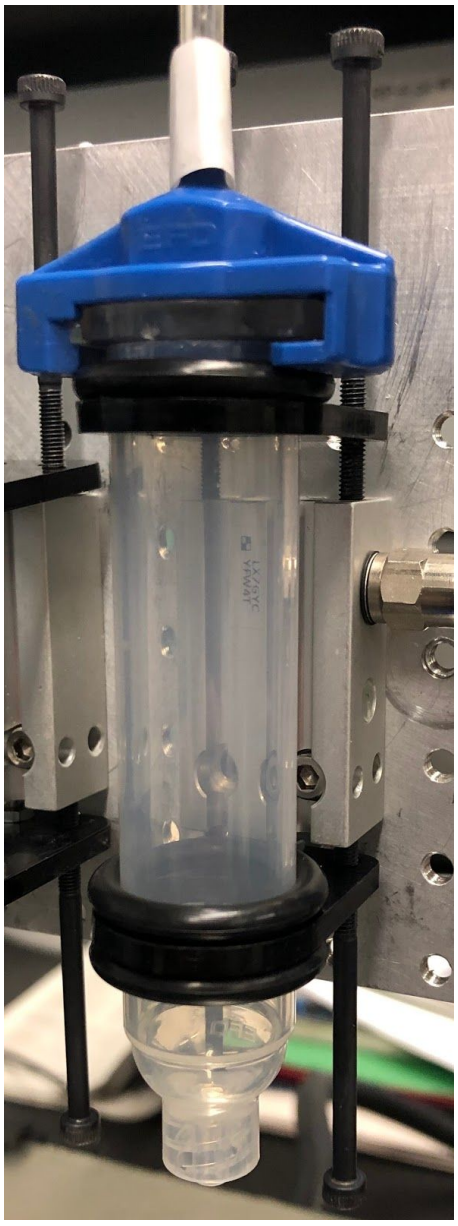
A picture of the current iteration of the retracting syringe mount is shown below:



Two laser cut acrylic pieces are free to slide along “rails” which are in this case M3x50 screws that fit into the threaded holes found on the top and bottom of the linear actuators. Additionally, the bottom acrylic piece is fastened to the actuating rod of the pneumatic actuator. The bottom piece drives the actuation, while the top piece is there for stabilization and to keep the syringe perpendicular to the build plate.

LOADING SYRINGES INTO RETRACTING MOUNT

To keep the syringes from slipping relative to the two acrylic pieces, several o-rings are put on the syringe body. One at the top of the syringe barrel, and two sandwiching the lower acrylic mount, as shown below. You sort of have to thread the syringe through the mounts while you slip the o-rings on the syringe body.



For a 10cc syringe, you’ll know the syringe is properly loaded into the mounts when the top o-ring is snug against the top of the upper acrylic piece.

You should also be able to use two fingers to pull down on the lower mount and actuate the linear actuator by hand to see that the syringe can travel the full length of its throw distance *without the top acrylic piece hitting into the top of the actuator.* If the syringe can’t seem to travel the full distance, it usually means that you have to slide the lower mount down further to grip the syringe barrel at a lower point.

For a 30cc syringe, the same o-rings can fit around the syringe barrel, it’ll just be a bit more of a stretch. The same procedure for the 10cc syringe can be followed.

There’s a bit of extra work that needs to be done when using two syringes of different sizes, like a 10cc and a 30cc syringe at the same time.

Load the 10cc syringe as normal. To match the level of the two nozzle tips in their retracted state, the 30cc syringe needs to be shifted up in the mounts so the nozzle tips line up with the 10cc syringe. The o-rings will keep the syringe stationary in whatever position you slide it into.

USING THE M2PY PYTHON MODULE

As shown previously, it is possible to communicate with the M2 over serial command prompts, sending lines of GCode manually for simple adjustments.

However, when you would rather control the printer programmatically, taking advantage of the logic structures and features available from Python, while controlling the M2 through a more user-friendly interface, you can use the Python module **M2-Python (M2PY)** [https://github.com/sorna-matthew/m2-python]

While the most up to date information regarding functions, their arguments and examples of implementation are found at the Github link above, a copy of stable function definitions are provided below:

M2PY *(a Python module)*

m2py.Makergear(*com*, *baud*, *printout* = *0*, *verbose* = *True*): If printout = 1, this function will instantiate a serial object used by all subsequent function calls to send serial commands to the specified printer. If printout = 0, this function will store all relevant coordinate changes (move and arc commands) to a temporary file that can then be used to visualize print paths before sending commands to the printer. By default, printout = 0. The flag verbose controls the print statements to the console. With verbose = True, all print statements are printed. With verbose = False, all print statements are suppressed.

```
import m2py as mp
mk = mp.Makergear('COM3',115200)
```

```
import m2py as mp
mk = mp.Makergear('COM3',115200, printout = 1, verbose = False)
```

close(): closes the specified Makergear object. If printout = 1, this function will close the necessary serial object. If printout = 0, this function will close the specified temporary file and plot a visualization of all relevant movement commands. Visualization function will

use whatever coordinate system you explicitly designate using coord. If coord isn't explicitly called, the coordinate system used by the visualization tool will be *absolute*.

```
import m2py as mp
mk = mp.Makergear('COM3', 115200)
mk.close()
```

GCODE WRAPPERS

G0 / G1

move(*x = 0*, *y = 0*, *z = 0*): moves to the specified point, keeping in mind the coordinate system (relative / absolute)

```
import m2py as mp
mk = mp.Makergear('COM3',115200)
mk.move(x = 10, y = -5) # x, y, z arguments are all keyword arguments, and default to 0 when not called
mk.close()
```

speed(*speed = 0*): sets the movement speed of the printer to the specified speed in [mm/s] (default `0` mm/sec)

```
mk.speed(speed = 40) # sets the movement speed of the printer to 40 mm/s
```

rotate(*speed = 0*): sets the rotation speed of the motor to the specified speed (default `0`)

```
mk.rotate(speed = 30) # sets the rotation speed of the motor to 30
```

ramp(start = 0, stop = 0, *seconds = 1*): sets the rotation speed of the motor from the start speed to the specified stop speed over a given time in seconds [0-127]

```
mk.ramp(start = 0, stop = 30, seconds = 1) # ramps the rotation speed from 0 to 30 in 1 second
```

