GraffitiBot

Margaret Meehan
Cornell University
G03 Gates Hall
mkm248@cornell.edu

Mike Merrill
Cornell University
G03 Gates Hall
mam546@cornell.edu

Ishaan Jhaveri
Cornell University
G03 Gates Hall
iaj8@cornell.eud

INTRODUCTION

GraffitiBot is a portable device that can be used to create art in public places. It consists of a drawing head, two wall-mounted stepper motors controlled via Pololu stepper drivers, and a Photon microcontroller. It is capable of drawing on virtually any smooth surface including blackboards, whiteboards, glass, paper, and even painted walls. GraffitiBot can also draw with nearly any implement. If the marker or pen does not fit within the drawing head, we have the ability to print different drawing head sizes. GraffitiBot can be used to inspire education, advertise a project, make public announcement, or create art. The project is useful because it promotes artistic and technological literacy while sparking a dialogue about the boundary between art and technology. Is it a tool or a creator? Is its work derivative or original? GraffitiBot revisits old questions about the role of machines in our lives.

GraffitiBot is capable of both reproducing original works and creating new public displays. Through publicly available algorithms virtually any vector image can be converted to GCODE. These instructions are then fed individually through a computer's serial port to the robot through a handshake protocol built in Processing. When the Photon is ready to receive a new command it sends a request character via the serial. In turn processing provides a new command and waits for the Photon to respond. Like all GCODE these instructions come in the form of circular arcs and straight lines. Linear instructions are left as is, while arcs are converted into a series of small lines. These lines are then converted to stepper instructions for both the left and right motor through an algorithm that translates the cartesian GCODE commands to our coordinate system. The steppers are then prompted to move to a pre-calculated number of steps to bring the drawing head to a new point on the plane. The drawing head is capable of lifting itself off of the plane using a servo-operated foot if the proper GCODE command is passed. If the foot is retracted the drawing head will mark a line as is moves. If the foot is touching the drawing surface the drawing head will not make a mark and will instead reposition itself. When the print is complete the drawing head returns to the origin.

GraffitiBot was designed to be educational, versatile, and portable. Our vision was to create a simple mechanism that could be quickly deployed anywhere. We plan to place GraffitiBot in public spaces where it can be a catalyst for conversations about art and technology. GraffitiBot can also be used as an educational tool. The mechanism is highly visible and relatively accessible. Young children may enjoy the engagement that comes from picking out an image from Google and watching it come to life on a large canvas, while older students can learn about trigonometry and basic control algorithms. GraffitiBot serves as an example of the complexity that can arise from simple mechanical designs and therefore provides a reachable platform for educational discussion.

RELATED WORK

Drawing machines are a common means of expression. Several previous developers have created systems that transfer vectorized images into the physical world by means of every-day drawing implements. In fact, previous work from this very class makes use of a simple Sharpie marker to draw on a flat surface [1]. Further work from the art community focuses on the role of mechanical structures as agents of creation. Artists such as Harvey Moon and Jürg Lehni use a combination of conventional writing tools and robotics to create pieces that are simultaneously familiar and foreign [2] [3]. These machines all share a common structure – stepper motors connected to a writing implement. We hope to emulate this design and artistic direction.

The work of Jürg Lehni also focuses on algorithms to transform vectorized line drawings to physical paths [4]. One challenge for our project is in taking an image from the internet and converting that into x/y instructions for our printer. While Lenhi's method relies on deprecated code, he points to other inventors who have developed similar projects. mDraw is a collection of open source hardware and software for creating drawing robots [5]. It includes software for converting images into x/y instructions for all manner of drawing things. Our project used a similar software, written by the team at Marginally Clever specifically to interpret G Code files into Stepper movement [6]. Following the advice from Yet Another Wall Plotter project, we used Inkscape to convert SVG files into GCode for our interpreter [7].

The physical design of our Grafftibot came largely from a project by Harvey Moon. The pulley system hanging off the wall with the pen that moves forward and backward

depending on whether the system is drawing or relocating was inspired by his design. Our 'pulley' system takes a different approach than his, and most vertical wall plotter's, in that there is actually no counter weight. Rather, our steppers wind or release string around an attached spool. Our drawing head is also a bit more designed than his, with a rotating laser cut disk attached to the servo rather than just a flimsy arm. This made for smoother motion. We also have various 3D printed drawing heads to firmly hold the drawing device in -- if we want to use a pen or an Expo marker. Some inspiration for drawing heads came from the Plotter Bot page [8].

