This is the master notebook for the Drawing Machine project.

Enter the desired height of the rendered image below, as well as the starting x and y coordinates. The width is automatically calculated from the image aspect ration, and the height. The x and y values don't matter for the cartesian plotter, but are important for the triangular hanging plotter.

In[73]:=
```
height = 30; (*centimeters*)
width = ImageDimensions[a][[1]] * height / ImageDimensions[a][[2]] ;
xbegin = 40;
ybegin = 58;
```

Below are hatching functions, they describe how the pixels will be shaded by the pen. Only one is used at a time, but more can be devised if different shading styles are so desired. In the case of the equitable project, NewerHatch2 was used. It renders the image with one long line that very rarely crosses itself.

In[77]:=
```
Hatch[x_, y_, level_] :=
  Table[
   {Table[i, {i, x, x + (width / ImageDimensions[a][[1]]),
        (width / ImageDimensions[a][[1]]) / level}][[i]], Flatten[
      Table[{y - (height / ImageDimensions[a][[2]]), y}, {i, 1, level, 1}]][[i]]},
   {i, 1, level}];

NewHatch[x_, y_, level_] :=
  Riffle[
   Table[{i, y},
    {i, x, x + (width / ImageDimensions[a][[1]]),
     (width / ImageDimensions[a][[1]]) / level}],
   Table[{i + (width / ImageDimensions[a][[1]]) / (2 * level),
     y - (height / ImageDimensions[a][[2]])},
    {i, x, x + (width / ImageDimensions[a][[1]]) - (width / ImageDimensions[a][[1]]) /
       (2 * level), (width / ImageDimensions[a][[1]]) / level}]
  ];

NewerHatch[x_, y_, level_] :=
  Table[{i, If[Mod[Round[(i - x) * level / (width / ImageDimensions[a][[1]])], 2] == 0,
     y, y - (height / ImageDimensions[a][[2]])]},
   {i, x, x + (width / ImageDimensions[a][[1]]),
    (width / ImageDimensions[a][[1]]) / level}];

NewerHatch2[x_, y_, level_] :=
  If[level == 0, {}, (* white space is now actually white *)
   Table[{i, If[Mod[Round[(i - x) * level / (width / ImageDimensions[a][[1]])], 2] == 0,
      y, y - (height / ImageDimensions[a][[2]])]},
    {i, x, x + (width / ImageDimensions[a][[1]]),
     (width / ImageDimensions[a][[1]]) / level}]];

Scribble[x_, y_, level_] :=
  Table[{RandomReal[{x, x + 1.2 * (width / ImageDimensions[a][[1]])}, level][[i]],
    RandomReal[{y - 1.2 * (height / ImageDimensions[a][[2]]), y}, level][[i]]}, {i,
    1, level}];
```

In this setting, the grayscale image data is normalized to the set [0,1], To get a good grayscale with the pen on paper, we need to define some functions to transform the image data. These functions were found by trial and error.

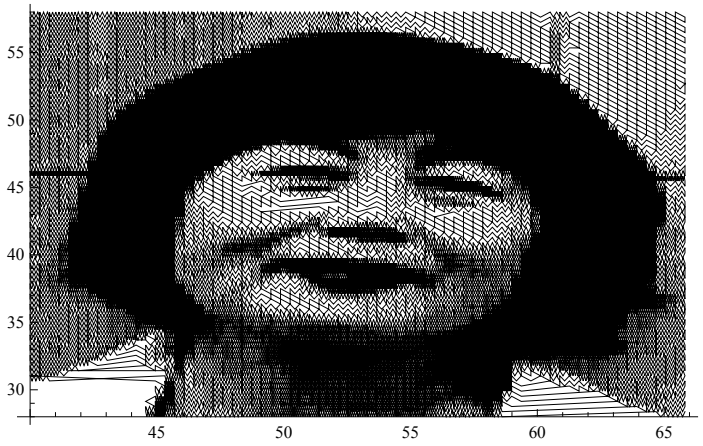
In[82]:=
```
GrayConvert[x_] := Round[60 (x^3)];
HatchConvert[x_] := Round[30 (x)^2]; (*useed to be x +0.2*)
```

Once the above inputs are loaded and functions defined, the rendering computation and visualization is executed. The first image is to scale, and the second is the data plotted, where the units for the axes are centimeters. Finally, a small sample of the positional data is displayed, in the form of (x, y) coordinates. The coordinates represent the pen tip positions on the paper.

In[84]:=
```
coordinates = Table[{
     {xbegin + i * width / ImageDimensions[a][[1]] // N,
      ybegin - j * height / ImageDimensions[a][[2]] // N},
     1 - ImageData[a][[j + 1]][[i + 1]]},
    {j, 0, ImageDimensions[a][[2]] - 1},
    {i, 0, ImageDimensions[a][[1]] - 1}
   ];

data = Flatten[Table[
    If[Mod[i, 2] == 1, coordinates[[i]], Reverse[coordinates[[i]]]],
    {i, 1, Length[coordinates]}],
   1];

linedata = Flatten[
   Table[
    If[Mod[Quotient[i - 1, ImageDimensions[a][[1]]], 2] == 0, (*Why is 75 here? it
      should be imagewidth (i.e ImageDimensions[a][[1]]), i think*)
     NewerHatch2[data[[i]][[1]][[1]], data[[i]][[1]][[2]],
      HatchConvert[data[[i]][[2]]]],
     Reverse[
      NewerHatch2[data[[i]][[1]][[1]],
       data[[i]][[1]][[2]], HatchConvert[data[[i]][[2]]]]
     ]
    ],
    {i, 1, Length[data]}
   ],
   1];

linedata = Prepend[linedata, {xbegin, ybegin}];

Graphics[
 Line[
  linedata]]

ListLinePlot[linedata, PlotStyle → Black]

linedata
```

Out[88]=

Out[89]=



Out[90]=

A very large output was generated. Here is a sample of it:

```
{{40, 58}, {40., 58.}, {40.0759, 57.6203}, {40.1519, 58.},
 {40.2278, 57.6203}, {40.3038, 58.}, {40.3797, 57.6203}, {40.3797, 58.},
 <<71 232>>, {65.3481, 28.}, {65.443, 28.3797}, {65.443, 28.3797},
 {65.538, 28.}, {65.6329, 28.3797}, {65.7278, 28.}, {65.8228, 28.3797}}
```

Show Less | Show More | Show Full Output | Set Size Limit...

This computation converts the list of (x,y) coordinates to individual stepper motor instructions. For example, if the first two (x, y) coordinates are (1, 1) and (2, 2) and it requires 1620 steps to move a centimeter in the x direction, and 1261 steps to move a centimeter in the y direction, then the instruction will be d0162001261, which breaks down to “d” “0” “1620” “0” “1261” (arbitrary control character, +x direction (a “1” would be -x direction), 1620 steps, +y direction (a “1” would be -y direction), 1261 steps, respectively). Since some distances take more that 9999 steps to tranverse, special cases had to be integrated.

```
stepsperx = 1650;
(* The number of stepper motor stepps per centimeter in the x-axis *)
stepspery = 1261.15;
(* The number of stepper motor stepps per centimeter in the y-axis *)
len = Length[linedata] - 1;
machinetest = Table[

   Join[

    {If[
       linedata[[i]][[1]] ≤ linedata[[i + 1]][[1]],
       "d0" <> IntegerString[Mod[Round[
            (linedata[[i + 1]][[1]] - linedata[[i]][[1]]) * stepsperx], 9999], 10, 4],
       "d1" <> IntegerString[Mod[Round[(linedata[[i]][[1]] -
               linedata[[i + 1]][[1]]) * stepsperx], 9999], 10, 4]
      ]
      <>
```

```
    If[
     linedata[[i]][[2]] ≤ linedata[[i + 1]][[2]],
     "0" <> IntegerString[Mod[Round[
          (linedata[[i + 1]][[2]] - linedata[[i]][[2]]) * stepspery], 9999], 10, 4],
     "1" <> IntegerString[Mod[Round[(linedata[[i]][[2]] - linedata[[i + 1]][[2]]) *
           stepspery], 9999], 10, 4]
    ]}

  ,

  Table[
   If[
    linedata[[i]][[1]] ≤ linedata[[i + 1]][[1]],
    "d0999900000",
    "d1999900000"
   ],
   {j, 1, Quotient[Round[
      Abs[(linedata[[i]][[1]] - linedata[[i + 1]][[1]]) * stepsperx]], 9999], 1}]

  ,

  Table[
   If[
    linedata[[i]][[2]] ≤ linedata[[i + 1]][[2]],
    "d0000009999",
    "d0000019999"
   ],
   {k, 1, Quotient[Round[
      Abs[(linedata[[i]][[2]] - linedata[[i + 1]][[2]]) * stepspery]], 9999], 1}]

 ]

 , {i, 1, len, 1}] // Flatten
```

A very large output was generated. Here is a sample of it:

```
{d0000000000, d0008310445, d0008300445, d0008310445, d0008300445, d0008310445,
 d0008300445, d0008310445, d0000000445, d0005810445, d0005800445, d0005810445,
 d0005800445, d0005810445, d0005800445, d0005810445, d0005800445, d0005810445,
 d0005800445, d0000000000, d0003910445, d0003900445, d0003910445,
 d0003900445, <<63 517>>, d0005800445, d0005810445, d0005800445, d0005810445,
 d0005800445, d0000000000, d0006510445, d0006500445, d0006510445,
 d0006500445, d0006510445, d0006500445, d0006510445, d0006500445,
 d0006510445, d0000000445, d0007310445, d0007300445, d0007310445,
 d0007300445, d0007310445, d0007300445, d0007310445, d0007300445}
```

Show Less | Show More | Show Full Output | Set Size Limit...

We then write the list of stepper motor instructions to a file on the PC. Unfortunately, this method adds quotation marks around the strings when writing the file. Those quotations were removed with

the follwoing Bash command:

```
tr -d '"' <input.txt >output.txt
```

```
str = OpenWrite[];
For[i = 1, i ≤ Length[machinetest], i++,
  Write[str, machinetest[[i]]]];
Close[str]
```

```
C:\Users\DavidP\AppData\Local\Temp\m-21e799de-374f-45c3-9330-2ca3c8e8570d
```

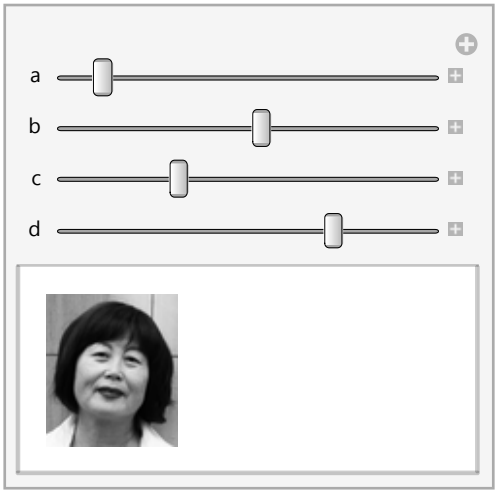
We must load the image we want the machine to draw, and then do some image manipulation to get it ready for rendering. The image is converted to grayscale, cropped and resized so that each pixel is a square with a side of length 2-3 cm.

In[1]:=
```
ev = Import["C:\\Users\\DavidP\\Downloads\\l1.jpg"]
```

Out[1]=

In[4]:=
```
Manipulate[
 evv = ImageResize[ImageTake[ColorConvert[ev, "Grayscale"], {a, b}, {c, d}], 68],
 {a, 1, ImageDimensions[ev][[2]], 1},
 {b, 1, ImageDimensions[ev][[2]], 1},
 {c, 1, ImageDimensions[ev][[1]], 1},
 {d, 1, ImageDimensions[ev][[1]], 1}
]
ImageDimensions[evv]
```

Out[4]=



Out[5]=
```
{68, 74}
```

In[7]:=
```
a = evv
```

Out[7]=

```
evv = ImageResize[ColorConvert[ev, "Grayscale"], 90]
```

Since the machine can perferm arbitrary sequential movements along the x and y axes, line drawings are also possible. The computation depends first on a binary image with the line in white (represented in data as 0) and background in black (represented as 1).

In[9]:= **EV =**

Out[9]=


```
ImageData[EV][[1]][[1]] (* Top left pixel *)
```

0

Plot the points represented by white pixels in the above image and resize to fit the desired drawing size in cm

In[10]:=
```
points = ImageValuePositions[EV, 1];
resize[{x_, y_}] = {16 / 521 * x, 16 / 521 * y};
pointscm = resize /@ points;
ListPlot[pointscm]
```

Out[13]=

Below the built in function FindCurvePath attempts to organize the points into sets of paths, where each path is a set of points ordered sequentially along a single path in euclidean space. The function is computentionally intensive and can take minutes or hours for high definition images.

In[14]:=
```
curves = FindCurvePath[pointscm];
segments = Table[pointscm[[curves[[i]]]], {i, 1, Length[curves], 1}];
ListLinePlot[segments, AspectRatio → Automatic]
```

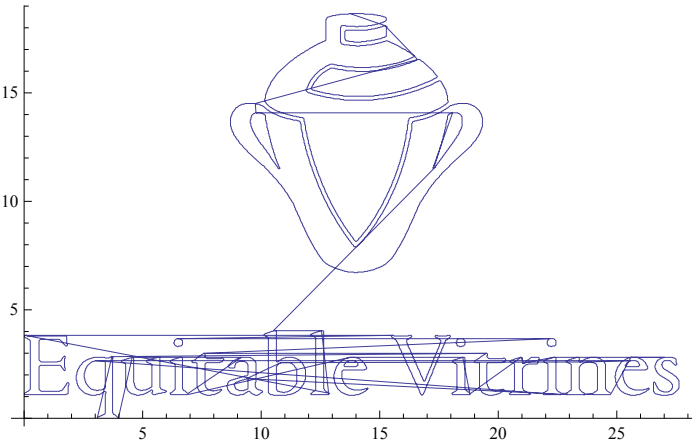
Out[16]=

Unfortunately, The drawing machine can't always lift the pen off the paper, and although the poitns

within the paths are ordered, the paths themselves are not, and that can cause and undesirable scribble effect, as shown below.

In[18]:=
```
ListLinePlot[Flatten[segments, 1]]
```

Out[18]=



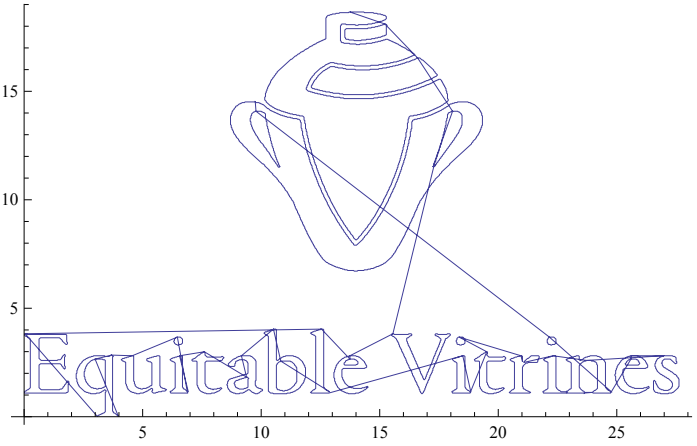
So what do we do? We re-order the paths. How? By choosing an initial path and then finding the path who's first point is closest to the last point of the initial path. Then, from the remaining paths, we choose the path who's first point is closest to the final point of the second path, and so forth. The algorithm is not optimized for performance, as using sort is gross overkill, but it works and was pretty simple to put together.

In[19]:=
```
orderedsegments = Append[{}, segments[[1]]];
While[Length[orderedsegments] ≠ Length[segments],
 orderedsegments = Append[
   orderedsegments,
   SortBy[
     Complement[segments, orderedsegments],
     EuclideanDistance[
       Last[orderedsegments[[Length[orderedsegments]]]],
       First[#]] &
    ][[1]]
  ]
]
```

Below you can see the improvement.

In[45]:=
```
finalorder = Flatten[orderedsegments, 1];
linedata = finalorder;
ListLinePlot[finalorder]
linedata[[1]]
```

Out[47]=



Out[48]= {13.7428, 18.6564}

Now that we've optimized line drawing that do not lift the pen off the paper, let's do the computations to insert servo motor commands to lift the pen off the page when moving from the end of one path to the beginning of the next. Specifically, lets compute the list indices of the path edges. Furthe down the page is the computation for inserting the servo commands into the list.

In[25]:=
```
orderedlengths = Length /@ orderedsegments;
orderedsums =
 Table[Total[Take[orderedlengths, i]] + 2 (i - 1), {i, 1, Length[orderedlengths]}]
```

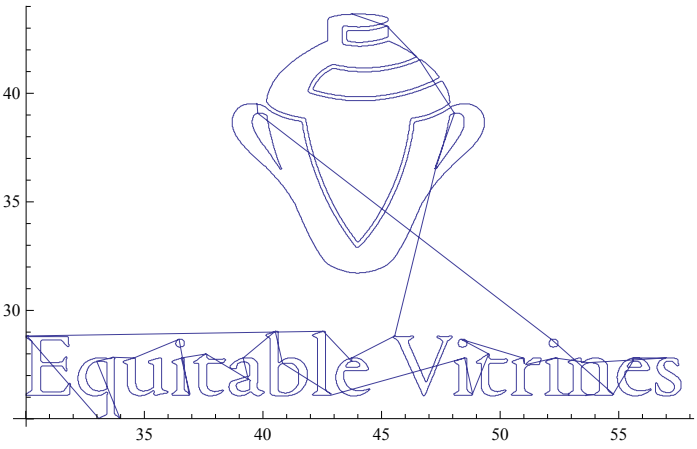
Out[26]=
```
{1256, 1400, 1760, 1936, 1946, 1950, 2321, 2534, 2581, 2682, 2802, 3073, 3301, 3361,
 3577, 3639, 3730, 3767, 4072, 4168, 4289, 4449, 4933, 5068, 5129, 5223, 5441, 5480,
 5493, 5686, 5779, 5806, 5834, 5881, 6095, 6313, 6429, 6651, 6690, 6873, 7980}
```

```
outputlist = Append[outputlist , "c"];
For[i = 1, i ≤ Length[orderedsums], i++,
  outputlist = Insert[outputlist, "c", orderedsums[[i]]];
  outputlist = Insert[outputlist, "m", orderedsums[[i]] + 2];
 ];
(*outputlist = outputlist//Flatten*)
```

For large line drawings on Drawing Machine #1 offset the axes

In[52]:=
```
offset[{x_, y_}] := {x + 30, y + 25};
linedata = offset /@ finalorder;
ListLinePlot[linedata]
```

Out[54]=



Have (x,y) coordinates is great, but to make a drawing with Drawing Machine #1, we need to convert the (x,y) coordinates to pen holder support lengths using the following formula.

In[55]:=
```
LandR[x_, y_, w_, h_] := {Sqrt[x^2 + (h - y)^2], Sqrt[(h - y)^2 + (w - x)^2]};
L = Table[LandR[linedata[[i]][[1]], linedata[[i]][[2]], 102, 89],
  {i, 1, Length[linedata]}]
L[[1]] //
 N
```

Out[56]=

A very large output was generated. Here is a sample of it:

```
{{63.0037, 73.8237}, {62.9824, 73.8479}, {62.9832, 73.891},
 {62.9619, 73.9153}, {62.9407, 73.9395}, {62.9194, 73.9638},
 <<7888>>, {63.3808, 79.6451}, {63.4, 79.6211}, {63.4192, 79.597},
 {63.4384, 79.573}, {63.4576, 79.5489}, {63.4768, 79.5249}}
```

Show Less | Show More | Show Full Output | Set Size Limit...

Out[57]=
```
{63.0037, 73.8237}
```

This is the computation to convert the list of pen support lengths to instructions for the steppers motors. It looks the same as a code block above, but has some crucial differences. Can you spot them?

In[62]:=
```
len = Length[L] - 1;
stepsper = 3200;
radius = 1.32;
outputlist = Table[

   Join[

    {If[
       L[[i]][[1]] ≤ L[[i + 1]][[1]],
       "d1" <> IntegerString[Mod[Round[(L[[i + 1]][[1]] - L[[i]][[1]]) *
             stepsper / (2 * Pi * radius)], 9999], 10, 4],
       "d0" <> IntegerString[Mod[Round[(L[[i]][[1]] - L[[i + 1]][[1]]) *
             stepsper / (2 * Pi * radius)], 9999], 10, 4]
      ]
      <>
      If[
       L[[i]][[2]] ≤ L[[i + 1]][[2]],
       "0" <> IntegerString[Mod[Round[(L[[i + 1]][[2]] - L[[i]][[2]]) *
             stepsper / (2 * Pi * radius)], 9999], 10, 4],
       "1" <> IntegerString[Mod[Round[(L[[i]][[2]] - L[[i + 1]][[2]]) *
             stepsper / (2 * Pi * radius)], 9999], 10, 4]
      ]}

    ,

    Table[
     If[
      L[[i]][[1]] ≤ L[[i + 1]][[1]],
      "d1999900000",
      "d0999900000"
     ],
     {i, 1, Quotient[Round[
        Abs[(L[[i]][[1]] - L[[i + 1]][[1]]) * stepsper / (2 * Pi * radius)]], 9999], 1}]

    ,

    Table[
     If[
      L[[i]][[2]] ≤ L[[i + 1]][[2]],
      "d0000009999",
      "d1000019999"
     ],
     {i, 1, Quotient[Round[
        Abs[(L[[i]][[2]] - L[[i + 1]][[2]]) * stepsper / (2 * Pi * radius)]], 9999], 1}]

   ]

   , {i, 1, len, 1}] // Flatten
```

Here's the code for inserting the servo motor commands into the stepper motor instruction list generated above

In[66]:=
```
outputlist = Append[outputlist , "c"];
For[i = 1, i ≤ Length[orderedsums], i++,
  outputlist = Insert[outputlist, "c", orderedsums[[i]]];
  outputlist = Insert[outputlist, "m", orderedsums[[i]] + 2];
 ];
outputlist = outputlist // Flatten
```

And finally, write to file.

```
str = OpenWrite[];
For[i = 1, i ≤ Length[outputlist], i++,
Write[str, outputlist[[i]]]];
Close[str]
```