G2 / G3

arc(*x = 0, y = 0, i = 0, j = 0, direction = 'ccw'*): moves to the specified x-y point, with the i-j point as the center of the arc, with direction specified as `'cw'` or `'ccw'` (default `'ccw'`)

```
mk.arc(x = 10, y = -5, i = 2, j = 3, direction = 'ccw')
```

G4

wait(*seconds = 0*): waits for the specified amount of time (default `0` seconds)

```
mk.wait(seconds = 5) # causes the printer to wait for 5 seconds
```

G28

home(*axes = 'X Y Z'*): homes the specified axes (default `'X Y Z'`)

```
mk.home(axes = 'X Y Z') # homes all three axes
mk.home() # homes all three axes
mk.home(axes = 'X Z') # homes only the specified axes
```

G90 / G91

coord_sys(*coord_sys = 'abs'*): sets the coordinate system of the printer [relative or absolute] (default `'abs'`)

```
mk.coord_sys(coord_sys = 'abs') # sets coordinate system to absolute
mk.coord_sys(coord_sys = 'rel') # sets coordinate system to relative
mk.mclose()
```

G92

set_current_coords(*x = 0, y = 0, z = 0*): sets the current position to the specified (x, y, z) point (keeping in mind the current coordinate system)

```
mk.move(x = 10)
mk.set_current_coords(x = 0) # sets this new position to x = 0
mk.mclose()
```

return_current_coords( ): returns the current stored coordinates of the Makergear object

```
mk.move(x = 10)
coords = mk.return_current_coords( ) # returns current coords
print(coords)
mk.mclose()
```

set_tool_coords(*tool = 1*, *x = 0*, *y = 0*, *z = 0*): sets internally stored coordinates of each tool, used in switching commands, relative to tool 1 which is defined at [0,0,0]

```
mk.on(1)
mk.move(x = 10)
mk.set_tool_coords(tool = 1, x = 0, y = 0, z = 0)
mk.set_tool_coords(tool = 2, x = 10, y = 10, z = 0)
mk.on(2)
mk.move(x = 10)
```

change_tool(*change_to = 1*): This subroutine automatically switches from the current to specified tool

```
mk.on(1)
mk.move(x = 10)
mk.change_tool(change_to = 1)
mk.on(2)
mk.move(x = 10)
```

M2PCS SPECIFIC FUNCTIONS

M3 - M8

on(*channel*): Turns pneumatic channel ON

off(*channel*): Turns pneumatic channel OFF

```
mk.on(1)
mk.move(x = 10)
mk.off(1)
```

set_channel_delay(*delay=50*): sets the delay time (in ms) between a channel turning on and the execution of another command. Can be used to fine tune under extrusion effects, depending on ink viscosity.

```
mk.set_channel_delay(delay = 50)
mk.on(3)
mk.move(x = 10)
mk.move(x = -10)
mk.off(3)
```

SIMULTANEOUS FUNCTIONS

allon(): Turns all three pneumatic channels ON

alloff(): Turns all three pneumatic channels OFF

```
mk.allon()
mk.move(x = 10)
mk.alloff()
```

ADDITIONAL FUNCTIONS NOT IN THE M2PCS CLASS

mp.prompt(*com*, *baud*): allows for quick, native GCode serial communication with the M2, provided that the proper com port and baud rate are selected, and match what is found in system settings. To exit the command prompt environment, just type `exit` in the IPython console.

```
mp.prompt('COM3',115200)
```

mp.file_read(*fid*, *com*, *baud*): reads in a text file of GCode line by line, and waits for the M2 to acknowledge that it received the command before sending another, maintaining print accuracy.

```
mp.file_read('C:/Users/Matthew/Documents/m2-python/trunk/print
paths/test_path.txt','COM3',115200)
```

HOW TO MANUALLY HOME YOUR NOZZLE BEFORE PRINTING

If you'd like to use the new Movement GUI to home the printer, skip ahead to the GUI section.

Since this system has been modified from its original configuration of FDM printing, some of the automatic functions that most commercial printers take advantage of now have to be done manually. One such function is how to zero your nozzle into its home positions from which you would like to begin printing.

As discussed in the section *COMMUNICATING WI*TH THE M2 OVER SERIAL PROMPT, you'll first need to initialize your printer and invoke a serial prompt in Spyder.

First things first, send the two commands, one at a time:

```
G28
G91
```

These two commands will zero the X Y and Z axes and set the coordinate system to relative, respectively.

The bed should now be all the way down, and back, with the x-gantry all the way to the left. This is the true (0,0,0) position of the printer, and will be reflected in the serial prompt.

Assuming that your nozzle(s) is properly mounted to the x-gantry, you can now begin the process of sending manual G1 movement commands in the form:

```
G1 X10
G1 Y20
G1 Z-100
```

Keep in mind that from this (0,0,0) position in the upper left hand corner of the build plate, positive X movement commands move the tool head to the right ->, and positive Y movement commands move the bed forward \/ (which basically moves the tool head "backwards"). HOWEVER, in order to raise the bed towards the nozzle you'll need to send NEGATIVE Z MOVEMENT COMMANDS.

If you'd like to increase the distance between the nozzle and bed, send POSITIVE Z MOVEMENT COMMANDS. This will be very important when designing print paths, especially when you have some dz value between layers.

You'll want to do this line by line and monitor the position of your nozzle tip relative to the print bed, as to not damage either.

Begin first with coarse adjustments to get the nozzle tip in the ballpark of the starting location of the print. It's easier to align your print in the XY plane and then work on the Z axis last.
With the tool roughly where you'd like it in the XY plane, you can begin to make the Z movement commands progressively finer as the nozzle approaches the print bed.

Don't be afraid to send movement commands as small as (G1 Z-0.1) to really fine tune your position.

When you make the necessary adjustments and feel as though the nozzle and print bed are touching, you will want to make note of the (X, Y, Z) coordinates reflected in the serial prompt.

If you know what you want your initial z displacement between the bed and nozzle to be at the beginning of your print, have the two be completely touching, and then add the +Z movement command equal to this initial displacement height, and record that coordinate instead,

You'd want to use these coordinates as the starting location of your print!
Regardless of whether or not you will be using relative or absolute coordinates, you'll need this information.

