DaVinci-Bot Project Documentation

Ellen Brigance, Nate Page and Evan Scullion

***Abstract*—The goal of this project was to design and build a 2D plotter with a TCP server and GUI application. This was accomplished by using stepper motors, CNC frame, a microcontroller, an ESP32 WiFi chip, and C# to build the application. With the project assembled, we were able to have the plotter draw images that were provided to the GUI by the user.**

***Index Terms*—Image Processing, Plotter, Stepper Motor Acceleration, WiFi Server**

I. Introduction

The DaVinci Bot was originally inspired by Sonny in I, Robot [1]. There is a scene where Sonny is drawing his dream and the picture he draws is an entirely cross-hatched landscape and we set out to create an artistic project that would imitate this effect with real world pictures. The resulting machine utilizes a CNC style frame with lead screws to move the gantry and drawing surface as seen in Fig. 1. Our machine takes an image provided by the user, isolate contours based on a user defined threshold, and draws the image on paper using a raster style drawing technique.

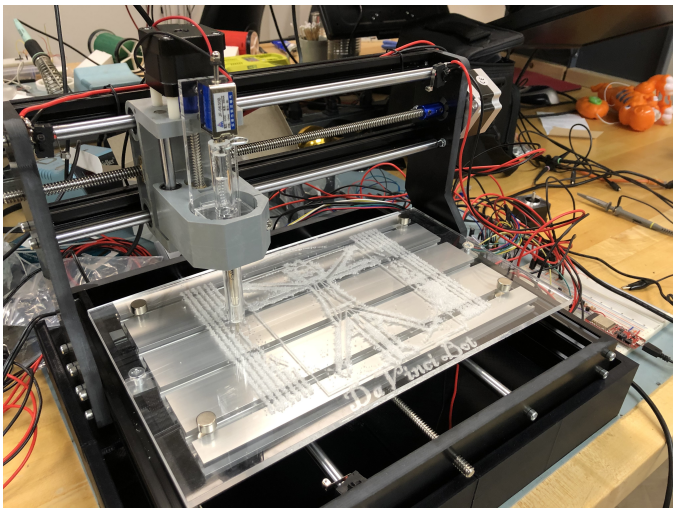

Fig. 1: DaVinci Bot

II. Main Project Components (Top-Down)

We divided this project into three main parts, GUI, Motor Control, and Server. The GUI, apart from the standard features, handles image capture and processing. The server passes information between the devices. Finally, the motor control takes on accelerating stepper motors and deciding when to move each and how far. The following sections will go into detail describing the layout of our machine and how each part functions.

Project Web Page
Code Repository
Manuscript received December 13, 2019

A. Graphical User Interface

We created a graphical user interface (GUI) as seen in Fig. 2 to integrate and interface with our software and hardware. We did this with a Windows Form, coded in C#, using Visual Studio. The purpose of this was so that our software would operate seamlessly in a Windows environments. The high-level language, C# gave us the means to integrate critical components, such as Powershell and terminal commands, and TCP listeners for interaction with our server. In its basic functionality, the GUI allows a user to take an image file, convert it to a monochrome, black-and-white image, customize its contour threshold, generate G-code commands for the CNC machine, and finally, send these commands to the server to begin the printing process. A more in-depth discussion about this process is as follows.

The GUI allows for a user to upload any image file for printing. Users with webcams and external video devices connected to their computers are also able to take pictures within the GUI to be printed. They can also save images to be uploaded again for later use.

Once loaded, the image can be fine-tuned by the user to highlight the desired contours of their photo. This can be done by using the threshold track-bar or the provided text-box if the user wants to type in their desired threshold. The user can also invert the contoured image to show outlines, rather than a filled-in image. Figure 3 shows the default contour setting being used on an image and Figure 4 shows the same image with the inverted contour setting at the same threshold as the original contoured image.

