**Choosing a motor drivers**

1. **DRV8825**

This breakout board for TI’s DRV8825 microstepping bipolar stepper motor driver features adjustable current limiting, over-current and over-temperature protection, and six microstep resolutions (down to 1/32-step). It operates from 8.2 V to 45 V and can deliver up to approximately 1.5 A per phase without a heat sink or forced air flow (rated for up to 2.2 A per coil with sufficient additional cooling). The driver has a pinout and interface that are nearly identical to those of our [A4988 stepper motor driver carriers](https://www.pololu.com/product/1182), so it can be used as a higher-performance drop-in replacement for those boards in many applications. This board ships with 0.1″ male header pins included but not soldered in.

## Overview

This product is a carrier board or breakout board for TI’s DRV8825 stepper motor driver; we therefore recommend careful reading of the [DRV8825 datasheet](https://www.ti.com/lit/ds/symlink/drv8825.pdf) before using this product. This stepper motor driver lets you control one [bipolar stepper motor](https://www.pololu.com/category/87/stepper-motors) at up to 2.2 A output current per coil (see the Power Dissipation Considerations section below for more information). Here are some of the driver’s key features:

* Simple step and direction control interface
* Six different step resolutions: full-step, half-step, 1/4-step, 1/8-step, 1/16-step, and 1/32-step
* Adjustable current control lets you set the maximum current output with a potentiometer, which lets you use voltages above your stepper motor’s rated voltage to achieve higher step rates
* Configured for mixed decay mode
* 45 V maximum supply voltage
* Built-in regulator (no external logic voltage supply needed)
* Can interface directly with 3.3 V and 5 V systems
* Over-temperature thermal shutdown, over-current shutdown, and under-voltage lockout
* Short-to-ground and shorted-load protection
* 4-layer, 2 oz copper PCB for improved heat dissipation
* Exposed solderable ground pad below the driver IC on the bottom of the PCB
* Module size, pinout, and interface match those of our [A4988 stepper motor driver carriers](https://www.pololu.com/product/1182) in most respects (see the bottom of this page for more information)

We also have a variety of other [stepper motor driver options in this same form factor](https://www.pololu.com/category/269/16-pin-stepper-motor-drivers) with different operating profiles and features.

This product ships with all surface-mount components—including the DRV8825 driver IC—installed as shown in the product picture.

This product ships individually packaged with 0.1″ male header pins included but not soldered in; we also carry a [version with male header pins already soldered in](https://www.pololu.com/product/2982). For customers interested in higher volumes at lower unit costs, we offer a [bulk-packaged version without header pins](https://www.pololu.com/product/2977) and a [bulk-packaged version with header pins installed](https://www.pololu.com/product/2987)



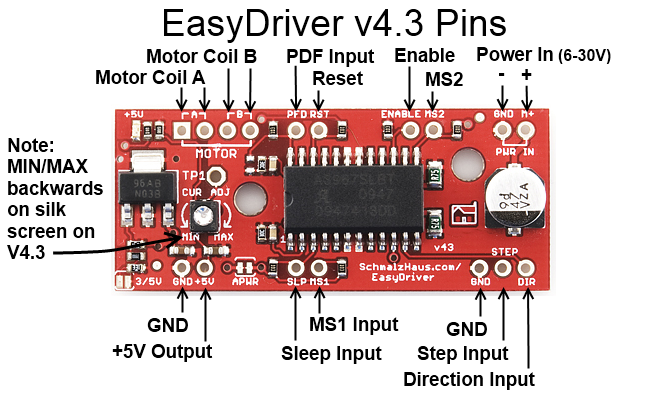
1. **EASY DRIVER**

The EasyDriver is a simple to use stepper motor driver, compatible with anything that can output a digital 0 to 5V pulse (or 0 to 3.3V pulse if you solder SJ2 closed on the EasyDriver). The EasyDriver requires a 6V to 30V supply to power the motor and can power any voltage of stepper motor. The EasyDriver has an on board voltage regulator for the digital interface that can be set to 5V or 3.3V. Connect a 4-wire stepper motor and a microcontroller and you've got precision motor control! EasyDriver drives bi-polar motors, and motors wired as bi-polar. I.e. 4,6, or 8 wire stepper motors.

This EasyDriver V4.5 has been co-designed with [Brian Schmalz](http://www.schmalzhaus.com/). It provides much more flexibility and control over your stepper motor, when compared to older versions. The microstep select (MS1 and MS2) pins of the A3967 are broken out allowing adjustments to the microstepping resolution. The sleep and enable pins are also broken out for further control.