Assuming you set the printer to (0,0,0) using **mp.home()** before you begin your print path, you'd want to move to this predetermined (X, Y, Z) coordinate as either the starting location of your print path which uses relative coordinates (and will treat this is a new (0,0,0)) or the absolute coordinate from which you manipulate to generate your print path.

USING *GCODE TOOL* TO PRINT GCODE PRINT PATHS

If you would like to print a path which was generated using a conventional FDM slicer, a simple GUI has been developed to let you do so, assuming you already have a **.txt or .gcode** file to read in. This is also assuming that you made some changes to the slicer software to allow for channel on/off commands instead of traditional retraction steps. The development of the GUI keeps you from having to load up Spyder every time and manually enter file directories. Instead you can browse using a regular file explorer.

Before we can run this file without having to open up Spyder, we have to set up a few things (this should already be done for you on the Shopbot PC):

ADDING PYTHON TO PATH

Ideally, we want to be able to run these simple GUIs just by clicking on a shortcut on the desktop. To do this, we must point out to the PC where python.exe is located so it can be accessed easily by other functions, such as **cmd.exe** in our case.

First open up Anaconda Prompt, found by searching in the Start menu. Type the following command: *where python*, and hit Enter:

```
(base) C:\Users\Matthew>where python
C:\Users\Matthew\Documents\Anaconda\python.exe
```

Take note of this directory, excluding the “\python.exe”.

Now, open up This PC (Windows 10) / My Computer (Windows 7 and earlier), right-click and open Properties.

On the toolbar to the left, open up Advanced system settings:

Under the tab Advanced (which you should be open to already) open Environment Variables.

Under the System Variables section, highlight the variable, Path, and select Edit.

In this window, click New. This should open up an editable text field where you can type in the file location for python.exe found earlier. Remember, the file location should only be that of the containing folder of python.exe, not the file itself (remove the last “\python.exe” from the string you got from the command *where python*).

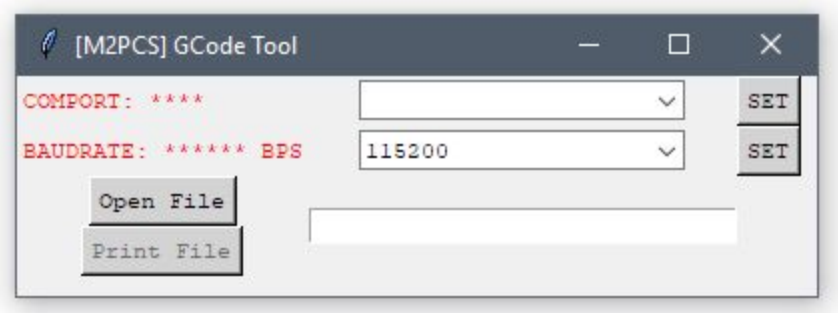
Hit OK to save and exit all these windows.

To test to make sure the proper string has been entered, open up a simple command prompt (**cmd.exe)** and type python. If a Python command prompt (>>>) is opened, you set everything up just fine. If not, double check that you got the right file location for python.exe entered under the Path variable.

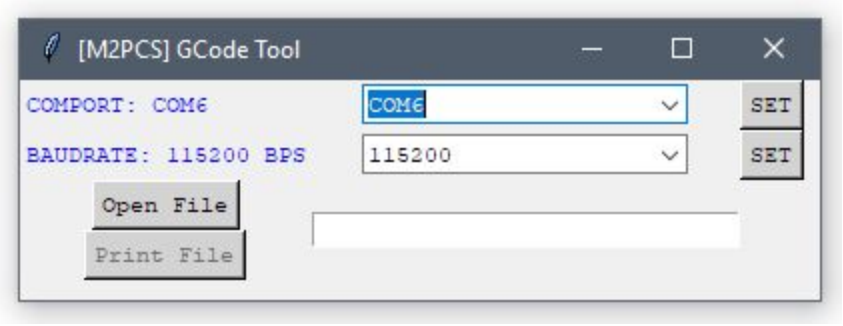
Assuming you have checked out the Github project discussed previously, you can browse to the **gui** folder and find the following file, gcode_tool.bat. Right click on this and create a shortcut. Drag this to your desktop or wherever you’d like this tool to be accessible from. You should then be able to click on this and open the GCode Tool GUI!

USING THE GCODE TOOL GUI

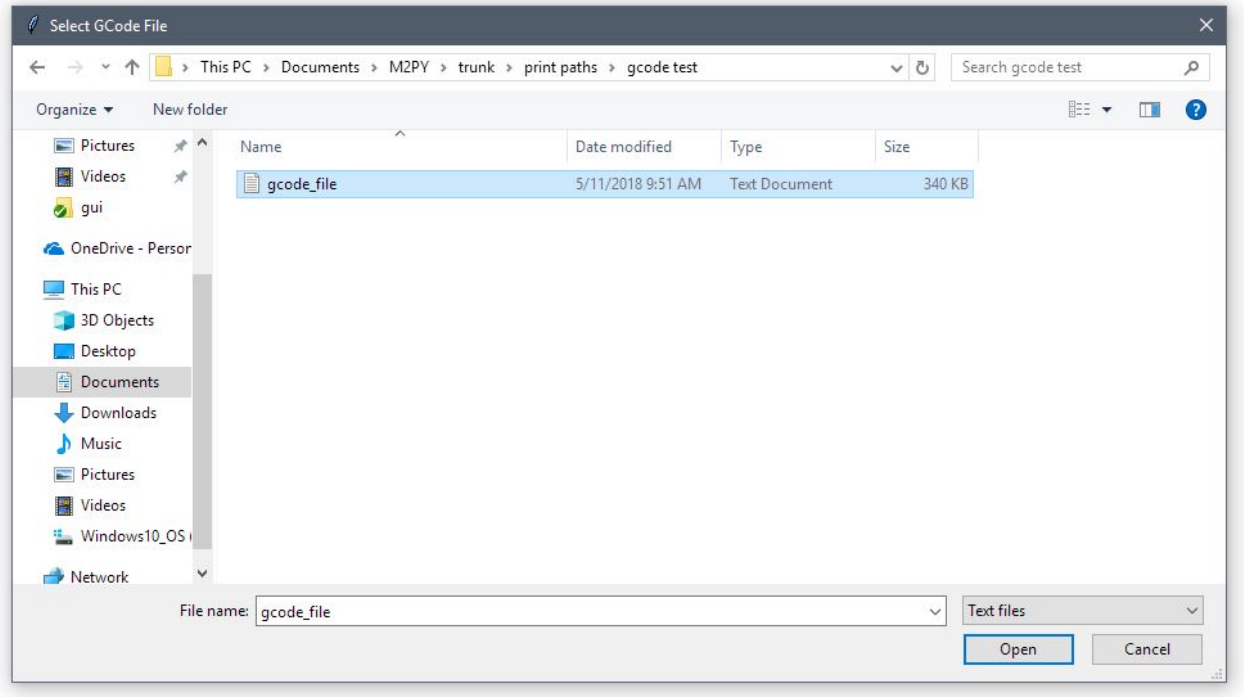
When you open the shortcut made previously, the following window should open, assuming everything was set up correctly:



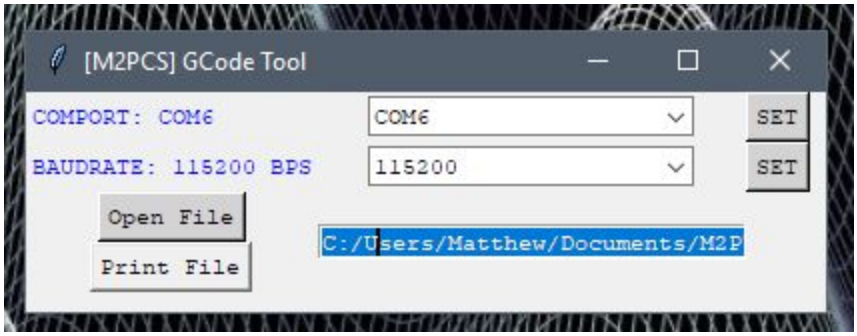
From the dropdown menus you should be able to set the COMPORT and BAUDRATE of the attached printer. Make sure to hit SET afterwards and you should see the red text turn blue when everything is set up.



You're now ready to browse to the selected GCode/text file you'd like to print with the selected printer by clicking **Open File**.



Hit **Open**, and the file location of this file should appear in the text box beside **Print File** which will also no longer be greyed out, and should be clickable. Clicking this file will start reading in the GCode line by line to the printer.

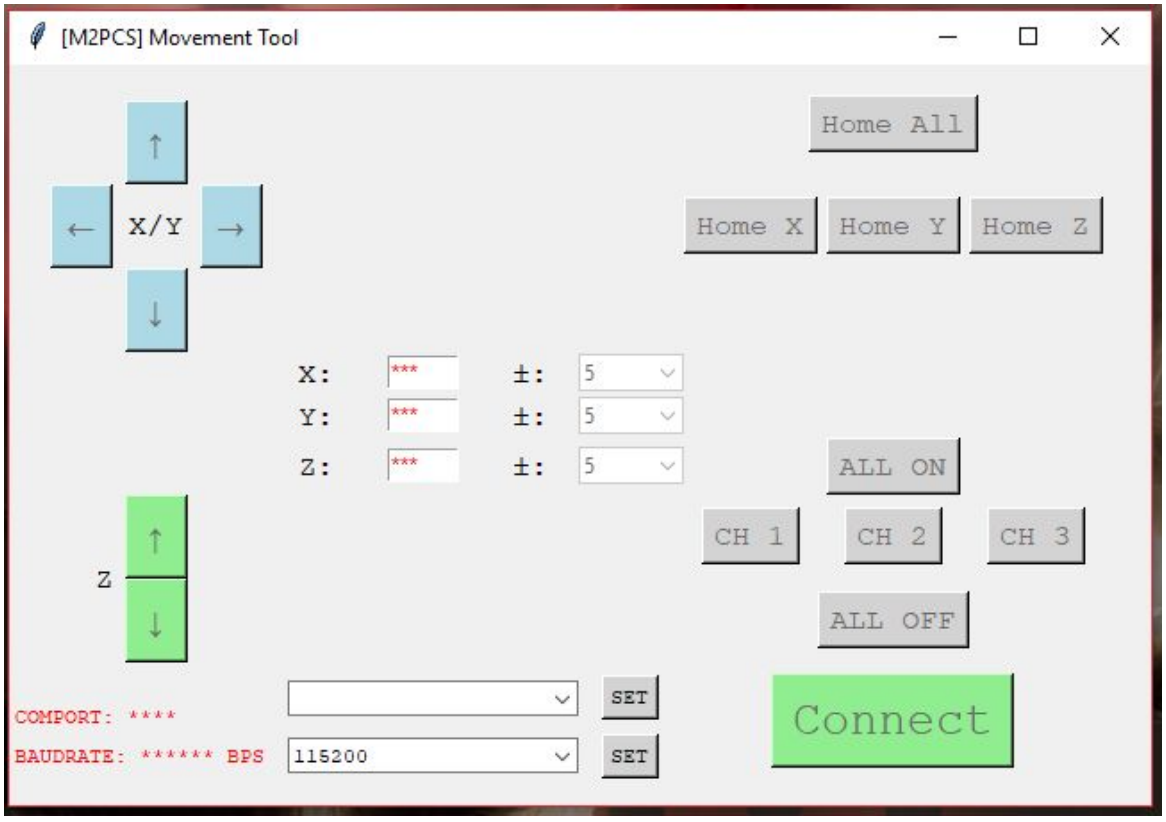


USING *MOVEMENT TOOL* TO CONTROL PRINTER POSITION

Another simple GUI has been developed to allow for simple control of printer coordinate position by manual x, y, z movement (also axis homing).

From the steps in the previous section to add Python to the Path variable, we should be able to make a similar shortcut from the **movement_tool.bat** found in the **gui** folder in the M2PY Github project.

Clicking this shortcut will open the following GUI:



Connect to the printer by specifying COMPORT and BAUDRATE (make sure to hit SET) and then hit the **CONNECT** button. This will make all the homing and movement buttons active and clickable.

You're going to want to hit HOME ALL to get your printer and its coordinates into a known state. All red *** should now be set to 0.0.

