A0 - sized - Plotter

- Arduino + IDE
- Ramps v1.4
- Python + IDE

Development Notes

This document describes the setup and usage of the following components:

- Arduino	CPU board + IDE + HW control software
- RAMPS	IO board
- Python	IDE + Application software

Document ID: Plotter
Date: 2018-Feb
Version: V1.0

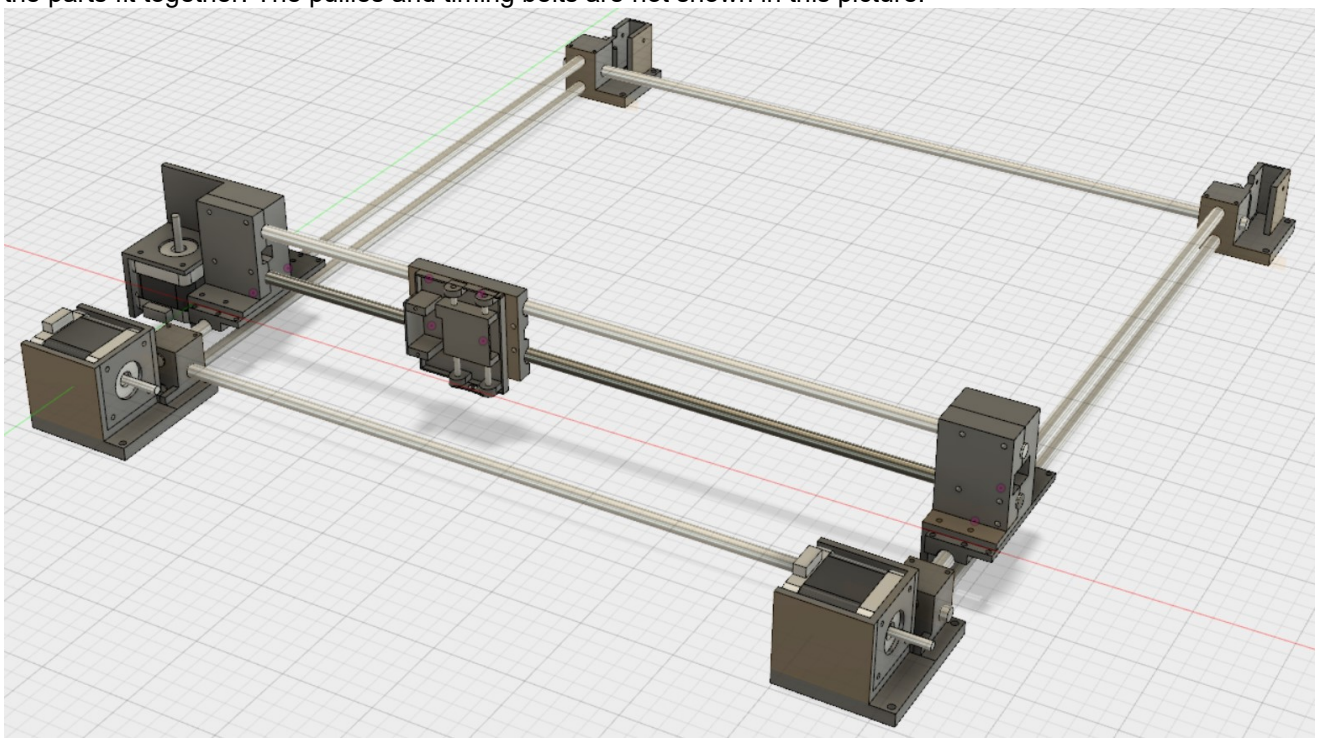
E-Mail: henk.beekhof@gmail.com
Website: htttp://www.hwbbox.com

Table of contents

1 Plotter components

1.1 Overview

The 3D print parts are designed by using Autodesk Fusion 360. The picture below shows an impression how the parts fit together. The pullies and timing belts are not shown in this picture.



1.2 Bill of Materials

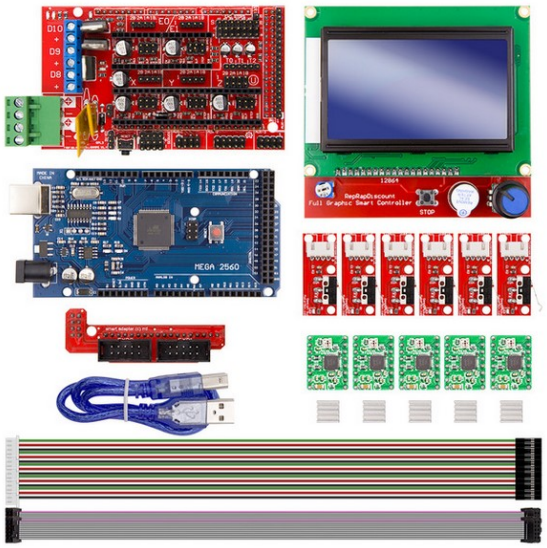
This chapter provides information about the parts which are 3D-printed and how to interconnect them. First a list of other materials is given, which are needed to construct the plotter.

Servo, Motors and related items:

- Mini 5V servo	1x Advice: Buy a heavy-duty servo, not the cheapest with plastic gear
- GT2 Timing belt	6.5 meter, or less dependent on the size you build
- SL42S247A 1.8° Stepper motor	3x (X axis + 2x Y axis)
- Pully 10mm diameter for GT2 belt	3x to fit on the stepper motors
- Pully 10mm ø 5mm	1x X axis return GT2 belt (smooth surface pully)
- Pully 10mm ø 3mm	2x Y axis return GT2 belt (smooth surface pully)
- LM8UU linear ball bearings	8x
- Fan DC 12V, 40mm x 40mm	1x
- Metal tube, ø 20mm, Length 40mm	1x Pen holder
- Metal rods, ø 3mm, Length 60mm	2x Pen up/down gliders
- Metal rod, ø 5mm, Length 20mm	1x axis for X axis return pully
- Linear axis rod, ø 8mm, Length 1m	4x X axis gliders (2x) and 2x Y axis glider (1x)
- Threaded rod, ø 8mm, Length 1m	4x Define XY size of the plotter and they fix the corner parts
- Nuts M8	16x
- Bolts and Nuts M3	various lengths

Electronics:

- Supply 12V DC 10A or more — RepRap supply
- 3D printer Kit — Arduino Mega2560, RAMPS 1.4, Pololu stepper drivers, End stops, ...
  See link below as example (Aliexpress, ~US$ 30)



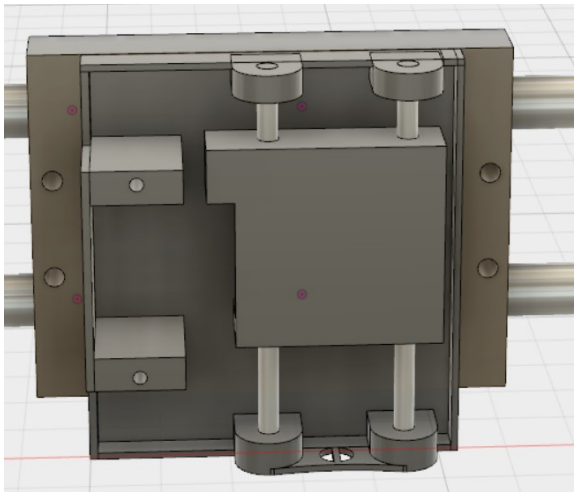
Example link for the 3D printer kit electronics:
https://www.aliexpress.com/item/CNC-3D-Printer-Kit-for-Arduino-Mega-2560-R3-RAMPS-1-4-Controller-LCD-12864-6/32685004463.html?spm=a2g0s.9042311.0.0.mA8KBh

2 The 3D printed components

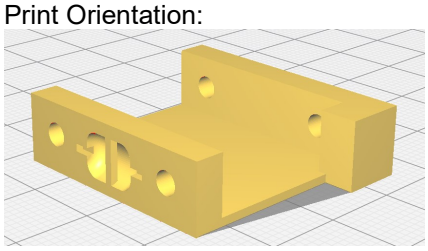
2.1 Pen holder

The pen holder has three parts:

1) Pen Part 1:
   A part which moves up and down to hold the pen. A metal tube will be glued on this part in which the pen will be connected. This part moves up and down over two 3mm metal rods and has a notch at the side where the servo motor pushes this part upwards. The downwards movement is achieved by a small spring at the bottom.
2) Pen Part 2:
   A part which holds the metal rods and the servo.
3) Pen Part 3:
   A part which holds part 2 and which has four LM8UU bearings to move the pen on the 2 rods of the X axis. Further holes are added to be able to connect the GT2 timing belt.