An initial motivation for our project was *what* we actually choose to draw with the machine. One idea was to remove news stories from the invisible internet and transmit them to a public space where they can provoke dialogue and intrigue. This is still a potential direction. Recent works by political artists have attempted to draw attention to important causes by leveraging viral media. This summer, the anarchist art collective Indecline placed naked statues of Donald Trump in five cities across the county [9]. The display, called "The Emperor Has No Balls" gained widespread media attention in part due to the public nature of the piece. This type of work inspires us to use our machine in public space.

As demonstrated by the list above, the idea of a drawing robot is not in of itself original. What we hope to contribute is a device for quiet reflection inspired by the noisy digital world around us. While it is true that many of the components of our project are modified from pre-existing solutions, we believe that by combining these technologies to create a flexible, cloud-based, and socially conscious drawing robot has demonstrated an original and elegant contribution.

Figure 1. Graffiti Bot System with its rendition of The Thinker

DESIGN

GraffitiBot is comprised of several working units that come together as a comprehensive whole. The winch mechanism, drawing head, and software components must all operate in unison to create the final print.

Winch Mechanism

The winch mechanism is internally defined as the assembly created by the union of the stepper motors, their mounts, the spools attached to the steppers, and the the string that unravels from the winches and reaches the drawing head.

The stepper motors are the same SM-42BYG011 model that is included in our kits. We went with this model because we had some familiarity with it after using it in class. We also bought a twelve volt power supply, but eventually ended up going with the nine volt supplies from our kits. The steppers were driven with Poulou stepper breakouts. The steppers are connected to the spools with universal mounting hubs. The spools we decided to use were 3D printed and were iteratively designed to keep the string as close to the wall as possible, as it is necessary to keep a large angle between the wall and the string to ensure proper drawing head contact. The rope enters the spool through its central axis and is tied to a small bolt to prevent from slipping back through. In later versions a dab of hot glue was added to the bolt to dampen the noise generated by rattling during stepper movement.

The steppers are mounted to the wall with 3D printed holders. The holders have an elevated platform where the stepper rests and two upright posts which attach to the drawing surface with Command Strips. Command strips were chosen because they create a strong bond with the wall and can be easily removed without damaging the surface.

In initial designs we planned on using a thin wire rope on the spools. However, after ordering the wire rope and playing around with its dimensions we realized that it was so stiff that in order for it to wrap around the winch the spool would have to be so large that we would not be able to 3D print it. As a remedy we elected to use some thin and strong vinyl string that we found laying around.

All in all the winch mechanism did not change much as our design evolved. When we built our initial prototype we had an issue with excessive vibrations from the steppers. These vibrations produced jagged lines and made it difficult to draw with any high degree of accuracy. However, when we shifted from H-Bridge control to the Poulou drivers they were dramatically dampened. Besides this hiccup the design performed rather well. The mechanism was simple enough that it performed reliably and as planned. There were some

issues with the control algorithms we developed, but these are covered below in the section on our software.

We again encountered no major difficulties when it came time to integrate the winch system into the final design. We had some trouble with our wire connections falling apart, but we were able to fix this issue with liberal application of tape. In a future design it might be prudent to implement crimped connectors where the stepper wires meet the long leeds from the breadboard. One other issue is that occasionally the string can come off the spool and become tangled on the stepper rod. This could be fixed by using a hub with a larger rim to keep the string in line, but we considered both this modification and the crimped wires to be non-vital to the success of our prototype and therefore outside the scope of this design phase.

At the end of the day the winch mechanism performed as expected. It is able to accurately move the drawing head to nearly any point in the plane formed by the floor and the parallel line between the two steppers.