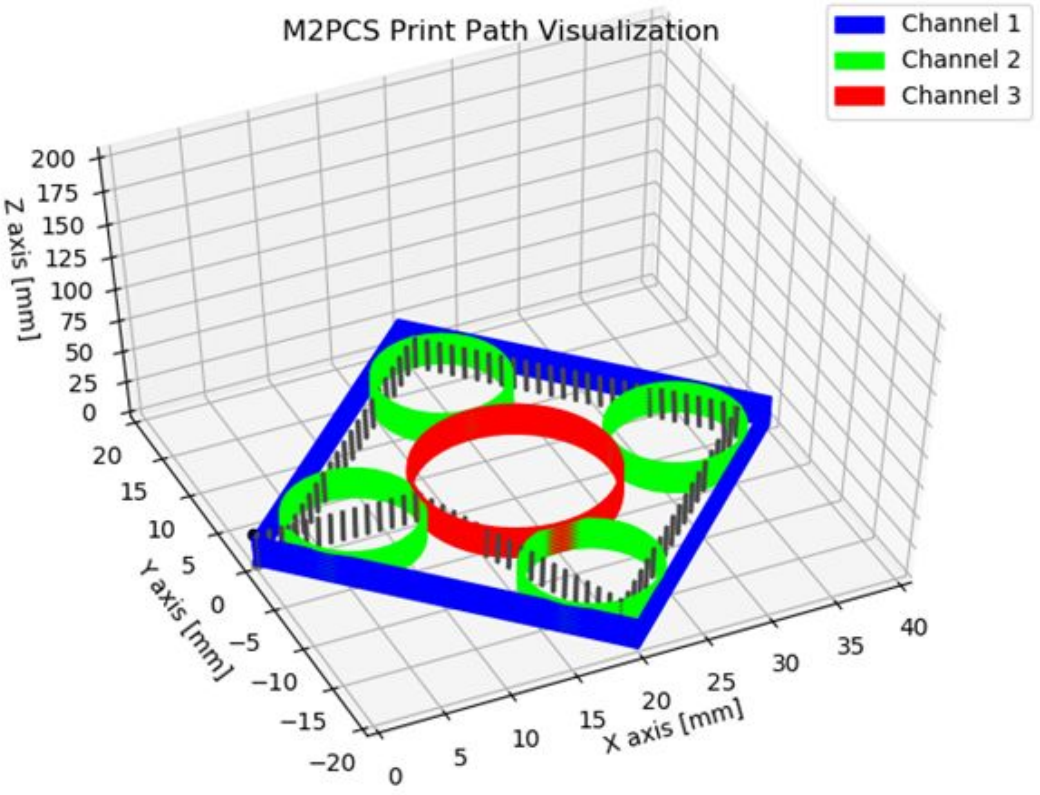
You can now control the movement increment in every axis separately. You could move X and Y with coarse 20 mm increments but may want fine (0.1 mm) control of the Z axis as your nozzle(s) approach the print bed. These can all be adjusted using the dropdown boxes beside the +/- symbols. The coordinates of the tool head are also reflected in the X Y Z text boxes, keeping in mind that they're with respect to the (0,0,0) position after the homing command.

You could choose to use this tool to help you home, rather than having to send all the homing and movement commands manually using a GCode prompt.

PRINT VISUALIZATION

There now exists the ability to visualize all movement commands done using the M2PY library, to plot print paths before actually sending movement commands to the M2.

By default, calling mk = **mp.Makergear()** will return a Makergear object typically named mk, used as a necessary argument by all subsequent command functions. If you explicitly call this argument and set **printout = 0,** rather than generating a Makergear object from **mp.Makergear()**, this function now returns a file handle which is uses to track all movement commands from **mk.move()** and **mk.arc()**, with coordinates tracked in a text file. When **mk.close()** is finally called at the end of the script, all these movement commands are automatically sent to the visualizing function, and the print path is plotted. An example plot is shown below:



Each section of the print path is printed, with colors corresponding to which channel is on during movement. If no channels are on, a dashed grey line is plotted instead. A light grey marker designates the beginning coordinate of the path, and a solid black marker designates the end of the print path.When printout = 0, serial commands are not sent to the printer, and you can inspect your print path before actually using material to make sure that things will work as expected.

THINGS TO KNOW WHEN PRINTING

Below is a list of a things to avoid when designing print paths. Some have been found to be best practices while others are workarounds for current bugs that need to be fixed.

- When you have two **mk.on(#)/mk.off(#)** commands sent as back to back commands (turn channel 1 off, followed directly by channel 2 on as an example), there's a chance that the Makergear will freeze. If you encounter this problem, a simple workaround is to just include a dummy movement command between the two channel commands, like *mk.move()*. Without any explicit arguments for (x, y, z) coordinates, this will send a (0,0,0) movement command. Assuming you are in relative coordinates, your nozzle won't move.
- If you send a **mk.speed()** command, the speed of all subsequent movements will be changed, and can only revert back to the original movement speed if explicitly set.
- Make sure to always end your print path code with an **mk.close(),** otherwise you'll run into issues when you try and open your COM port again. The steps to fix this problem are addressed in the *Troubleshooting* section.

VERSIONING

All of the code for the Python module, M2PY, as well as a stable version of the modified Marlin firmware can all be found at the following website:
https://github.com/sorna-matthew/m2-python

Anyone is free to use SVN software to copy this directory to a local file location on your computer, especially if you would like to set it up as a host PC for controlling the M2PCS.

One software I recommend for keeping this project up to date is TortoiseSVN:
https://tortoisesvn.net/

Following the simple instructions on this website you can download this software and “checkout” the Github directory above to a location on your PC. Making sure that you update this folder before use to make sure you have the most recent version of this project will guarantee that your PC is able to control the Makergear as bug-free as possible.

If you would like to be given commit privileges, please send an email to:
sorna@seas.upenn.edu, giving me the Github account username that you’d like to use.

I can add you, and give you the ability to push new commits to the project. Make sure to use your best judgement when committing changes to critical files such as the source code for the Python module, or the modified Marlin firmware. For keeping your print paths up to date across multiple machines, I would consider making a seperate folder inside the project directory that you can maintain individually.

The only stylistic choice I ask that each new user follow is that when they commit something to the project, they leave a descriptive comment of what was changed/added/removed, preceded by your initials, that way it can easily be identified who last updated a file. Example:

```
mas: updated core shell Marlin firmware with a new CH1 PIN, and
increased the command buffer from 16 -> 32 bit. Removed unnecessary
SDSUPPORT to free up space.
```

TROUBLESHOOTING

ENDING A FAILED PRINT

Assuming something has happened that requires you to end a currently printing path, all you need to do is click within the IPython console, which should be outputting print statements, and hit the `[ctrl + c]` buttons simultaneously. This should raise an error in Python, which sends an end print command to the printer, automatically shutting off the channels. When you want to print again, close out of the current IPython prompt, and allow another one to load before running your script again.