**Note:** Do not connect or disconnect a motor while the driver is energized. This will cause permanent damage to the A3967 IC.

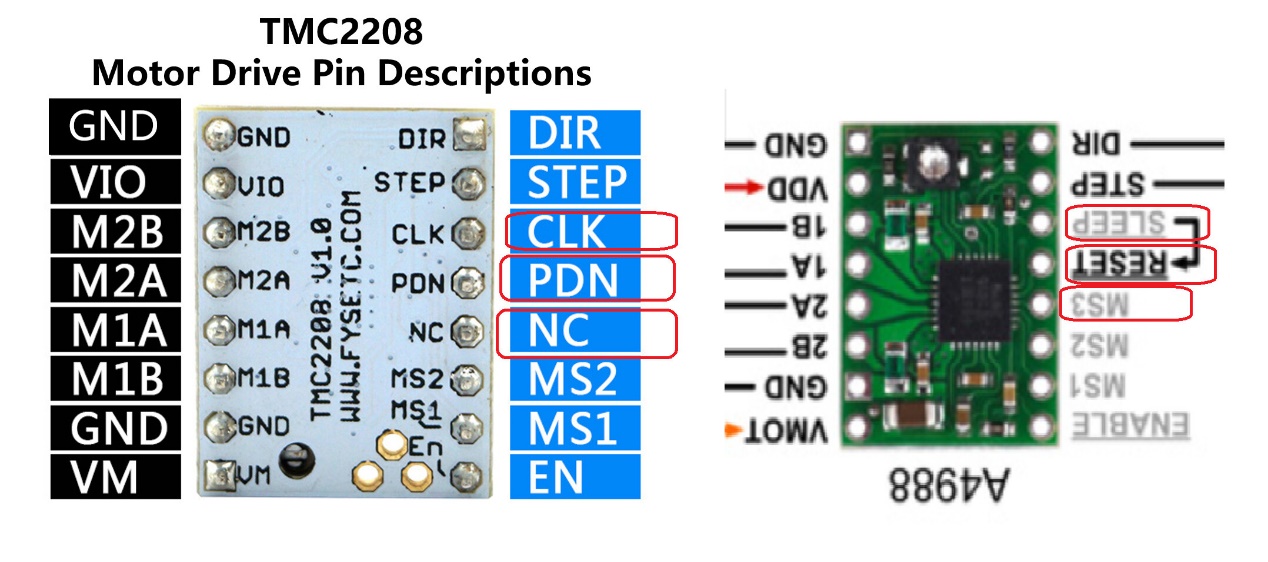
**Note:** This product is a collaboration with Brian Schmalz. A portion of each sales goes back to them for product support and continued development.



1. **TMC2208 DRIVER**

The TMC2208 v3 from BIGTREETECH is one of the most powerful 3d printer stepper drivers. It is mostly used in 3D printers’ driver motors with continuous drive current 1.2A, peak current 2A, voltage range 4.75V-36V. The TMC2208 stepper driver supports 256 micro steps (very smooth stepper motor motion) and also supports stealthChop mode of operation. StealthChop enable completely silences stepper motors by eliminating the noise caused by unsynchronized motor coil chopper operation, PWM jitter, and regulation noise. In addition, The TMC2208 has much better thermal dissipation and cooling performance which means more heavy duty and longer life of the driver.

The TMC2208 can be used as an upgrade of stepper driver such as A4988 or DRV8825 on Ramps style board or MKS GEN L board (this is called legacy mode). It can be used also in standalone mode or in UART mode where the stepper motor setting and tuning can be done through simple serial UART interface with your micro-controller. This make the TMC2208 driver flexible to use with any 3d printing board or technology without the need to redesign or make software or hardware changes.

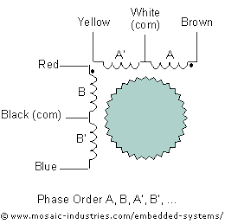


****After testing on choosing the appropriate driver, the EasyDriver and DR8825 were adopted as the best result, but the EasyDriver was excluded because of the high temperature emitted from it, but it does not affect the system’s work and stability. The TMC2208 was excluded because of its very high price compared to the DR8825.

**STEPPER MOTOR NEMA17**

**1.8 STEP**

A pair of stepper motors were used in the project ,The start and end ends of the motor windings were determined using the LED or using the data sheet.