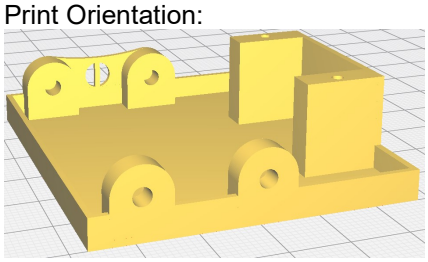


File: Pen_part1.stl
Print Orientation:



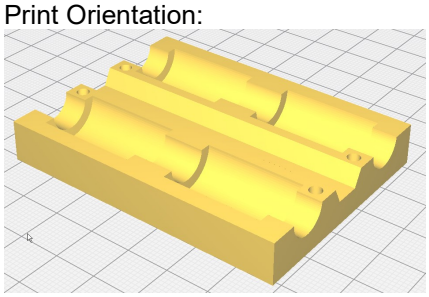
In this view you look at the bottom of this part where you see a small piece of plastic with a hollow area. A small spring should be connected to this, while the other end of the spring is connected to part2 where you find an equal looking small piece of plastic inside a hole, which you can see in the picture above.

File: Pen_part2.stl
Print Orientation:



Use two 3mm metal rods to connect part 1 and part 2, as shown in the first picture. Connect the spring and connect the servo by using two small screws.

File: Pen_part3.stl
Print Orientation:



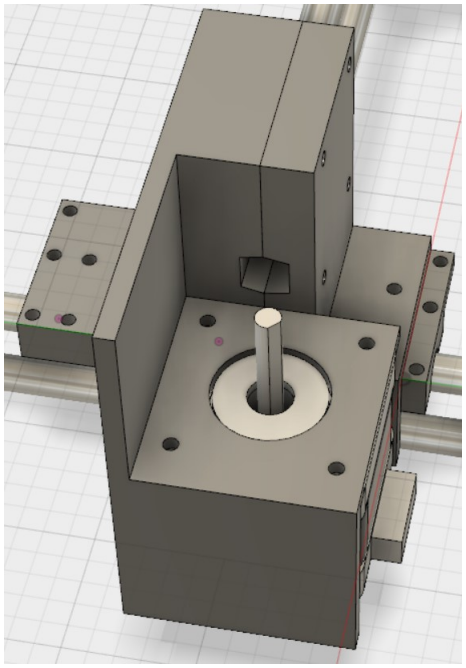
The bottom side of part 2 is glued onto the bottom side of this part 3. Four LM8UU linear ball bearings are pressed (or glued) into the four holes seen on this diagram. The 4 holes are later used to mount the GT2 timing belt.

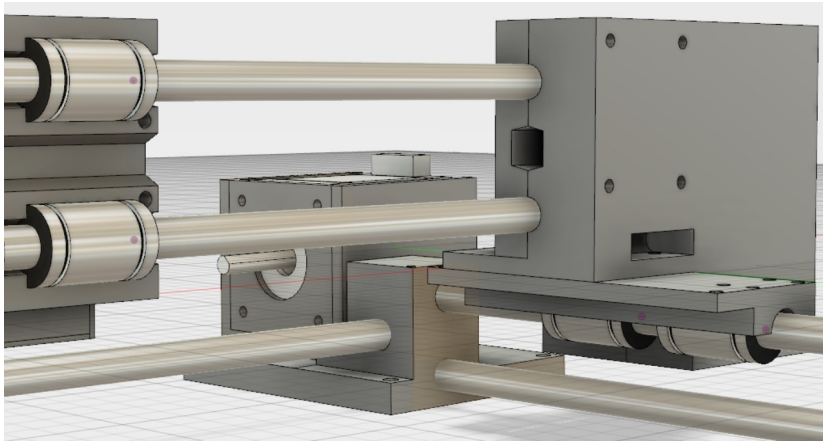
2.2 X-axis left

The X-axis left module has three parts:

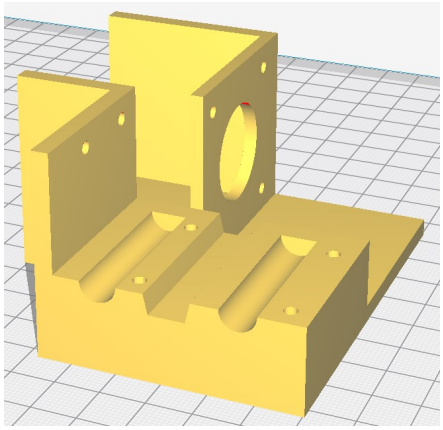
1) X-End A Part 1:
   Holds the motor and the other parts are mounted to this part.
2) X-End A Part 2:
   Fixes the position of the two X-axis rods.
3) Y glider:
   The Y glider is mounted at the bottom. Two LM8UU linear ball bearings are pressed (or glued) into the two holes at the bottom.

This view shows the backside including the spare in part 1 to mount the Y glider.





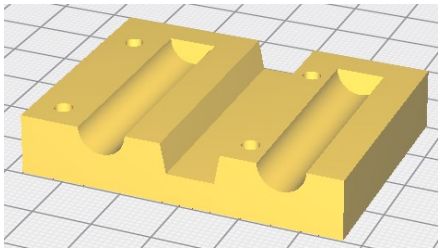
File: X_end_A_part_1.stl
Print orientation



This part holds the X-axis stepper motor. The two half circular holes are used to hold the two X-axis rods on which the pen holder moves. The spare in the middle exists to guide the timing belt through. On top of this, part 2 is mounted to fix the X-axis rods.

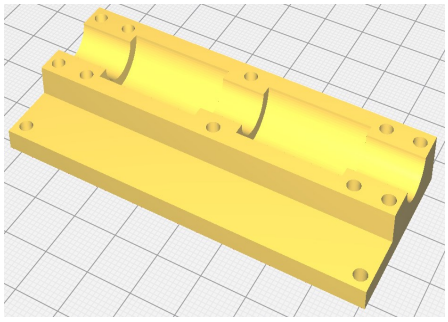
The sides of the parts which hold the motor and Y-axis glider are 4 mm thick. These sides are made solid by using walls of 2mm thick in the slicer program.

File: X_end_A_part_2.stl
Print orientation



This part is mounted on part 1 by using four 3mm bolts.

File: Y_glider.stl
Print orientation



The Y glider print orientation as shown here has six times in a row with different settings in Cura lead to under-extrusion at the top due to the many movements around the 10 holes. In Simplify the first print was ok. In Cura there is a very simple solution: use another print orientation. After turning the part 90 degrees after which the two holes at the front are at the top, printing is ok. This is because the holes are no longer holes in that orientation, there is a lot less retractions. So dependent on the software you might need to change the orientation.

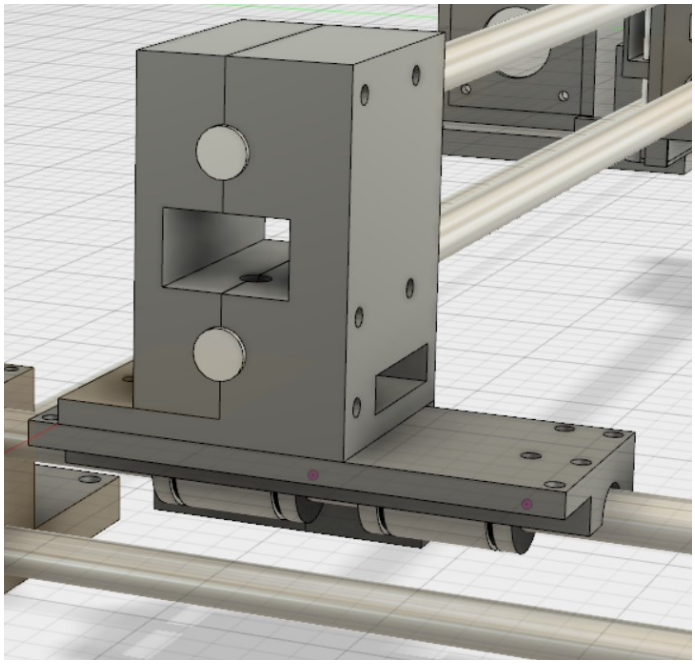
This part is mounted on part 1 by using four 3mm bolts and nuts.