SPYDER IS UNABLE TO CONNECT TO COM PORT

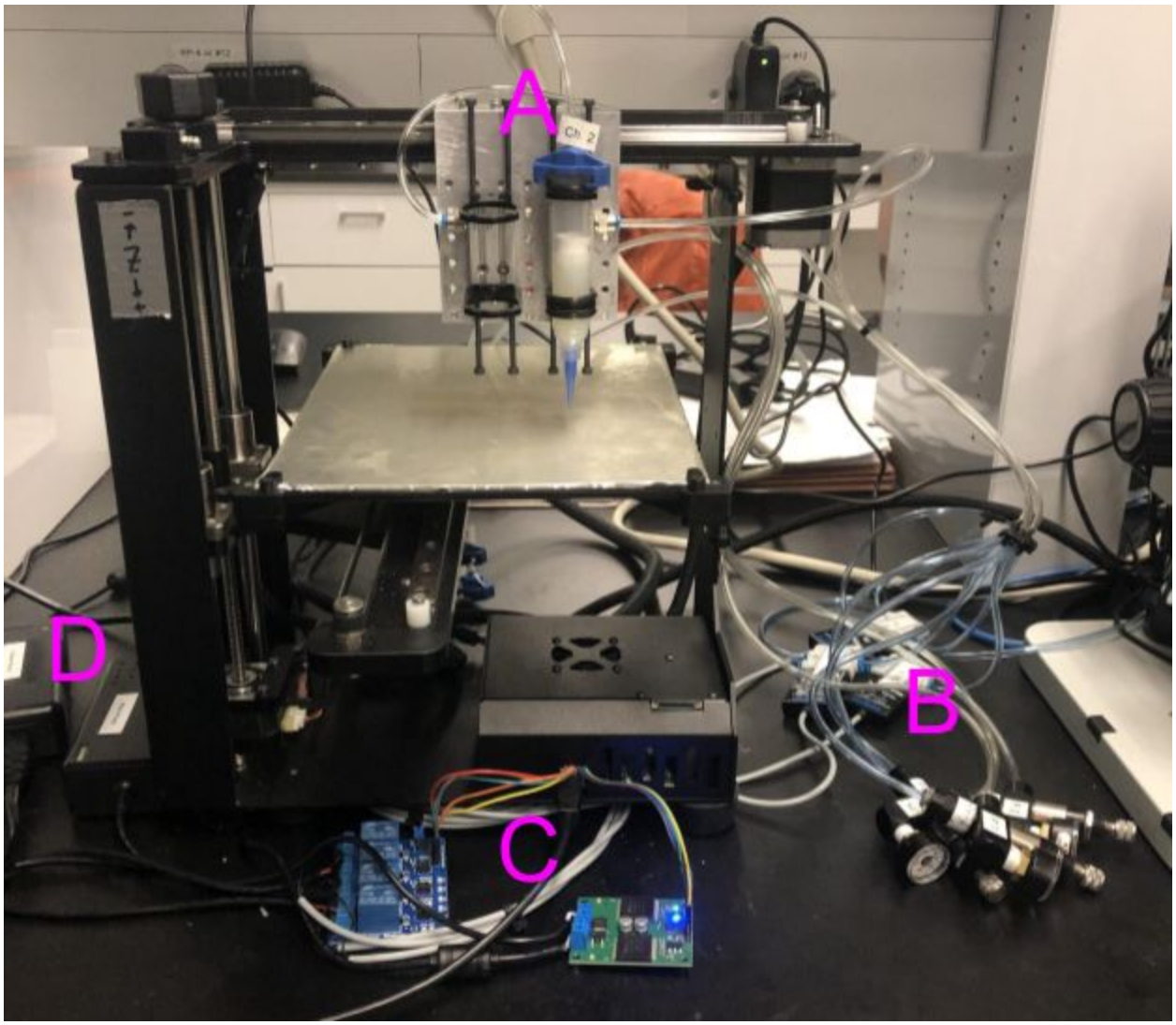
If for any reason you are to end a print prematurely as described above, or if the Python code for your print path contains an error and doesn't properly run, you face running into an issue where the COM port is opened using the function *mp.Makergear()*, and isn't properly closed upon completion using *mk.close()*. If you then try to open the COM port by again calling mp.Makergear(), you'll run into an error saying that access to the COM port is denied. To fix this issue and continue communication with the printer, simply unplug and then plug in the USB connector to the host PC, and then run your code.

If you still cannot connect to the COM port, disconnect the USB connector, reconnect it and close out of the current IPython console and let a new one load. You should now be able to connect to the printer.

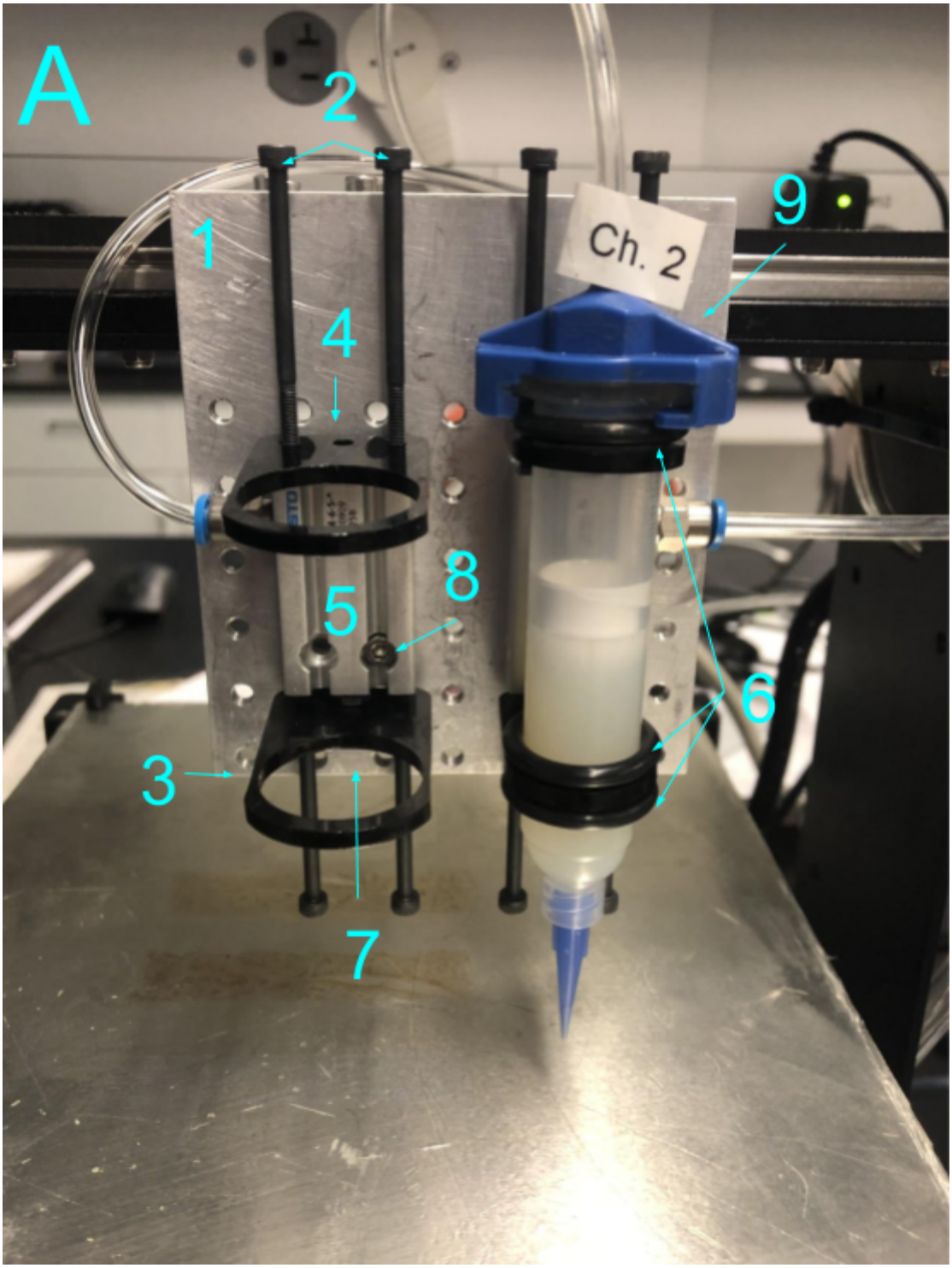
NOZZLE UNDER/OVER EXTRUDES MATERIAL AFTER CHANNEL ON COMMAND

A simple delay function was written in the M2PY library, **set_channel_delay(*delay=50*).** The delay argument in this function sets the delay in milliseconds between the channel turning on, and the print executing the next movement command. Adjust this delay accordingly to allow for material to make it through the nozzle before printing after turning a channel on.

SYSTEM PARTS LIST



FACEPLATE AND PNEUMATIC LINEAR ACTUATORS [A]



A1 Faceplate

Machined by machine shop at Penn. CAD file:
https://drive.google.com/open?id=1Nkcx7Vp1uV9MdUEIYcI8VjP--WVc1Mkm
Extra raw material should be found in cardboard tube by the door in 415

A2 M3x50 Socket Head Cap Screw

Extras in the spare parts drawer

A3 / A4 Upper and Lower Retraction Mounts

These parts which hold the syringes (10 or 30cc) have been laser cut. More can be made / edited from cutting the DXF files found in:
Team Drive >> M2PCS >> System CAD Files >> Retraction Laser Cut Files

A5 Pneumatic Linear Actuators

Linear Actuator
Festo Part #: DPDM-6-5-S-PA
QTY: 3

Push-fit adapter
Festo Part #: QSM-M5-4
QTY: 3

Extras in the spare parts drawer*
* One additional actuator, which has the same throw distance, should also be in the extra parts drawer. This other one however, extends upon actuation rather than retracting. It also has a different mounting surface than the three currently in use.

A6 O-Rings

Used to secure the syringes against retraction mounts.
Extras in the spare parts drawer

A7 M3 Lock Nuts

Used to fasten retraction mounts (A3/A4) to the threaded rod on the pneumatic linear actuators (A5).
Extras in the spare parts drawer

A8 M3 Screw and Hex Nut

Used to fasten linear actuator (A5) to faceplate (A1), Extras in the spare parts drawer