Raspberry pi3 Model B +

Raspberry Pi brand

Compatible devices PC

The memory clock speed is 1.4 GHz

Linux website

Series RPI 3 Model B+

The maximum amount of RAM is 1 GB

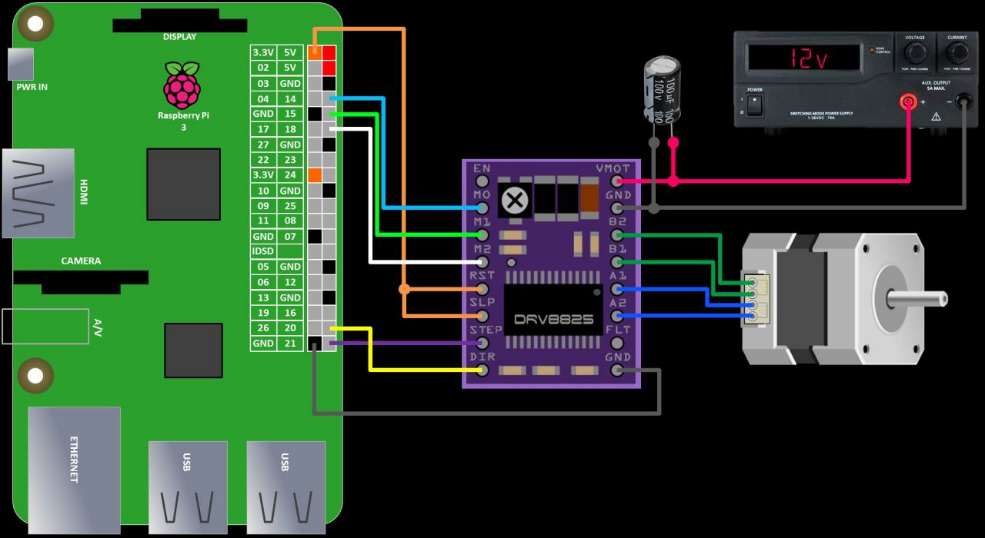
Integrated graphics card interface

Number of ports 4

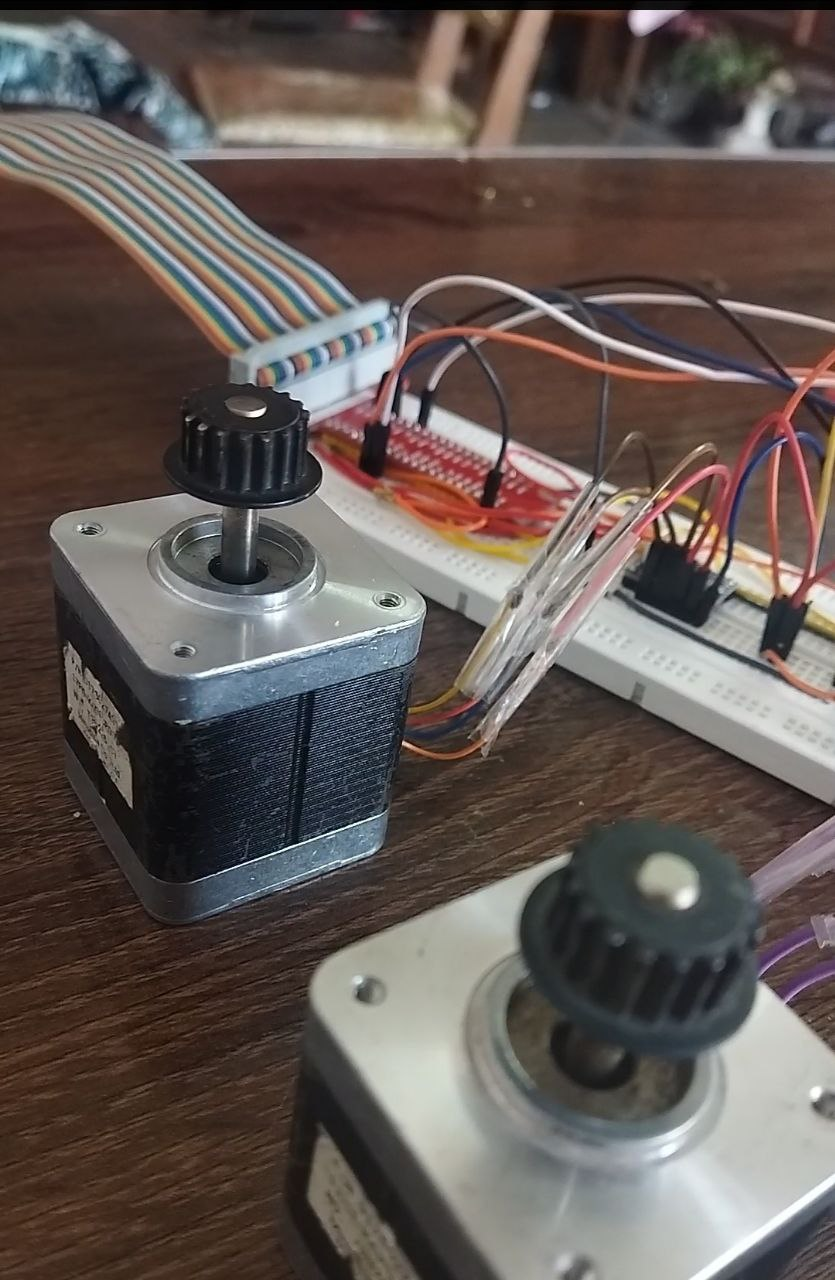
Wireless type 802.11bgn, 802.11ac

Number of USB 2.0 ports

**#TEST1**



<https://www.rototron.info/raspberry-pi-stepper-motor-tutorial>



**Generate pulse arrays from G code and drive stepping motors with Raspberry Pi**

<https://ymt-lab.com/en/post/2021/move-stepper-with-gcode-from-raspi/>

Convert image to SVG by website

<https://svgconvert.com>

Convert image to SVG by python

<https://pypi.org/project/pypotrace/> 1-

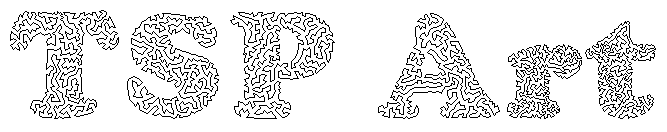
2- <https://github.com/ianmackinnon/png2svg>

After working on the Python language for a while...we had a problem in our office that we mainly depend on in the way of drawing and analyzing the pixels of the image. This library has not been updated since 2011. It needs an older version to work on. I searched for another programming language to solve this problem, which is the JavaScript language. Because it is the language of first-class websites and because it is very excellent in drawing and effects.

**Drawing using the JavaScript language and TSP Art**



**Travelling Salesman Problem (TSP) Art in Python**



The intention of this repo is to provide a beginner-programmer-friendly way to enable people to make their own TSP Art using Python, where you can put in any image and generate a version made purely out of dots, and then another version that connects all these dots in a single continuous line where the end points meet each other.

**Outline of Algorithm**

There are two major steps to the algorithm:

1. Stippling (or 'pointillism') - the image is represented by small black dots of identical size in a way such that darker areas have more dots clustered closely together than lighter areas. The algorithm used here is called 'weighted Voronoi stippling'.