2.3 X-axis right

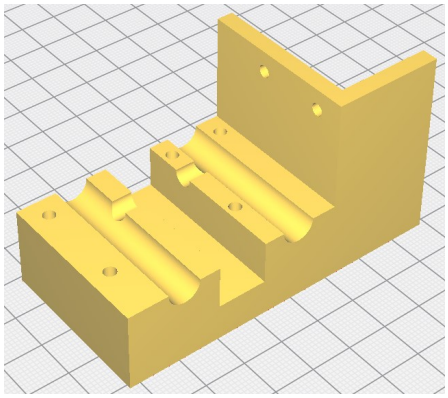
The X-axis right module has three parts:

1) X-End B Part 1:
   The other parts are mounted to this part.
   It is located at the right side on this picture.
2) X-End B Part 2:
   Fixes the position of the two X-axis rods.
   It is located at the left side on this picture
3) Y glider:
   The Y glider is mounted at the bottom. Two LM8UU linear ball bearings are pressed (or glued) into the two holes at the bottom.
   This is the same Y glider as used at the other end of the X axis.

A 5mm axis pully is put in the squared hole in the middle which guides the GT2 timing belt.



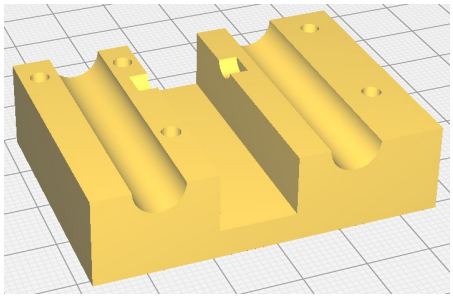
File: X_end_B_part_1.stl
Print orientation



See the overview diagram how the parts are mounted together. A 5mm in diameter small metal bar holds the pully for the timing belt. This metal bar should be positioned in the hollow space in-between the two X axis rods and is kept on its place by mounting part 2.

5mm axis pully for the GT2 timing belt.

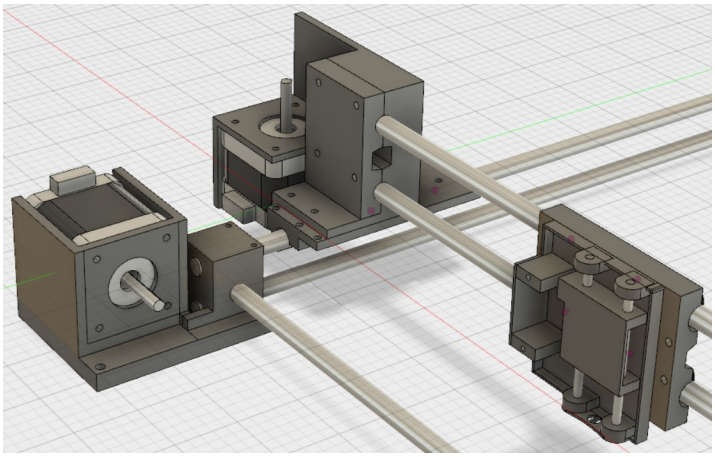
File: X_end_B_part_2.stl
Print orientation



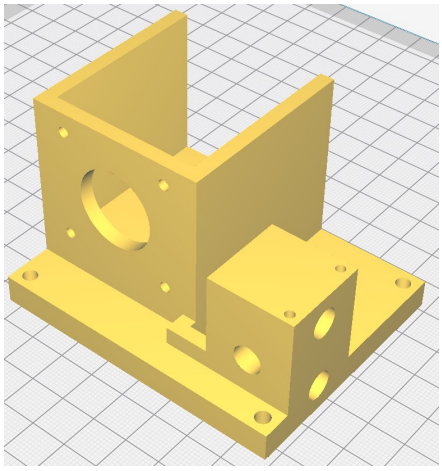
This part is mounted on part 1 by using five 3mm bolts and nuts.

2.1 Y-axis front

The Y-axis front module is one part. It should be printed twice since it is used on both Y axis, left and right. An 8mm threaded rod is used in both X and Y direction to securely control the distance between the four corners of the plotter. The third rod is the smooth 8mm rod on which the Y-gliders move back and forth.
Note: Use nuts on the threaded rods to position the corners. All glider rods are just fed through the holes but are not mounted with glue. Therefor the 8mm holes are a bit smaller as 8mm, use an 8mm hand drill to make the holes larger. The glider rod should 'almost' fit, the threaded rods might move smoothly through the holes because the nuts will keep the position.



File: Y-end-A.stl
Print orientation

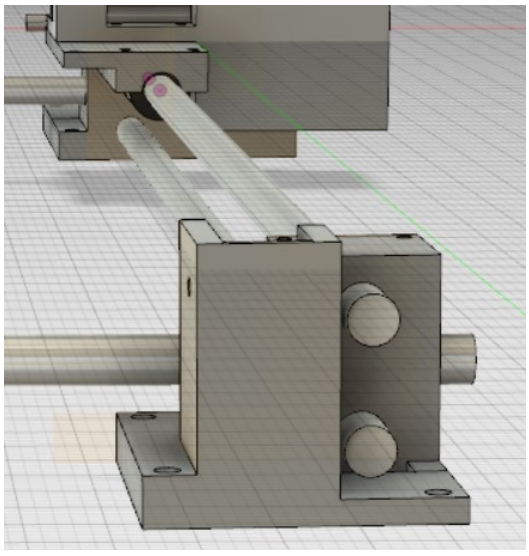


Holds the motor, two threaded rods and one glider rod (on top). At the bottom are several holes to mount the corner part on a wooden plate. This functions as the plotter plate and should be a smooth wooden plate (MDF).

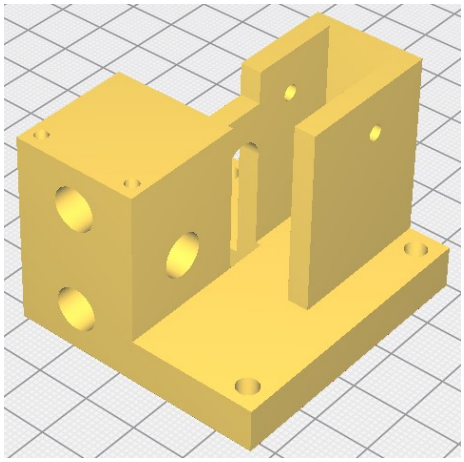
The two holes at the top of the part which holds the rods is used to mount a standard end-switch. It is positioned related to the holes on the Y-glider. A simple bolt and nut touches the end-switch.

2.1 Y-axis back

The Y-axis back module is one part. It should be printed twice since it is used on both Y axis, left and right. An 8mm threaded rod is used in both X and Y direction to securely control the distance between the four corners of the plotter. The third rod is the smooth 8mm rod on which the Y-gliders move back and forth.
Note: Use nuts on the threaded rods to position the corners. All glider rods are just fed through the holes but are not mounted with glue. Therefor the 8mm holes are a bit smaller as 8mm, use an 8mm hand drill to make the holes larger. The glider rod should 'almost' fit, the threaded rods might move smoothly through the holes because the nuts will keep the position.



File: Y-end-B.stl
Print orientation



Holds the two threaded rods and one glider rod (on top).
At the bottom are several holes to mount the corner part on a wooden plate.