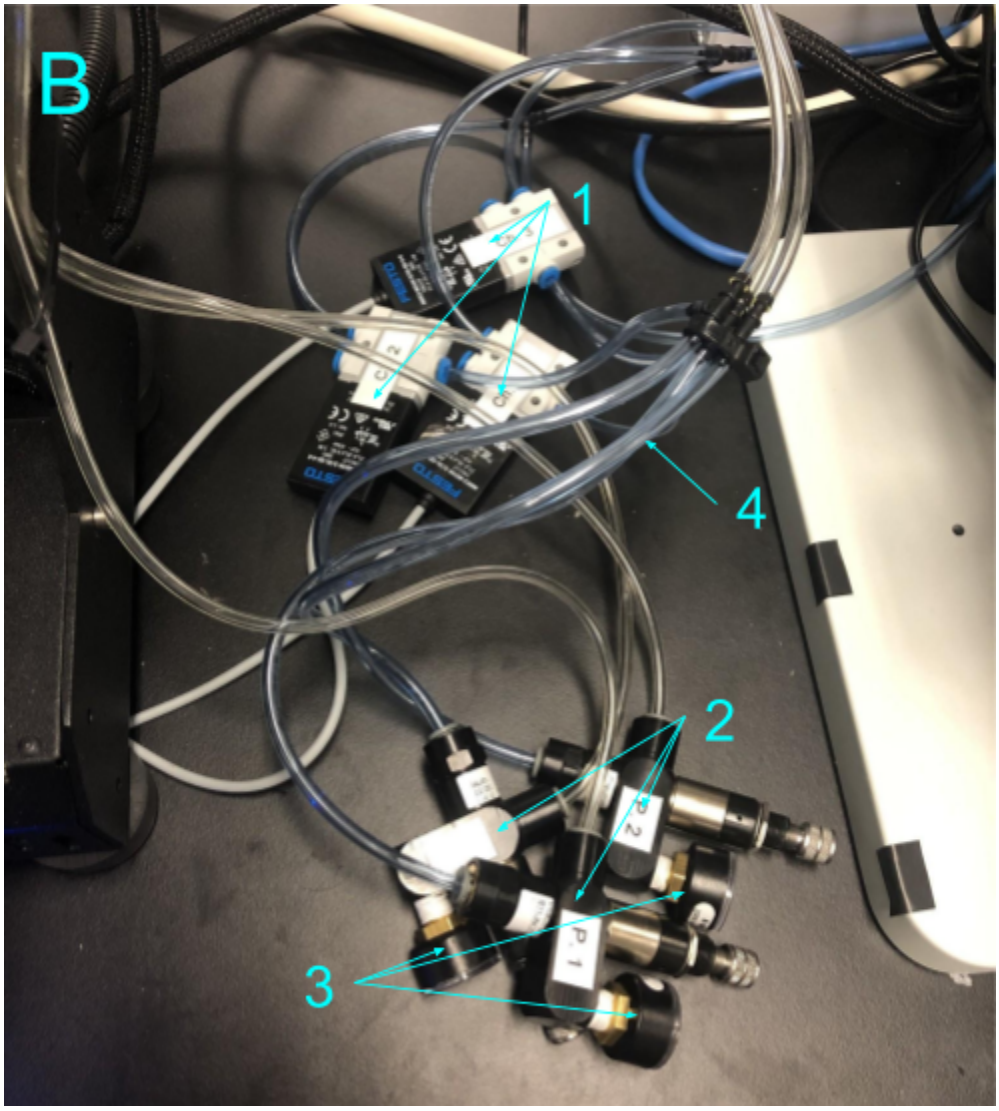
A9 Nordson Syringe Adapter

Ordered from Nordson:
http://www.nordson.com/en/divisions/efd/products/accessories-kits-and-auxiliary-equipment/optimum-adapter-assemblies

A-EX1

Dual threaded adapters that go from 8-32 (into the faceplate) to ¼" (can be used to thread into the camera mounting arm)
Extras in the spare parts drawer

PNEUMATIC SWITCHES AND REGULATORS [B]



B1 Pneumatic Switches

Festo Part #: MHE2-MS1H-3/2G-QS-4-K
QTY: 3

B2 Manual Regulators

Pneumadyne Part #: R11-RK-77-G
QTY: 3

B3 Pressure Gauges

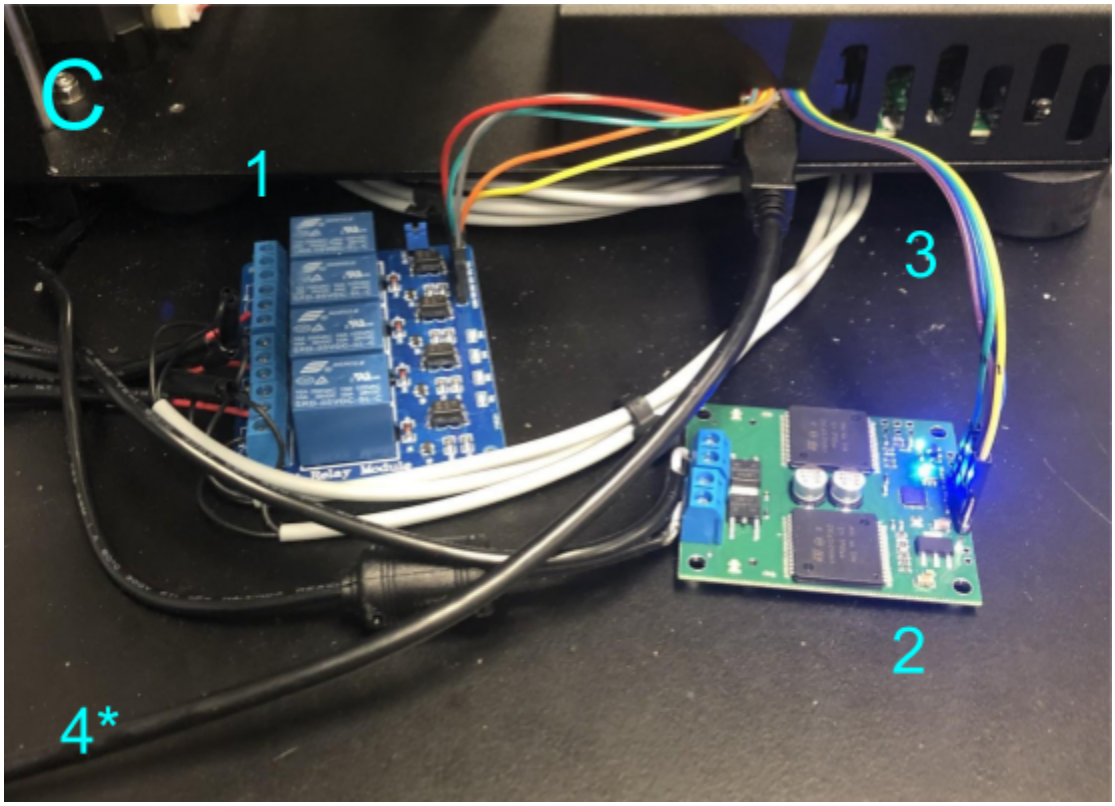
Pneumadyne Part #: PMG-160
QTY: 3

B4 5/32” OD Tubing

All tubing for this system is 5/32” OD / 3/32” ID, expect for the tube that comes off the cutoff valve behind the Shopbot, which is 6mm OD / 4mm ID.

Extra tubing can be found in the spare parts drawer, 5/32” is translucent blue and 6mm is opaque blue.

RELAYS AND MOTOR CONTROL [C]



C1 Mechanical Relay Board

https://www.amazon.com/SainSmart-101-70-101-4-Channel-Relay-Module/dp/B0057OC5O8

C2 Motor Control Board

https://www.pololu.com/product/1112

C3 Jumper Cables

Extras found in the spare parts drawer

C4 DC Motors

Not pictured, currently have two options. 500 RPM which is connected to the printer, and an extra 1000 RPM motor which hasn't been connected or tested yet.

500 RPM https://www.pololu.com/product/3203/specs
1000 RPM https://www.pololu.com/product/3202/specs

EXTERNAL POWER BRICKS [D]

D1 Channel Power Brick

Two-pronged plug, 24V / 2.5 A output

D2 Motor Power Brick

Three-pronged plug, 12V / 5 A output