2. Determining and drawing the Travelling Salesman Problem Path - the [Travelling Salesman Problem](https://simple.wikipedia.org/wiki/Travelling_salesman_problem) is a classic mathematical optimization problem where given a list of locations, we are to find a single path that travels through all the locations only once and returns to the starting point. Here we use the dots drawn in the first step as our locations and use an algorithm to determine and then draw an appropriate path.

* [Python 3](https://www.python.org/downloads/) - you should also know how to use the console/command prompt, and run/execute a Python script. Note that command line options might be different for those using Anaconda.
* Optional: [Concorde TSP Solver](http://www.math.uwaterloo.ca/tsp/concorde/index.html)
* Optional: [Git](https://git-scm.com/)

Optional: Image editing program. Free/open-source ones: [Krita](https://krita.org/en/), [GIMP](https://www.gimp.org/), And lastly, the image(s) that you want to convert!

**Python Dependencies**

* Pillow (error )
* ortools (Note that this requires [Microsoft Visual C++ Redistributable for Visual Studio 2019, which can be found at the bottom of this link.](https://visualstudio.microsoft.com/downloads/?q=Visual+C%2B%2B+Redistributable+for+Visual+Studio))
* tqdm
* imageio
* scipy
* matplotlib

**What kind of images should I use for best results?**

**Format:** This will work for the common image formats (.jpg, .png). More obscure image formats might have some issues, so I'd recommend converting them to .jpg or .png first.

**Type**: Generally you'll want to use images that is a single object against a white background. Colour doesn't matter as much since the image is converted to grayscale as part of the stippling process.

Three images are provided for you in the images folder for you to practice on and to observe results. The scripts are initially configured to use smiley face-inverted.png in the images folder, and you can get an idea of what the output might look like if you check the example-output folder.

**1. Image Preprocessing (and potential problems!)**

Skip this step if you're a first timer. This step will only be relevant after you've run through a few images and want to tweak things a little, or have run into certain problems.