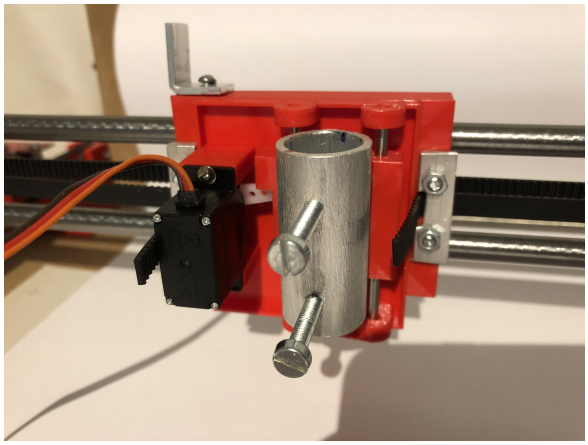
The two holes at the top of the part which holds the rods is used to mount a standard end-switch. It is positioned related to the holes on the Y-glider. A simple bolt and nut touches the end-switch.

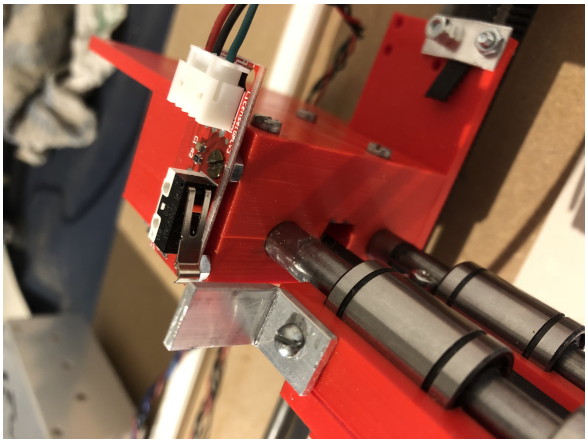
At the backside of this part two 3mm holes are at each side. This is meant to mount a 3mm bolt through which holds a 3mm axis pully which is used for the GT2 timing belt.

3 Construction

General note related to mounting the threaded rods: The M8 holes are just a little bit too small. This is done on purpose. I used an M8 drill to make them fit perfectly.

Finalize the pen holder by gluing the metal tube which holds the pen. I've drilled two holes and used two M4 screws to fix the pen in this tube. You could also use another solution. See the picture below as example.





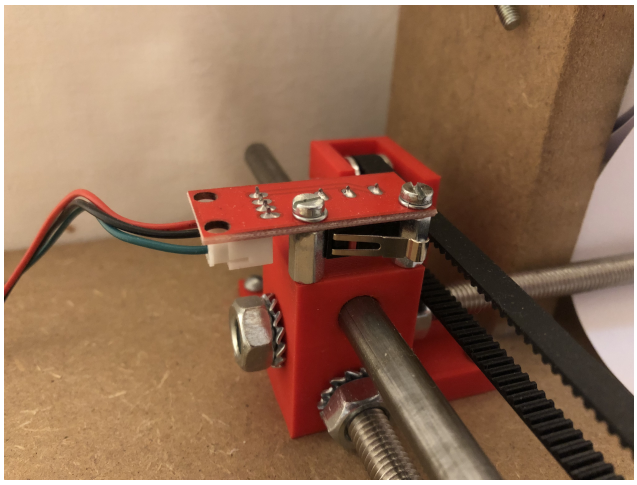
When the X rods, pen holder and X ends, X-motor and pullies are mounted, the GT2 timing belt can be mounted. The picture above shows a simple small piece of metal which is mounted with M3 bolts and nuts to fix the GT2 timing belt.

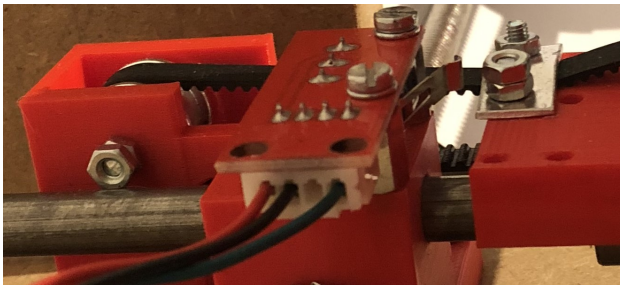
Mount the end stop switch on top of the X axis left part. There are no holes prepared since during creation of these plastic parts I had no idea where the X axis end stop would be mounted. I use a 'tap' screw to fix this PCB to the plastic part. Underneath the PCB of the end stop switch an M3 nut is used as separator.

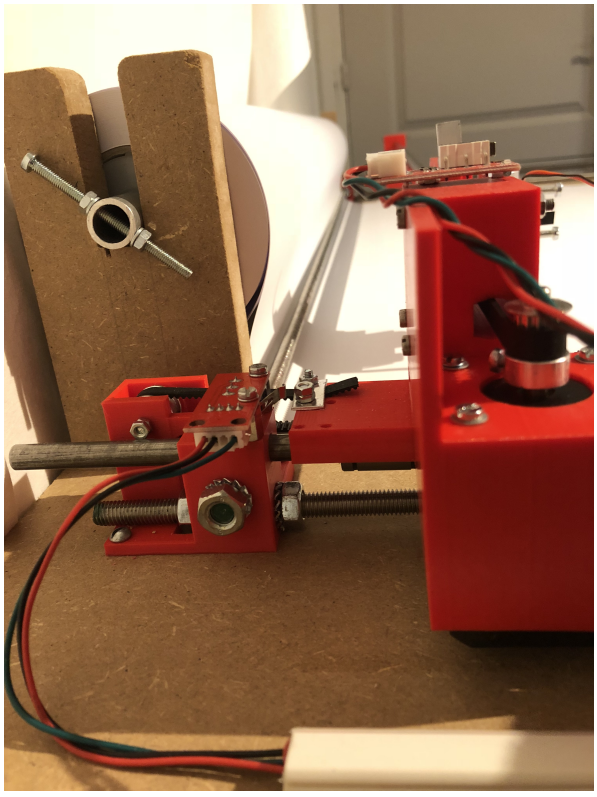
Use a small piece of metal and fix it on top of the pen holder (see picture), this piece of metal activates the end stop switch.

Mount the Y rods and threaded rods to all corners as shown in first below picture.
Make sure the pen holder moves smoothly in both X and Y direction. Fix the threaded rods with the M8 nuts.
The Y movement is best tested without the timing Y belts.
After the distances between the corners are fixed I mounted the corners on a wooden plate to have one 'ground plane' which also holds the electronics.

Mount the pullies by using an M3 bolt and nut on both Y-end-B parts.
Mount the end stop switch on the Y-end-B part, in the upper left corner of the plotter. This switch is triggered by a bolt on the X_end_A_part_1 which connects the Y-glider. See the second picture below for the details.
Mount both Y motors and pullies, then the GT2 timing belts can be mounted. The same method is used as on the X axis, see the second picture for this detail: The metal strip which holds the belt.