1) Image Processing Software: We used functionality from the OpenCV (Open Source Computer Vision) library to perform our image processing. The library is originally written in C++, but we used Python functions to interface with the library. OpenCV contains functions for finding contouring hierarchies in images. OpenCV uses machine learning to do much of its image processing work [2]. In our GUI, we gave the user the ability to alter the contour threshold of the image and invert the contour style, as previously mentioned. When the user adjusts the threshold in the GUI, our C# code calls a Python script in Windows Powershell, with the desired threshold passed to the python function. In our case, the threshold refers to the depth of the pixels in an image. In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally 255). In a binary image file, all pixels included in the threshold are set to 1, which means they show up in the picture as black pixels, while everything else shows up as white. Finally, we save the user's desired contour image to a default file. The user can then finalize their desired print by pressing the "Generate

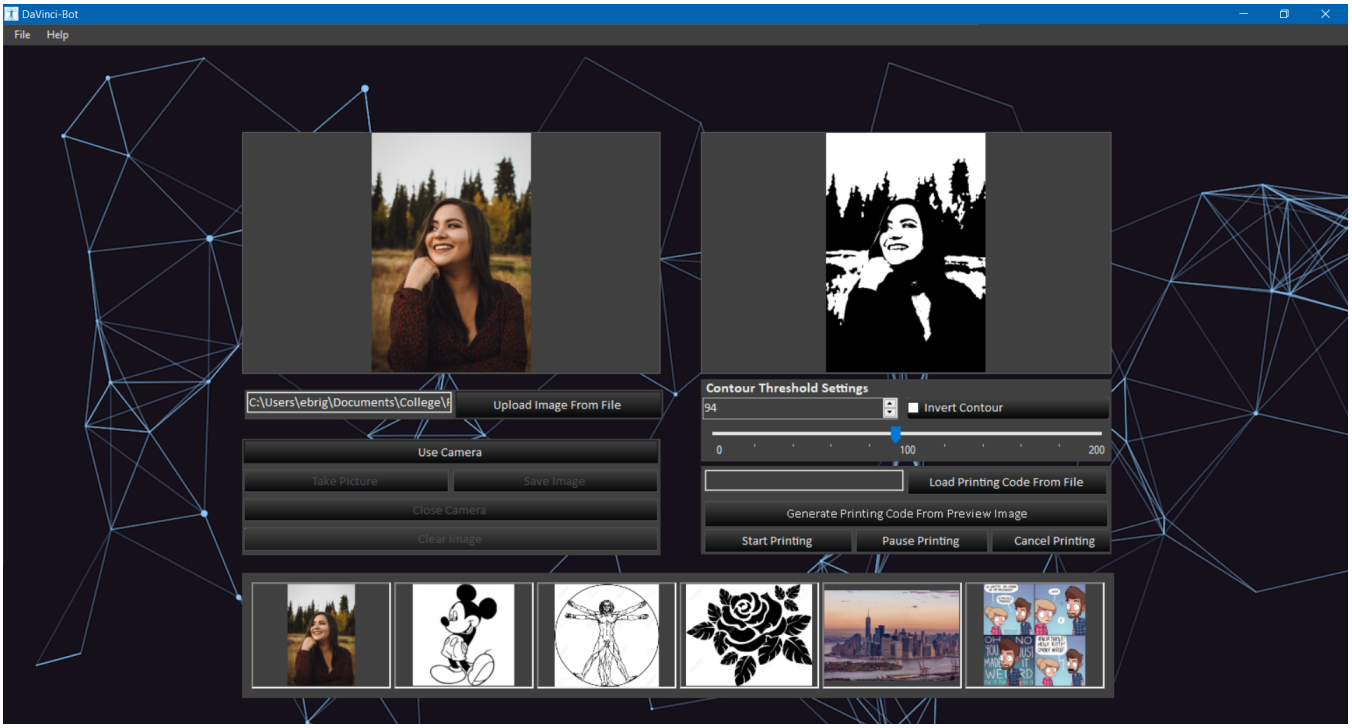

Fig. 2: DaVinci-Bot Interface

Fig. 3: Default Contour in GUI

Fig. 4: Inverted Contour in GUI

Printing Code" button.

B. G-Code Encoding

When the user chooses to generate the code to print their image, our program calls a Python script called imgcode.py, developed by M. "Vidmo" Widomski, which we found on GitHub. To convert the binary image to a G-Code file. G-Code is the language commonly used to send X-Y-Z coordinates to a CNC-style printing machine. There are many settings in the functions we call in imgcode.py. Among these parameters are an image file to be processed, a maximum horizontal width for printing, and a scaling factor. After the G-code file is generated, it is sent to the server for decoding.

C. GUI Challenges

We preset all of the conditions of our G-Code generator for the user in the interest of printing time. The G-Code converting code that we used was only capable of creating raster-style images, which are time consuming to print. Longest-path, smooth curves are much faster to print. It is easy to linearize an image with defined curves into a raster-style plot. However, converting from raster to curves is challenging. The process for that involves even more computationally

expensive, exponential-time, path finding algorithms. Again, in the interest of processing time, we decided that a linear-time algorithm to generate raster images would simplify our problem. In general, pixel-by-pixel image processing and g-code generation was very expensive, depending on the size of the image. We had even loaded images that took up to 30 seconds to process. G-Code for an image with 1000 x 1000 pixel dimensions could generate around 100,000 lines of G-Code. We found a loophole around this by re-scaling the user's chosen image as soon as they uploaded it. Though the details of the image weren't as sharp, for the purposes of printing, it was negligible. This alone cut our processing time down to milliseconds to find contours, and around 3-7 seconds to generate G-Code.

D. WiFi Server

We used an ESP32 Sparkfun Thing Plus set up as an access point to allow for communication without the need for an external network. The server to handle data transfer was implemented on this device in C/C++ with libraries used for control of the TCP socket and internal memory. The client was coded on the PC in C# and integrated into the GUI. When called, the client would take a predetermined file (commands.gco) containing GCode and, once the server sent the allowed number of bits, would send the GCode line by line until either the end of the file or the allowed number of bits was reached. The allowed number of bits was hard coded in memory and represented the amount of data able to be stored in the internal memory of the server. If no more bits were allowed the server would pause writing to memory and begin sending the instructions to the Arduino over UART one by one, waiting for an acknowledge signal after each one was completed. Once the transfer was complete the connections were all closed and the ESP32 memory was wiped by writing zeros to each point in memory, this also reset the memory to allow for new writes.

E. CNC-Style Machine

Our original plan was to fabricate a frame using aluminum t-slot rails and acrylic. We had hoped to use a rack and pinion actuation mechanism for the X and Y movement. Due to several setbacks we were forced to use prefabricated pieces for the basis of the bot. This was acceptable as we were instructed to remember that we are not mechanical engineers so the frame was not to be a priority or large focus. We used the prefabricated frame and attached our own NEMA 23 stepper motors which gave us a larger margin for current allowance and stepping rates. We also added switches to the rails that we set up to trip when the device moved to its limits to prevent damage. Our program would "home" the device when turned on, move to the upper left limit, and set its position. If for some reason it lost its place and attempted to move beyond the ends of the drawing area the switch would kill the program and the power to the motors by sending a signal to the enable pin on each driver. The drawing area was fabricated from $\frac{1}{4}$" acrylic with a laser engraving of a parody of the Vitruvian Man[3].

Evan designed a box-frame in FreeCAD to be printed by a 3D printer for the bot to rest on and to allow for easier wire routing. The box was divided into four pieces to fit onto the available printers and printed using standard black PLA filament.

Peripheral circuits were placed on breadboards to save on cost, manufacturing time and to make them more flexible if changes needed to be made. The ESP32 requires a step down circuit for the UART connection to the Arduino as it takes 3.3V and the Arduino uses 5V for IO. This circuit uses a cross connected pair of BJTs which would pull the opposing transistor into saturation to prevent exceeding the desired voltage on each side[4]. The solenoid requires a high current which none of the IO can handle so we used a 10A relay that is controlled by the Arduino. The high current is provided by a wall adapter with a standard barrel connector. The most important peripherals are the stepper drivers, we used the Pololu TB67S249FTG compact drivers with added heat sinks to remove waste heat. Each controller handled stepping and direction for a single motor as well as stepping down the current running through the coils to allow us to run the device at 28V without destroying the stepper motors. The benefits of running the steppers at a higher voltage being that motors can step faster without losing steps and rotor wobble between steps is reduced.

The power supply we used was a specialty power supply that would provide between 28V and 36V with a total of 9A of current. This single power supply provided power for the steppers in the X and Y direction and the Z direction. We had a small hiccup where we damaged the Z driver due to loose connections to the stepper and were not able to use it in the final product. Luckily this motor was only going to back the solenoid away from the drawing surface when powering down, it was not necessary for normal operation of the device and could safely be removed from the scope of the project.

F. G-Code Interpreter

Instructions passed to the Arduino are formatted as G-Code with fields for X and Y in addition to the type of instruction. The Arduino takes in an instruction from UART and parses it based on the type of instruction given in the "G" field. Depending on the case, it will perform the required calculations needed for moving the motors in the desired manner. The method will then call the move method from the stepper library, stepper_motor.h, to move the corresponding motor the specified amount of steps.

G. Arduino and Motor Control

The micro controller we used was the Arduino Mega 2560. This one was chosen primarily due to the abundance of digital IO and the speed of the processor. Many of the digital IO were used to handle boundary switches as an output and an input were needed for each. The stepper drivers required a connection for step, direction and an enable signal. The Arduino also provided power for the logic on the drivers. A single UART signal was connected to the ESP32 through

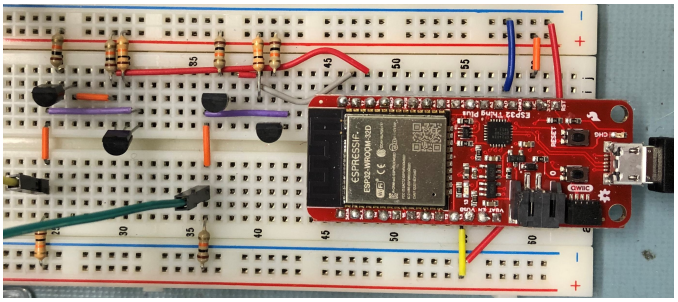

Fig. 5: Level shifting circuit for UART communication between ESP32 and Arduino Mega 2560[4]

the level shifting circuit shown in Fig. 5 for accepting and acknowledging commands.

We controlled the speed and acceleration of the motors by using a software interrupt to trigger at different timed intervals, depending on which motor was to move next. This was done by iterating through a list of the motors and checking to see if they needed to step. After a motor stepped, we would check to see if one or both of the boundaries were triggered and or if we had taken the required amount of steps. We would then calculate a new value to count to and store it into the compare register of the CTC interrupt. This was all handled in the Main.ino file.

III. Conclusion

The machine functioned exactly as specified in our proposal, we were able to take in a picture from the camera and raster draw it on a piece of paper. The solenoid functioned perfectly and better than expected when designing the device but the Z stepper broke down prior to the final demo, this part was not in the original specifications though so it was discarded. One issue we ran into on demo day was that a power setting in Windows 10 timed out the connection to the server which would occasionally cause the GUI to stop sending commands to the server if the image was large enough. Unfortunately this small issue was not resolved as it was not in our code but depended entirely on the system running the GUI.

In the end we were able to achieve our original goal, albeit without some of the stretch goals and without implementing rack and pinion. Operation was smooth without errors or exceptions and no excessive heat was generated in the motors, drivers, or power supplies. The resulting images were distinctly identifiable and had an effect similar to that seen in cross hatched images. Overall this project was a resounding success and demonstrated our combined knowledge and expertise in our area of study.

References

[1] "I, robot," 2004.
[2] "About," 2019. [Online]. Available: https://opencv.org/about/
[3] juzmental, "Vitruvian rick." [Online]. Available: https://www.deviantart.com/juzmental/art/Vitruvian-Rick-439228720
[4] J. Hagerman, "Two transistors form bidirectional level translator." [Online]. Available: http://www.hagtech.com/pdf/translator.pdf
[5] D. Yan, J. Wang, M. Xu, and G. Lian, "Accurately counting algorithm of incremental rotary encoder," *Advanced Materials Research*, vol. 468-471, pp. 225–228, 2012.
[6] J. Miley, "Drawing machine's scintillating work of art," *Interesting Engineering*, Mar 2018. [Online]. Available: https://interestingengineering.com/drawing-machine-creates-scintillating-work-of-art
[7] G. Bruney, "Sketch intricate designs with a hand-cranked drawing machine," Mar 2016. [Online]. Available: https://www.vice.com/en_us/article/aenqq4/joe-freedman-drawing-machine-cycloid
[8] *4848 4x4 CNC Router*. ZenbotCNC, 2019. [Online]. Available: https://www.zenbotcnc.com/4848-4x4-CNC-Router-_p_20.html
[9] M. Budimir, "Rack and pinion drive system: What is it?" Oct 2017. [Online]. Available: https://www.motioncontroltips.com/what-is-a-rack-and-pinion/
[10] M. Starr, "Robotic printer paints portrait of artist in his own blood," Aug 2014. [Online]. Available: https://www.cnet.com/news/robotic-printer-paints-dot-matrix-portrait-of-artist-in-his-own-blood/