**Images with transparent backgrounds**: You'll want to colour these backgrounds white in your image-editing program, as some transparent backgrounds are set to black by default.

**Difficult to distinguish sections of similar colour/shade**: You may have seen this when trying out croissant-emoji.png. There's two ways to deal with this: one is to increase the number of dots available. The other is to increase the contrast or recolour sections appropriately using your image editing program.

**Compression noise/grain/artefacts**: Sometimes your initial image might not necessarily be smooth, or you'll see 'bits that aren't supposed to be there'. These vary a huge amount so there's no one surefire method for dealing with all of them. I know there are built-in methods in GIMP/Krita/Photoshop for dealing with them, but am no expert - usually the examples I work with are simple enough to manually clean using Brush and Fill tools.

**2. Stippling**

Open up stippling.py in the editor of your choice, and change the ORIGINAL\_IMAGE variable to the folder and image that you wish to stipple. So if our image is figure.png located in the images folder, you'd rename the variable to "images/figure.png"

What should happen on your first time:

* the console should show something similar to a progress bar, showing each iteration on a new line
* a window will pop up and show the dots arranging themselves
* closing aforementioned window will finish the script, and you should see two new files in the images folder:
  + figure-1024-stipple.png which is a stippled version of your original image, and
  + figure-1024-stipple.tsp which is a record of the coordinates of each of the points. This is the file we need for the next step.

Note: 1024 refers to the number of dots used, assuming you use the initial settings as given. If you change this number, the resulting filenames will also have their numbers changed. This is to make it easier to experiment with different numbers of dots without constantly having to delete the old files.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3. Acquiring & Drawing the TSP Solution**

**Using OR-Tools in Python**

Open draw-tsp-path.py in your editor, and change the variables as follows:

* ORIGINAL\_IMAGE should be the stippled image you obtained for Step 2: images/figure-1024-stipple.png
* IMAGE\_TSP should refer to the stipple .tsp file that is generated after Step 2: images/figure-1024-stipple.tsp

Run the file, wait for Python to do its job and when it's done, the final image will be generated as images/figure-1024-tsp.png.

**Using Concorde (Windows GUI)**

Open Concorde either by double clicking on figure-1024-stipple.tsp or opening the program separately and then loading figure-1024-stipple.tsp file into it. Concorde should then display a series of dots that should resemble what you see in figure-1024-stipple.png.

In the menu, click on 'Heuristics', select 'Lin Kernighan', then click OK. Concorde will then generate a tour that goes through all the points and returns to the starting point.

Save the tour as a file by selecting in the menu: File > Save Tour. In our example we'll save it as figure-tour.cyc.

Open draw-tsp-path-concorde.py in your editor and change the filenames at the top of the file

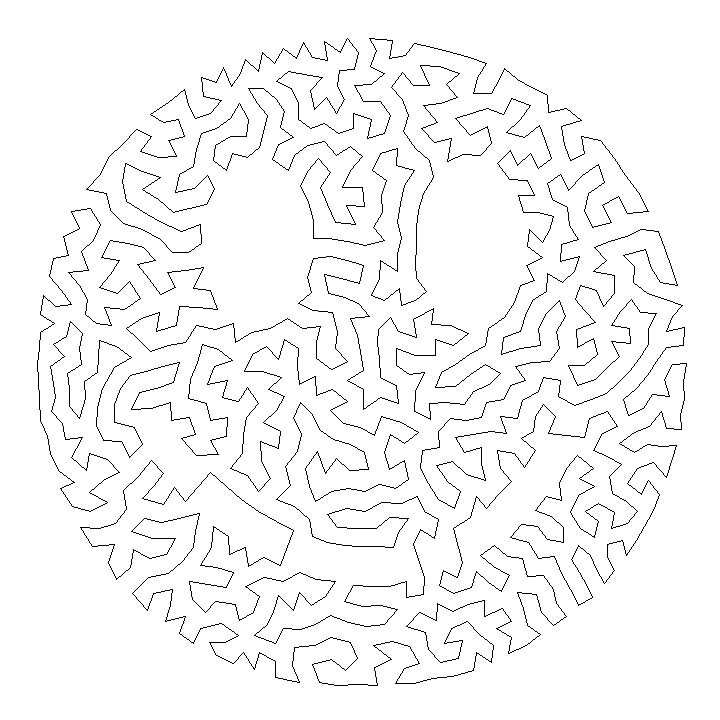
* ORIGINAL\_IMAGE should be the same initial image you used for Step 2: images/figure-1024-stipple.png
* IMAGE\_TSP should refer to the stipple .tsp file that is generated after Step 2: images/figure-1024-stipple.tsp
* IMAGE\_CYC should refer to the .cyc file that is generated from Concorde: images/figure-tour.cyc

Run the file and the script should generate the final image at images/figure-1024-tsp.png.

**When do I use Concorde over OR-Tools?**

Generally speaking you'll only want to use Concorde over OR-Tools if you have an image that has 'gaps' where you don't want a path to cross. Sometimes the OR-Tools algorithm may result in paths that 'cross-over' areas where you don't want them to, whereas the Concorde solver is less likely to achieve such a result.

To demonstrate as an example, in the example-output folder we have a smileyface-inverted.png:



Robert (Bob) Bosch ([Website](http://www.dominoartwork.com/), [Twitter](https://twitter.com/baabbaash/)) for the [original TSP art idea](http://www2.oberlin.edu/math/faculty/bosch/tspart-page.html). He's also written a book [Opt Art](https://www.amazon.com/Opt-Art-Mathematical-Optimization-Visual/dp/0691164061) which I highly recommend!

Adrian Secord ([Twitter](https://twitter.com/ajsecord)) for the weighted voronoi stippling algorithm. Nicholas Rougier ([Website](https://www.labri.fr/perso/nrougier/), [Twitter](https://twitter.com/NPRougier), [Github](https://github.com/rougier)) for the Python implementation.

See the Collection of Reference Links below for what I've found in my research while putting together this project.

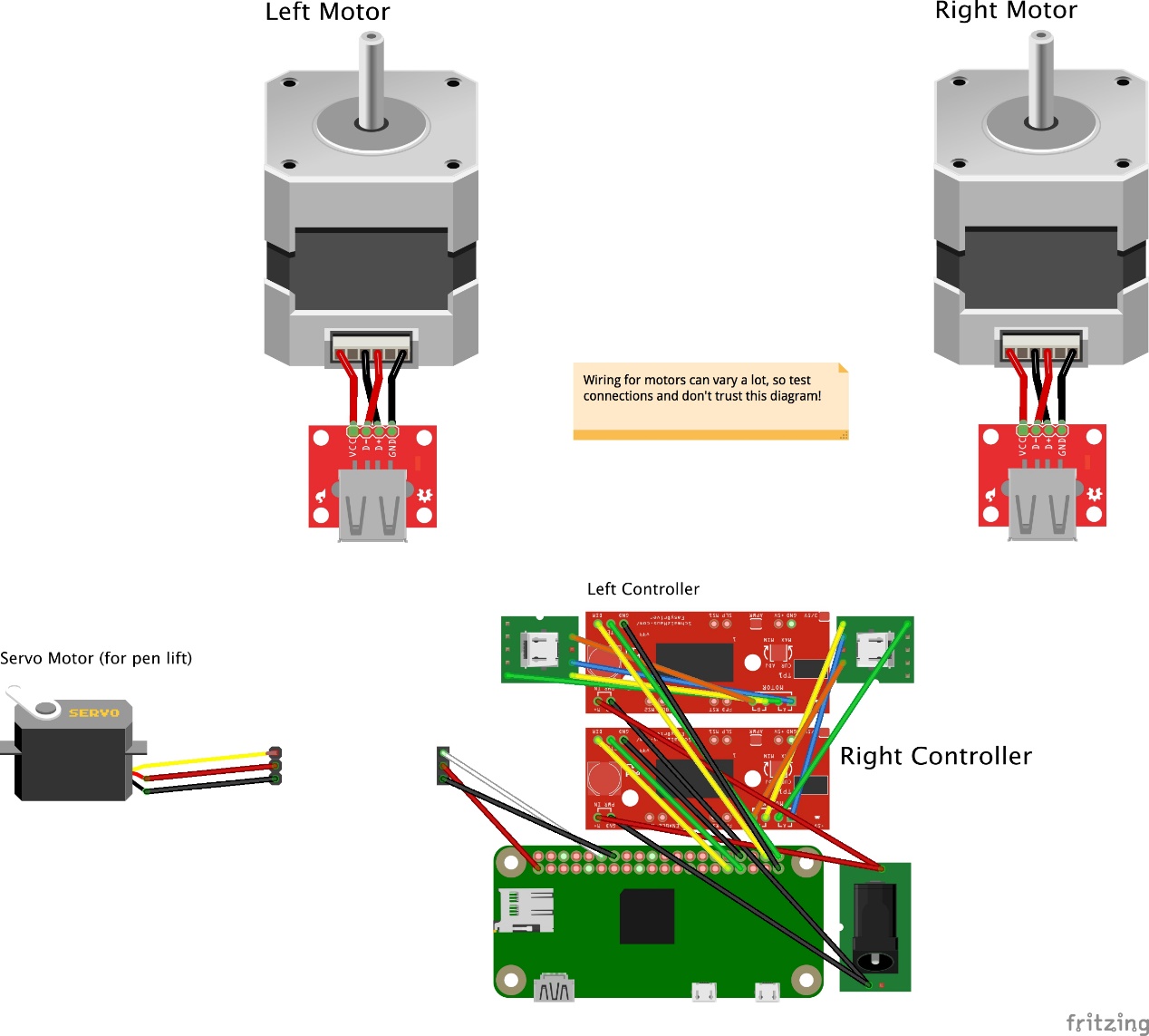
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

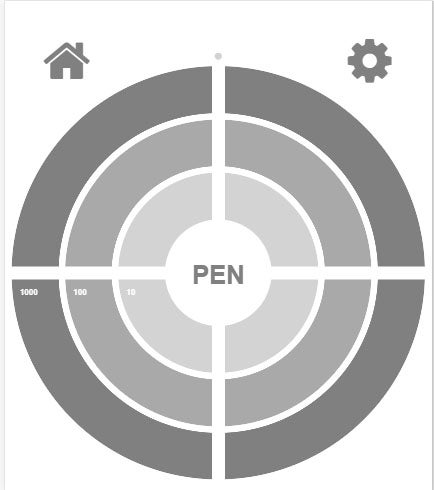
* Pillow (error )

**Python Imaging Library** is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source_software) additional [library](https://en.wikipedia.org/wiki/Library_(computing)) for the [Python programming language](https://en.wikipedia.org/wiki/Python_(programming_language)) that adds support for opening, [manipulating](https://en.wikipedia.org/wiki/Image_editing), and saving many different [image file formats](https://en.wikipedia.org/wiki/Image_file_formats). It is available for [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows), Mac OS X and [Linux](https://en.wikipedia.org/wiki/Linux).

The latest version of PIL is 1.1.7, was released in September 2009 and supports Python 1.5.2–2.7.

Development of the original project, known as **PIL**, was discontinued in 2011. Subsequently, a successor project named **Pillow** [forked](https://en.wikipedia.org/wiki/Fork_(software_development)) the PIL repository and added Python 3.x support. This fork has been adopted as a replacement for the original PIL in [Linux distributions](https://en.wikipedia.org/wiki/Linux_distribution) including [Debian](https://en.wikipedia.org/wiki/Debian_GNU/Linux) and [Ubuntu](https://en.wikipedia.org/wiki/Ubuntu)



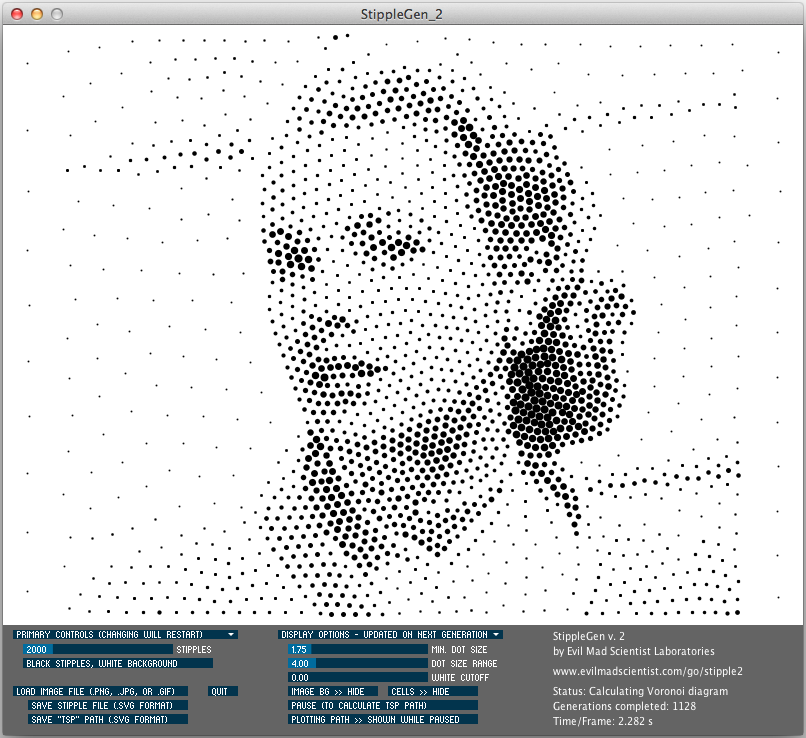


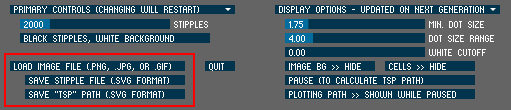
**Convert photo to SVG by Java Program**

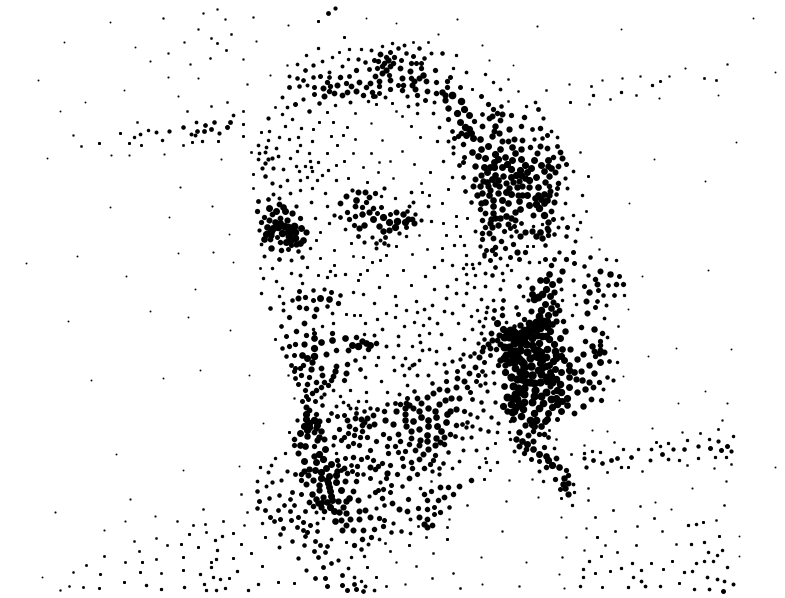
**StippleGen**

StippleGen is a free, open source, and cross-platform application from Evil Mad Scientist Laboratories that can create stipple drawings and “TSP art,” from image files.

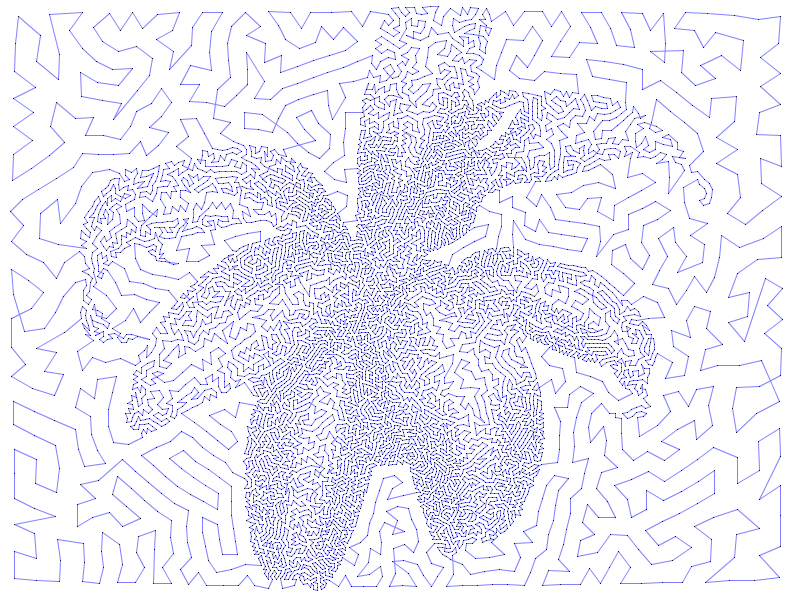
* Press the "Load Image" button to load your image file, usually JPG or PNG format.
* Adjust the "Stipples" slider for the dot density that you want-- more stipples takes longer to run.
* Allow it to run until the stipple distribution stops changing much, and adjust the display options (dot size and so forth) to your taste.
* Press the Pause button to calculate the TSP path between the stipples.
* Press one of the "Save" buttons to save the stipple locations or the path between them to an SVG file.







To load a new image file, click the "Load Image File" button that is located above the "save" buttons. You can open any image file in .png, .jpg, .tga, or (non-animated) .gif formats. The file name must end in one of the following: .png, .jpg, .tga, .gif, .PNG, .JPG, .TGA, or .GIF.

**test**

<https://wiki.evilmadscientist.com/StippleGen>

**Design and project experience**