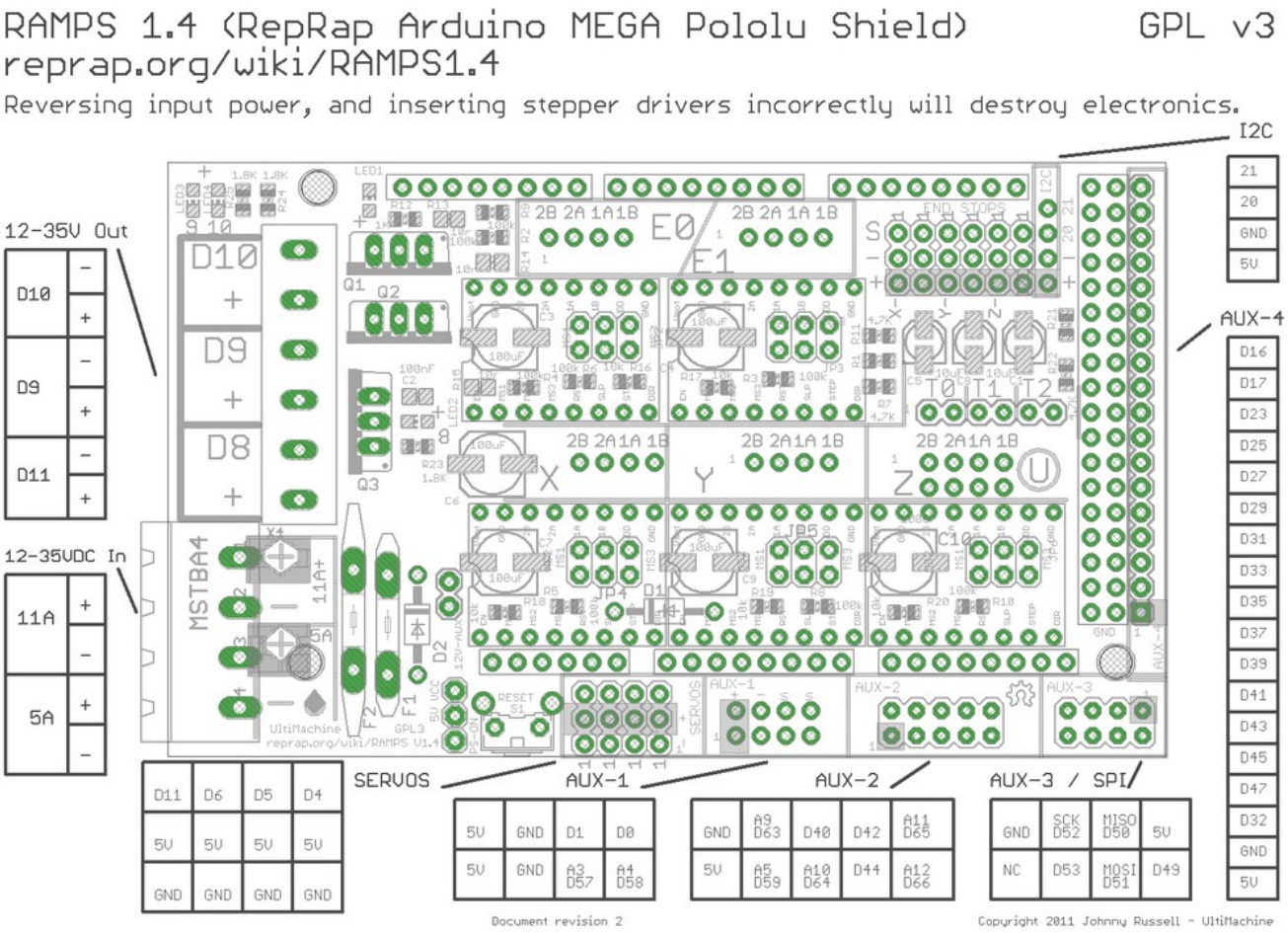






Optionally you can add a holder for the roll of paper.
Add the wiring as shown in the next chapter.

4 RAMPS Connections



Pen Servo
The pen servo is connected to the servos pins, using the first slot (D11, 5V, GND)

XY stepper motors: select micro steps and mount the Pololu A4988 driver boards
The X motor uses the X driver
The Y motors uses the Z driver due to the two connections

Use JP4 for the X motor
Use JP6 for the Y motor (which is Z on the board)

Micro stepping is used to dramatically reduce the vibrations in the system. 1/8 step size is used.

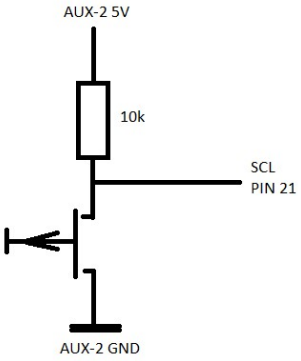
Put a jumper on the first and second (1A,1B) of JP4 and JP6.
Then mount two driver boards and use the delivered heatsinks.

jumper 1	Yes/No 2	3	step size
no	no	no	full step
yes	no	no	half step
no	yes	no	1/4 step
yes	yes	no	1/8 step
no	no	yes	1/16 step
yes	no	yes	1/32 step
no	yes	yes	1/64 step
yes	yes	yes	1/128 step

The X motor is connected to the X interface (2B,2A,1A,1B). Both Y motors are connected to the Z interface, use the same ordering of connecting these 4 wires to the coils of the X and Y motors (this depends the direction).

Fan
Connect the fan to the D10 plus and minus connector at the upper left corner of the board.

Emergency Stop
Since stepper motors can create quite some force on the mechanics, it might be a good idea to create an emergency stop. This also guarantees a direct stop, while pulling the power plug still powers the motors for some time dependent on the capacity of the used supply.



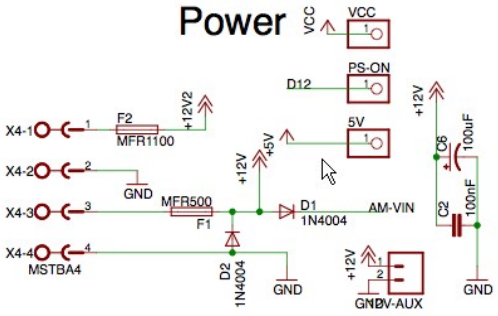
A pin is used which is connected to an interrupt pin of the microprocessor. And this pin may not be used for another function (such as the serial interface). One option is to use the I2C clock signal SCL since the I2C bus is not used for the plotter.

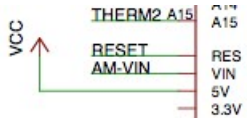
See the diagram, you could use a small PCB and mount it on AUX-2.
It is possible to use an internal pullup, but here it's chosen to put it on this small PCB which also holds the switch. A wire is used to connec to pin 21 in the upper right corner of the RAMPS board.

Power supply
Connect the plus of the power supply to connector 5A, + pin.
Connect the 0V (GND) of the power supply to the 5A, - pin.
It is not required to connect the power to connector 11A. That is only required if connector D8 or D9 is used. Note the error in the layout as shown above, it shows at the upper left corner D10, D9, D11, that should be D10, D9, D8. Mark that D11 is used for the servos.

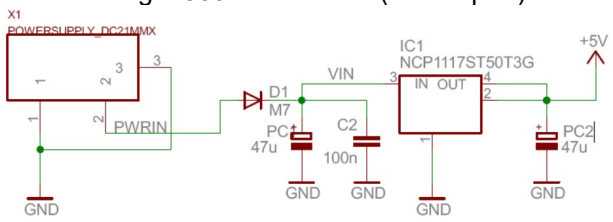
The Arduino board is powered by the RAMPS power. The connector 5A on the layout is X4 in the schematic, see pin 3 and 4. Pin 3 is connected via a diode D1 to AM-VIN. In the interconnection diagram it is shown that this RAMPS signal AM-VIN is connected to VIN on the Arduino board. The voltage level is 12V minus the diode threshold, so roughly 11.3 Volt. On the Arduino schematics VIN is the input voltage of the 5V regulator. It is not required to supply the Arduino board through the power plug X1.

RAMPS V1.4 Schematics (Power part)





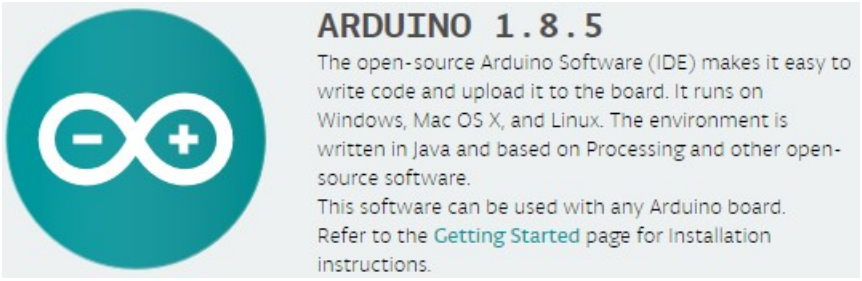
Arduino Mega2560 Schematics (Power part)



5 Arduino - Firmware

5.1 Integrated Development Environment (IDE)

The Arduino IDE is downloaded from: https://www.arduino.cc/en/main/software
Version which is used: 1.8.5



5.2 Location of the code + way of working

By default, the Arduino IDE stores the projects in c:\Documents\Arduino\

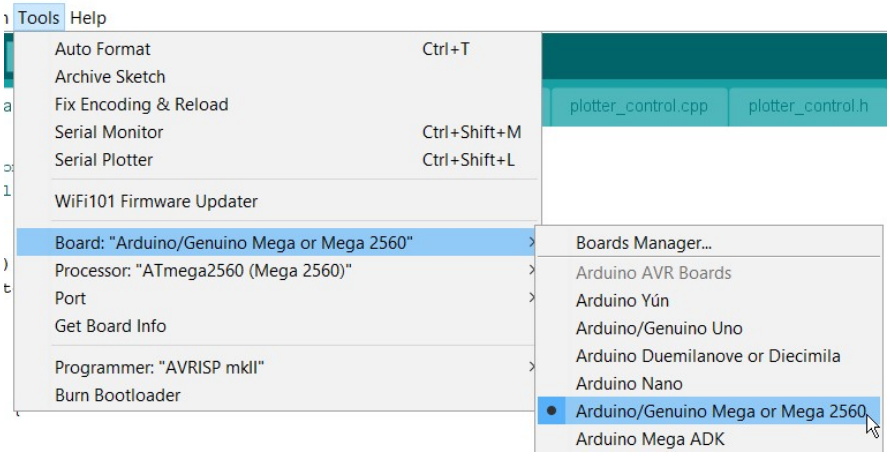
The (to be) developed code is not checked-in and checked-out by using a revision system, for now the way of working is to save a new project on a major new step in the development process. The Arduino IDE makes this very easy, since a 'Save as' saves all files of the project into the new project folder.
Method: Use 'Save as', move up to the projects folder, define a new name, which is the new project folder.