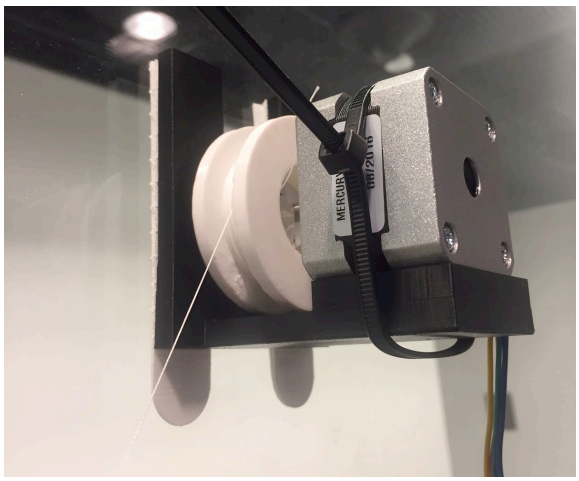

Figure 2. 3D printed stepper holder and spool

Drawing Head

The drawing head is the assembly that makes marks on the drawing surface. It is comprised of a writing utensil, a servo, and a foot that lifts the drawing head off of the drawing surface when the head is to move without making a mark.

In the initial prototype the drawing head was simply an Expo marker ziptied to the servo. The lifting food was a three-inch length of wooden dowel that we attached to the servo with wire. The strings from the winches were tied to the zipties.

This part of our first prototype was probably the worst. The string attached to the drawing head closely to the marker, causing it to become unstable and preventing the marker from making consistent ninety degree contact with the drawing surface. This problem only became worse when the lifting foot was engaged. The marker was easily thrown off center when it returned to the drawing surface, and often could not recover on its own. This design was easy to evaluate: the inability to make consistent contact made it impossible to draw with any degree of accuracy.

As we saw it there were two points of failure: the pen was unstable when it rested on the surface and the lifting mechanism was too unpredictable and jerky. To solve the first problem we thought it would be important to have the string attached to the head farther from the pen or marker. This meant building a dedicated unit to hold all of the piece together rather than relying on zip ties for everything.

Our first thought to improve the lifting mechanism was to lift just the marker off of the surface rather the whole head. We thought that it would be easier to maintain consistent head contact rather than lifting and removing it every time we wanted to move without marking the surface. This led us to develop our second generation drawing head. This head used castors to maintain wall contact and a servo mount to lift the pen when desired. However, this head actually performed worse as the castors dragged on smooth surfaces without turning and the servo could not consistently lower the marker to the correct height to allow drawing.

Following this setback we were back to the drawing board. We decided to keep the dedicated holder, but return to a lifting mechanism that lifted the whole assembly off the drawing surface. This led us to our third generation drawing head: a 3D printed assembly with a servo mount and a clip to connect to the Expo marker or pen. This version provided a solid base for all components and allowed the market to securely fit into the assembly. We also improved the lifting foot by changing it to a smooth curve instead of the rod that we used in the initial prototype. The new shape of the foot provided a gradually movement that was a large improvement on the jerky lift that had plagued our first prototype.

When it came time to finally integrate the drawing head we didn't run into any major roadblocks. The lifting mechanism performed as expected and the drawing head moved well on the wall. We did encounter some issues with keeping the servo connected, but all in all things turned well.

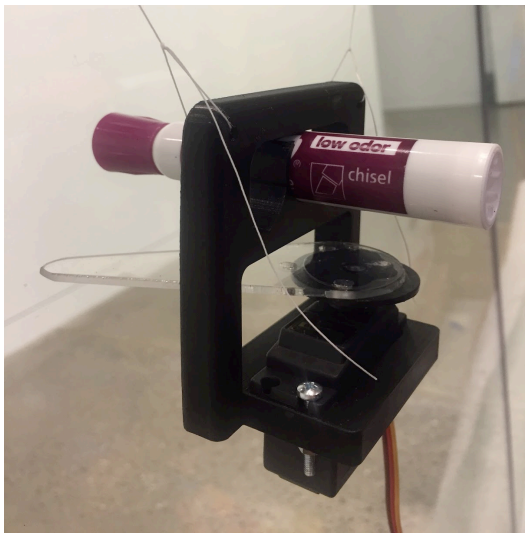

Figure 3. Current 3D printed drawing head

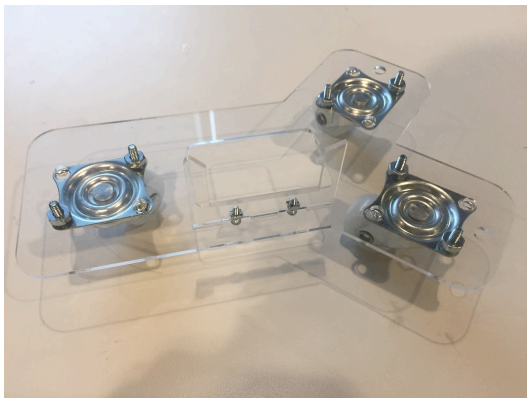

Figure 4. Discarded prototype of drawing head with castors

Software

I. Initial Software

After the hardware was in place we began with a very rudimentary software design that could just make straight lines and polygons. This was before we invested any serious time and effort into thinking about the mathematics involved in transitioning lines with x-y coordinates into motion that the steppers could emulate. At this stage we wanted to see how the lines would come out so we could decide how best to proceed from there.

As a result the initial software had the two steppers set up just as outputs from the photon which we would make high and low accordingly to move the stepper in one or the other direction. We didn't use any libraries. As for the pen, we moved it to and from the wall by turning a servo 90 degrees to 0 degrees. The servo is attached to a simple laser cut piece that either forces the pen to or away from the wall. This is explained in further detail in the hardware section above. The only interface with the outside world of this initial software was simple commands written to the Serial port like 'v' for vertical line and 'h' for horizontal line. A little later we added 'c1,2' for the head to travel to point 1,2 for example and 'l0,100' for the right winch to stay constant and the left winch to release 100 units of string, for example. This will be explained in more detail in the following section.

We had simple approximations to horizontal and vertical lines. If we always started the drawing head in the center (in the x-direction) between the left and right winch (the height, or y-direction doesn't matter) we reasoned that extending the two strings equally would make a vertical line downwards just as contracting the two strings equally would make a vertical line upwards. Similarly we reasoned that from this position, contracting the left string the same amount that we extend the right string would move the drawing head in a straight left line, and vice versa for a straight right line. We tested this out to see how straight our lines came out, and how our polygons looked. We learnt from this test that the lines are straight when they are small compared to the distance between the drawing head and either winch. Whenever that distance becomes too small, the lines become closer to arcs, and not straight. Thus we decided to make the limits of our drawing surface small compared to the distance between the winches. We never have the drawing head rise too much, or go too far right or left. Using these constraints, we could use x-y coordinate images, instead of converting our images to arc representations.

II. Development of Software

2.1 Libraries

2.1.1 Developments

The first area in which we brought changes to our initial software was in our use of libraries. We realized that if we used a more advanced library for our steppers, we could make finer steps, and could rely more on our steppers to work properly. So we replaced simple outputs with the Accel MultiStepper library. This library ensures that the two steppers move together, and ensures that even if they have different distances to move, they will adjust their speed so as to arrive at their destinations at the same time. This helps the lines be solid.

2.1.2 Bugs/Problems and Solutions

There weren't too many bugs getting this part to work. We had to modify the code in Accel's MultiStepper.h and MultiStepper.cpp files in order for them to work with the Photon but this was trivial since we already had an example of how to do this for AccelStepper.h and AccelStepper.cpp from the course materials, so we just followed this example for MultiStepper.

2.2 Third party software

2.2.1 Developments

In order to have our images in x-y coordinates, in other words in a form that we could transform into commands to our steppers we had to use the third party software InkScape, specifically the JTech photonics lasertool plugin. This program and plugin allow us to take an svg image and convert it to Gcode. The GCode essentially breaks the image into many tiny lines. This was perfect for our design because the smaller the lines, the more precise the drawing, given the wall-hanging and arc structure of our system. Finally, we n eeded an interpreter for the Gcode. We considered writing one ourselves but found one from Maginally Clever's github page, (https://github.com/MarginallyClever/GcodeCNCDemo) to do the job just fine. This code reads Gcode and converts it into commands to a line() function that just takes x1,x2,y1,y2. This was perfect for us since we knew how to convert x-y coordinates to motor motion. More on this in the next subsection on Mathematical Developments. It also reads when the Gcode is in "write" mode, and in "move" mode. These modes corresponded perfectly to our commands to the servo, on whether the pen should be down (write mode) or up (move mode).

2.2.2 Bugs/Problems and Solutions

Installing these softwares was easy, but there were some bugs in using them. With inkscape we had an issue where the Gcode commands it was outputting weren't always correct. At times it would reverse the direction of a curve, and do the complement of the curve. So for example if in the svg image there was a small curve in the anti-clockwise direction, (curves would be approximated to lines) it would instead make a much larger curve in the clockwise direction, drawing everything in the larger circle but the actual target curve itself. We fixed this by adding logic that, when it sees curves larger than a semi-circe, reverses the direction and makes the complement of that curve. Once we added this logic, we had no issues with inkscape. As for the Gcode interpreter, at first it wasn't working because part of what it does is parse a string that is one line of a Gcode program, but it was having issues parsing the lines properly, and would often loop infinitely, never terminating when it reached the end of a line. The reason for this is that the Gcode interpreter was written for C but we were using it with Arduino, so some conversions between types that works in C, wasn't working in Arduino. The solution was reimplementing the parse function from scratch, which didn't prove to be too difficult. This was the only issue with using Marginally Clever's Gcode interpreter.

2.3 Mathematical Developments

2.3.1 Developments

This was our biggest addition to the code. The ability to transform a line with simply four coordinates (x1,x2,y1,y2) into commands to the steppers. To do this, we had to do some mathematical conversions. We had to map the size of the line to the size of the contraction or expansion of each string's length. We found help online from Geek Mom Projects which gave us this conversion function [8].

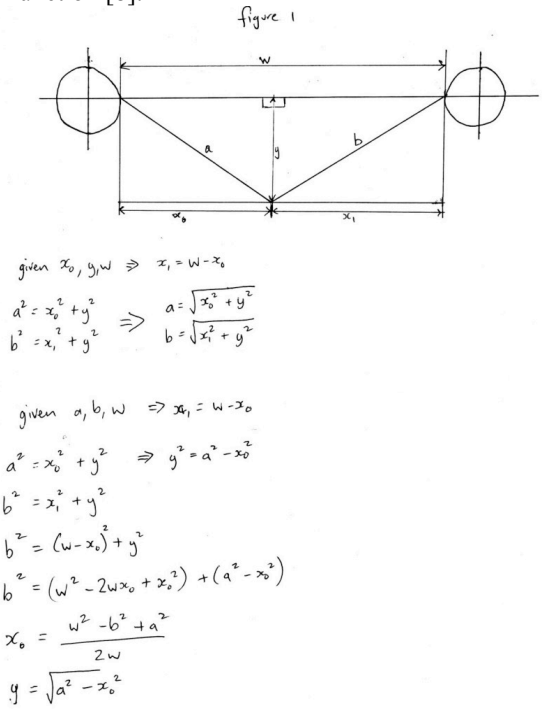

Figure 5. Math for determining string length

X_0 is the distance of the drawing head from the left winch, w is the distance of the winches from each other. Y is the starting height of the drawing head (positive downwards where 0 is in line with the winches). If one knows these numbers one can calculate a and b (the lengths of each string) to figure out how to move the steppers to any point. This is just what we did. Thus our line() function would take in the current position and the target position, and work out the lengths a and b, and adjust each stepper accordingly so that the string attached to it would attain the correct length. This approach worked well.

2.3.2 Bugs/Problems and Solutions

There were many bugs getting the math to work out exactly correctly. The solutions to these bugs was being careful about our equations and our use of sqrt and other such functions. The one caveat with this approach is that since the steppers have no state, the initial Y and X_0, the position of the drawing head when the program begins have to be known. We got around this by defining an "origin" that the drawing head should always start at, every time we restart the system.

2.4 Communication with system

2.4.1 Developments

Finally since we had all of the parts of the system working we needed a way of getting the Gcode commands into the Photon so that it could control the steppers. At first when we tried out smaller images like Smiley faces we could simply paste all the commands into the serial port. However when the size of the Gcode we wanted to draw grew in size, pasting it wasn't sufficient anymore, because writing that volume of data at once to the serial port would lead to loss of data. So we made a processing program that reads a Gcode file and writes it line by line to the serial port following a handshake protocol. When the processing code begins it writes a character signalling to the Photon that it has data to write to the serial port. When the Photon sees this character it begins listening for lines of Gcode. When it receives a line, it blocks, performs the command encoded by that line, and then signals to the processing code that it is ready for the next line. This continues till the processing code signals to the Photon that there are no more lines in the file.

2.4.2 Bugs/Problems and Solutions

It took us a long while to get this working. The serial port proved tricky and the signals weren't always communicated properly. Our first bug was that processing was reading the file properly and the Photon was reading the processing code's commands properly but the processing code wasn't reading the photon's commands. The issue here was that we were *writing* to serial and not *printing* to serial from the Photon. We also had an issue with either party missing a message that was sent. We patched this up by adding delays so that messages had a lower chance of being missed, but it is not completely deterministic. A better solution would be adding resending and timeouts to our handshake protocol.

3. Final Software

- Currently our system begins with an SVG file, that can be obtained from the internet, or manually made, or anything.
- A user then imports this SVG into Inkscape and exports it as Gcode.
- Then a user flashes the Photon and sets the drawing head to the origin.
- Then a user edits the Processing code to give it the name of the Gcode file.
- At this point the user can also select the mode - timed, or handshaked. Timed simply prints each line of the file to Serial with a small delay. Handshaked does the entire handshake protocol
- Then a user runs the processing Code.
- Then you can sit back and watch the Graffitibot do its thing!

FUTURE WORK

We did not have the machine quite ready at the ScienCenter presentation, and a few improvements have already been made to it. Bug fixes and improved processing have allowed us to input more complex images, as shown in the figures. To further improve the software, we would like to make the handshake protocol -- used in the image processing -- more reliable. We can do this by adding re-sending and timeouts.

Now that we have our drawing mechanism in a good state where it can produce precise images given virtually any SVG, we would like to implement the Natural Language Processing part of our initial idea. The goal of the NLP software is to create a system that could be used to draw the "pulse" of the news. A separate software system hosted on a computer/server would be fed a news article (or several) from a user. It then would use NLP techniques to identify keywords that describe the news article(s). It then searches for images (specifically line drawings) related to these keywords and chooses a few of these line drawings to draw. Using our pre-existing drawing software, we can then feed those line drawings as GCode to be drawn on a large wall or paper. Seeing these images in public places increases the number of people engaging with this news and invigorates their desire to consciously affect issues around them in any way that they can. This project is useful because it allows people to visually engage with the most relevant news around them, thereby gaining an awareness of the issues of their time. Whether by raising awareness or galvanizing people to action, this system could positively affect social change.

We also would like to make the system more portable. It is quite time consuming to set up -- and although its simplicity and modularity is a positive, perhaps more rigidity to allow for quicker setup would be an improvement.

If we continue to draw with the Expo Marker on surfaces where the drawings can be easily erased and redrawn, we would like to add a mechanism that can also erase to the drawing head.

1. Information Science at Cornell. Facebook post. May 16, 2016. https://www.facebook.com/Information-Science-at-Cornell-427363513984703/photos/?tab=album&album_id=1008903802497335
2. The Creators Project. "Robot Art: Harvey Moon's Drawing Machines." Online video clip. YouTube. YouTube, 21 June 2013.
3. Hektor. http://juerglehni.com/works/hektor/.
4. Scriptographer. http://juerglehni.com/works/scriptographer
5. mDraw. https://github.com/Makeblock-official/mDrawBot

6. Marginally Clever Library.
https://github.com/MarginallyClever/GcodeCNCDemo
7. Plotter Bot. http://plotterbot.com/robots/
8. Yet Another Wall Plotter.
http://www.geekmomprojects.com/yawp-yet-another-wall-plotter/
9. Garber-Paul, Elisabeth. “Mega-Developer Backs New Trump Statues.” *RollingStone*, 14 September. 2016, http://www.rollingstone.com/culture/news/mega-developer-backs-new-naked-trump-statues-w439701.