Differences in Arduino code related to other systems:
A project has two procedures, called setup() and loop(), after a powering on the Arduino board, first the setup() is executed, then the loop() is started.

5.3 Setup Board and Port to compile and upload software

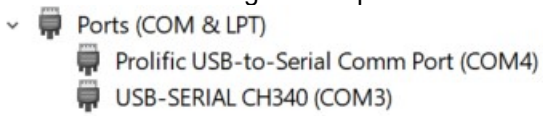
5.3.1 Define the target board

In Arduino IDE: Tools -> Board: -> Select the Mega2560 board.



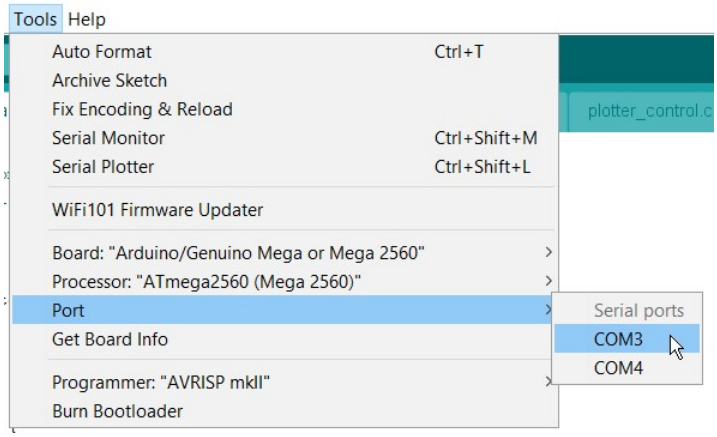
5.3.2 Define the serial port

Start the Device Manager and open the section Ports to find out which port is used.

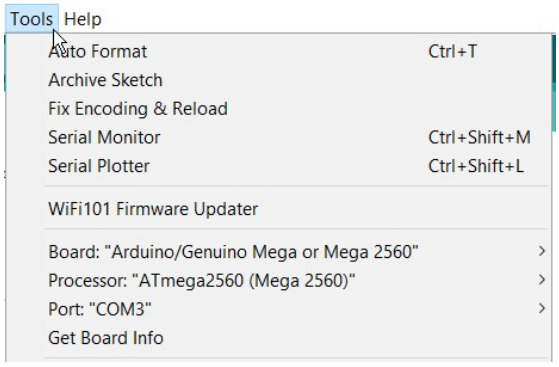


In this case COM3 is used (USB-SERAL CH340)

In Arduino IDE: Tools -> Port -> Select the correct COM port found in the Device Manager, (COM3)



After both definitions the tools menu directly shows these settings:

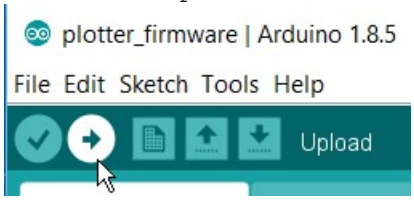


5.3.3 Test: compiling and uploading the code

If you want to test these settings then write a simple program. For instance:

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  while (1) {
      Serial.println("Ok");
  }
}
```

Press the Upload button which also compiles the code if not yet done:



Then open the Serial Monitor, that should show lots of 'OK's.
Tools -> Serial Monitor

5.4 Software architecture / Partitioning

The Arduino board in combination with the RAMPS board control the actual hardware, motors, servos and sensors. The main split in functionality is based on debugging facilities and ease of use. The Arduino IDE is ok to develop pieces of software, but it is not an IDE where you have full visibility and full controllability over the code during development. Features like single stepping, etc are missing while this is for ages fully possible by using an Instruction Set Simulator (ISS) with models of peripherals. So, the choice is to keep the software on the Arduino to the bare minimum, while it should take care of all the hardware control.

The abstraction level is a set of commands to communicate with an application running on a pc or notebook. That application should be developed in an IDE which has all the required features to develop good quality software and bring ease of use during development. It is anyway a good split to let the Arduino take care of the hardware and have the application running elsewhere. That allows also a richer set of application features, like a 'simulated' result of what the plotter would do.

The Arduino application will use a serial interface to receive commands. These commands are executed one by one. Commands are:

Command	Description
TURN_OFF	Pen Up + Disable motors and servos
HOMEX	Home the X position, set this position as zero
HOMEY	Home the Y position, set this position as zero
HOMEXY	Call HOMEX and HOMEY
PU	Pen Up
PD	Pen Down
MT x0,y0	Move to (x0,y0)
DL x0,y0,x1,y1	Draw Line from (x0,y0) to (x1,y1)
DS x1,y1	Draw Line Segment from current position to (x1,y1)
DR x0,y0,dx,dy	Draw Rectangle left upper corner is (x0,y0) with sides dx and dy
SSX speed	Set Speed X motor, overrule the default
SSY speed	Set Speed Y motor, overrule the default
SPUP position	Set Pen Up Position in [us], overrule the default
SPDP position	Set Pen Down Position in [us], overrule the default
FAN on	Turn the fan on or off, 1=on, anything else is off

5.5 Arduino source code files

5.5.1 main file - plotter_firmware

This is the entry point of software execution. The bootloader which is default available on the Arduino board takes care of being able to download the user software and it calls the setup() and loop() functions after a reset.

Several variables are defined, the setup() function takes care about initialization.

The loop() function holds the main endless loop which gets commands from the serial interface, interprets the command and executes it. This main loop looks like this:

```
void loop() {
  cmd_info ci;
  char command[COMMAND_LENGTH];

  while (1) {

    // Fetch a command + optional parameters from the serial interface
    get_command(command);

    // Decode the command to an instruction and optional parameters
    interprete_command(command, &ci);

    // Execute the instruction
    if (execute_command(&ci)) {
      Serial.println("Ok");
    } else {
      Serial.println("ERROR");
    }
  }
}
```

5.5.2 command_interface.cpp

The code in this file handles the serial commands received from the application on the other end of the serial interface. It basically contains three functions. Function 'get_command' reads a single command from the serial interface. Each command is terminated by a \r character. Function 'interprete_command' splits the command into the actual instruction and the optional parameters of this instruction. Function 'execute_command' checks the actual instruction and calls the execution function of that instruction with the by the instruction required amount of parameters. Example code of a single instruction:

```
  // Command: MT x1 y1  - Move To
  if (strcmp(ci->instruction, "MT")==0) {
    if (ci->nr_cnt<2) return ERROR;
    moveToXY(ci->nr[0], ci->nr[1]);
    return OK;
  }
```

5.5.3 plotter_control.cpp

This file contains the software which controls the hardware. It contains the execution functions of the commands and initialization code as well as other functions required to support the commands.

Most of the commands control also the pen up and pen down movements. Meaning the command moveToXY should not draw a line, so the penUp function is called and an execution function which does draw a line makes also sure that the pen is in the down position.

The pen movement commands contain fixed numbers for the servo position. This is likely something which needs to be changed on your version. You can either change the defaults or use the commands SPUP and SPDP to overrule these defaults. Here is the code:

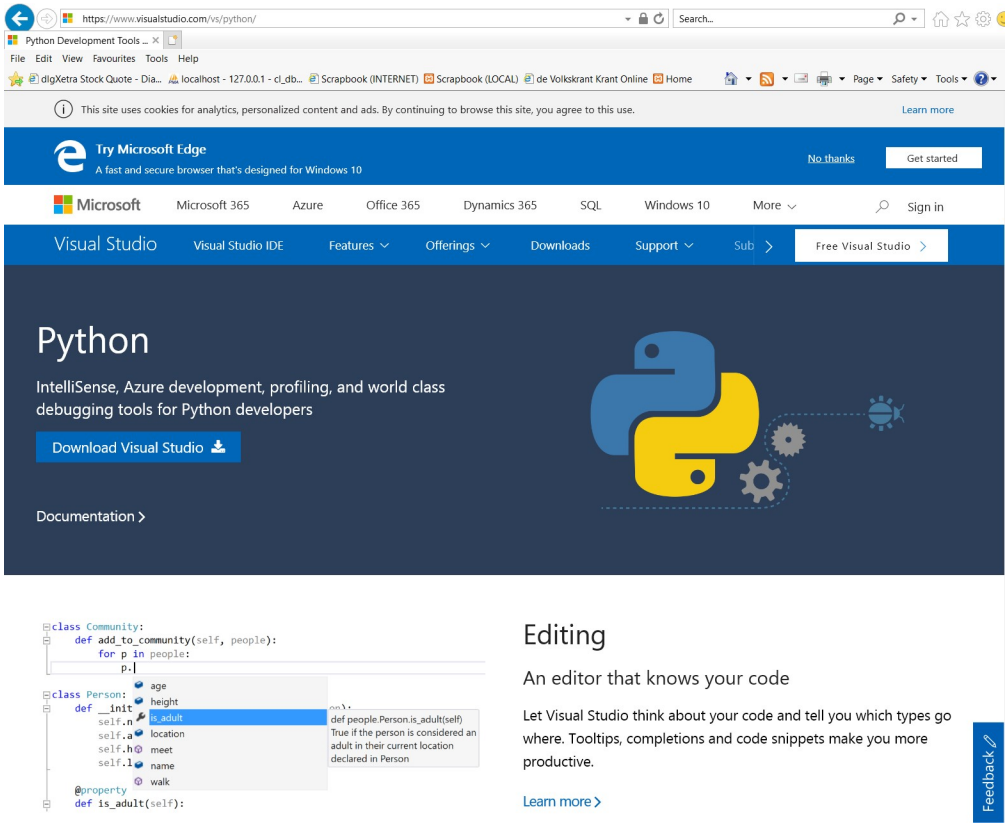
```
void penUp() {
  penA.writeMicroseconds(1520);  // PenA Up
  penIsUp=true;
  delay(100);
}

void penDown() {
  penA.writeMicroseconds(1250);  // PenA Down
  penIsUp=false;
  delay(100);
}
```

6 Python application

6.1 Integrated Development Environment (IDE)

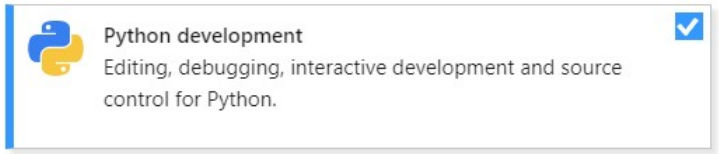
There are several python IDE's available on the internet. IDLE seems to be promoted a lot, while this IDE does not have the basic functionality of showing line numbers. Thus, an error on line 437 of a file leaves you wondering how to find the location. I've also tested several other IDE's, most of them lack basic functionality, such as setting breakpoints, watching variables, single stepping, etc.

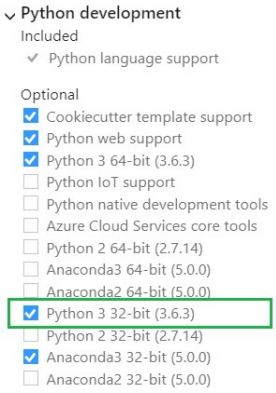
For the application development I use Visual Studio 2017 which you can download and use for free.
URL: https://www.visualstudio.com/vs/python/



Before installation you can specify what to install. After installation it is possible to access the same setup screen by using: Tools -> Get tools and Features. Select Python development and select which python version(s) you would like to add. For the plotter I've used Python 3, 32 bits. If you select the same, then you will not run into issues that a certain package is not available.

Note: Python 3, 64 bits will also work with the setup described in this document.





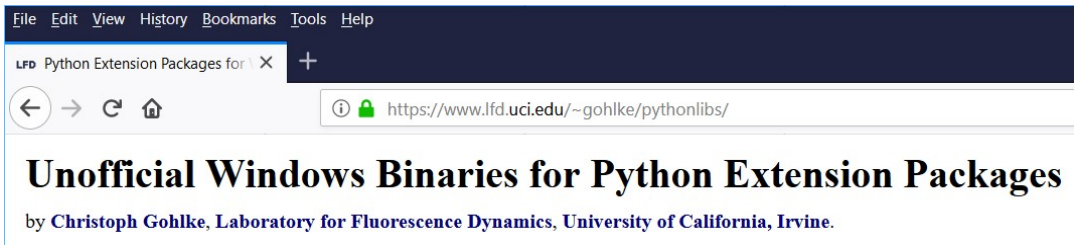
After installation, you can access the Python setup in Visual Studio. Especially when you have installed several python versions or if you already had a python version on your system, then it's a good idea to check if the version where you want to work with, is selected. Select: View -> Other Windows -> Python environments. Select: Python 3 32-bit (3.6.x)

6.2 Installation of packages

The benefit of Python is that it is relatively easy to create applications, which do complex things. This is based on the usage of pre-developed packages by others. The drawback which belongs to this benefit is the need to install these packages. And the version of the package should match with your Python version and in some cases also with the architecture of the PC (32/64 bit).

Visual Studio has the capability to install packages. However, that did not work. So, I recommend to do it manual. Here is how:

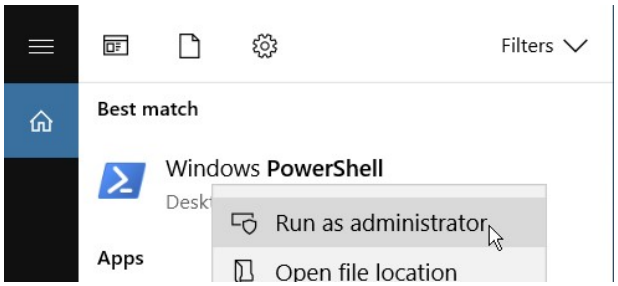
1) Download these packages from: http://www.lfd.uci.edu/~gohlke/pythonlibs



a) numpy-1.14.2+mkl-cp36-cp36m-win_amd64.whl (because it's on a 64 bits PC)
b) PIL-2.0+dummy-py2.py3-none-any.whl
c) pyserial-3.4-py2.py3-none-any.whl

Copy or Move these packages from your download folder to the python directory:
C:\Program Files (x86) \Microsoft Visual Studio\Shared\Python36_86

2) Open Windows PowerShell and run it as administrator.



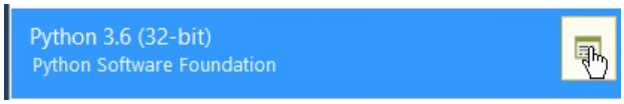
CD to the python directory: C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_86

Run this command:

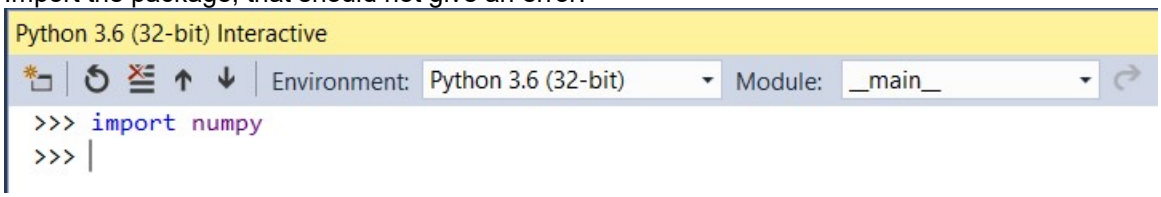
```
py -3.6 -m pip install numpy-1.14.2+mkl-cp36-cp36m-win_amd64.whl
```

Do this for all downloaded packages.
Note: If you type the first part of the package name, then type TAB, the tool will auto complete the name.

To test this, you can open an interactive window. Select: View -> Other Windows -> Python environments
Press on the button where the hand is located:



Import the package, that should not give an error:



6.3 Opening the python application in Visual Studio

Store the files where you would want to have them. The directory and file structure is:

```
<your path>\Plotter\Plotter.sln                Solution file
<your path>\Plotter\Plotter\Plotter.pyproj     Project file
<your path>\Plotter\Plotter\*.py               Source files
<your path>\ASCII\*                            Files used by ascii art
<your path>\img\*                              Files used to print an image as lines
```

Double click the solution file. This will load the application in Visual Studio.
For ease of use (direct access to your project files) it is convenient to have the Solution Explorer visible. To do so use the menu "View"->"Solution Explorer".

6.4 The Python application

Unpack the zip file and store the files on your computer. Within Visual Studio, open the file "Ploter.sln", this is the solution file with the project details, and this will load the project into Visual Studio.

File plotter.py:
This is the main entry point. The application does not have a Graphical User Interface. It is not meant to be a tool to be used by 'everybody'. If you want to add a GUI, then you should do so. In this file an application is selected by setting a variable like: "application=8". The following if statements jump to that application. Each application can be seen as a different function for the plotter.

Tip: In Visual Studio, put your mouse on a function, like print_ascii_art_file(), then press F12 to jump to that function.

File proj_settings.py:
This file takes care about initialization. The most important part is this section:

```
# Initialize Project Settings
#ps.set_plotmode(ps.PLOT)
```

The second line defines if you actually want to plot or simulate the chosen application. Remove the comment (#) character to plot.

This setting is checked in several functions, for instance in the function which draws a line. It either sends a drawline command to the plotter or draws a line on an image. When the application is completed the image is shown on screen. This gives you a good idea of what the end result will be. There is of course some inaccuracy because the plotter is roughly 41000 x 41000 pixels while a 'normal' screen is at least a factor 10 smaller. But the result is good enough to decide to really plot the result or not.

The other files are not discussed here. See the different applications and follow the program flow which starts in the plotter.py file.

7 Available plotter applications

7.1 ASCII art

The book "Python playground" from Mahesh Venketachalam shows in chapter 6 how to convert an image to an ascii text file which represents the image if you look from a distance. For dark areas characters with a lot of black are used (@#$) and for light areas characters with less black are used (space.,;). This algorithm is used as a starting point.

Use 'application=2' to convert an image to an ascii text file.
See file 'func_ascii_art.py' on line 96 and 99 to define the input image file and output ascii text file.
Play around with the scale and cols variable to get the result you want.

The result is a text file which is the input for the plot process.

Use 'application=4' to plot the ascii text file.
Define the file to be used in 'func_application.py' on line 26.
First simulate the plotter output by this setting in the file 'proj_settings.py':

```
#ps.set_plotmode(ps.PLOT)
```

Simulated result shown in 'Paint':

Photo of the plotted result, the photos around the plot are A3 sized photos, to show the size of the plot:

7.2 Images as lines

The idea is to plot an image as a set of horizontal lines. The image is loaded as a grey scaled image where the grey scale ranges from 0 (black) to 255 (white). The image is scanned line by line. When within a line a pixel is found which is darker than the middle value (127) then a to be plotted line starts. The scanning process now looks for the end of this line. When found it sends a drawline command to the plotter.

In the example shown below an image line is drawn, then a line is skipped (shown as blank line), etc.
For performance reasons the scanning direction swaps at the end of the processed image line, this prevents a lot of unwanted moves of the pen and limits the plot time.

For such an image the X motor becomes too hot without cooling. This leads to a wrong plot due to failures of the stepper motor. For this purpose, the fan is added to the X motor.

Use 'application=8'
In the file 'func_abstract_img.py' on line 85 the source image is defined. On line 90 you can set a scale factor.

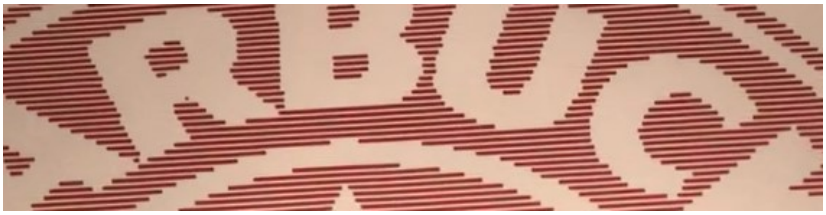
7.2.1 Example file: Starbucks logo

Scale is set to 100.

Simulated result shown in 'Paint':

Detail of the picture in 'Paint':

The plotter at work to plot the image, using a thick red marker.
Note: This original picture is only 375 x 375 pixels and is blown up to roughly 80 cm wide.



7.2.2 Example file: Darth Vader

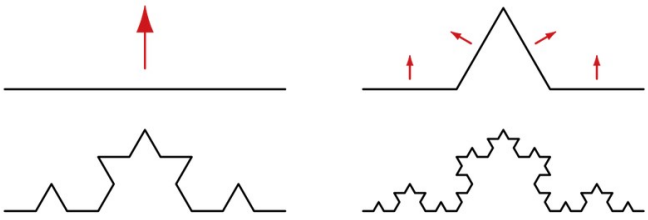
Scale is set to 60.

Simulated result shown in 'Paint':

The plotted result:

7.3 Koch curve

The Koch curve is a mathematical curve, which originates from Helge von Koch in 1904. It is also called a fractal. The theory is to device a line into 3 similar parts and create a 'bubble' in the middle part. This results into four line segments, with which the same procedure is applied, etc. etc. In software this results into a recursive program.



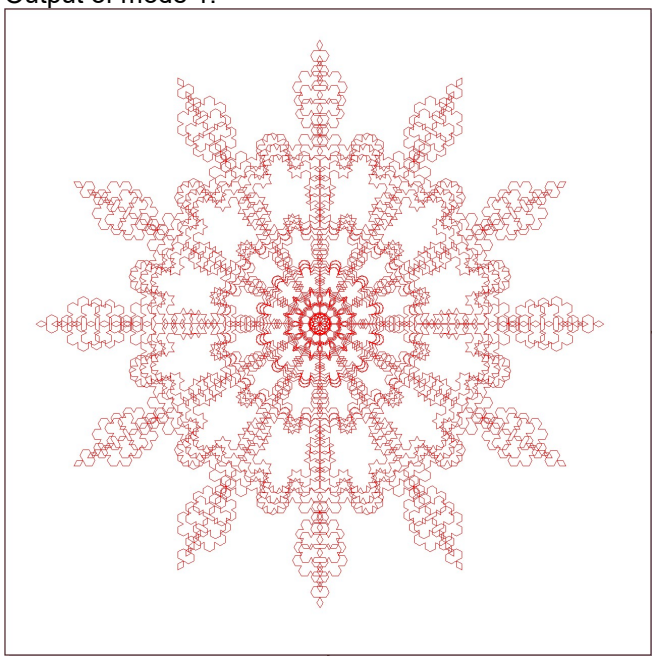
This curve becomes longer after each iteration while the line segments become smaller. A practical explanation related to this curve is: If you would measure a coast line of for instance Sweden or England, then the measured length depends on the size of the measurement tool you use. The smaller this tool, the more accurate the measurement and the longer the coastline.

With this curve a triangle is drawn. It is a choice if the curve moves inwards the triangle or outwards. This triangle is then rotated and drawn again. Such simple algorithms lead to complex figures. See example picture below.

Use 'application=6'
This creates an image file, the code is downloaded from the internet
This does not plot, it is the source code which is used to develop application 7.

Use 'application=7'
This creates a picture or plot based on the code in function 'print_koch()'. Several modes are part of the code, which is just playing around with the principle of drawing triangles, based on the Koch curve, including rotations, etc. This is the place to change the code and simulate the result.

Output of mode 1:



Output of mode 4: