**Simulacrum Creation Using Linear Production Techniques (SCULPT):**

**Soap Carver on PIC32**

**Summary**

SCULPT is a 2.5D carver that accepts a user selected image and etches it onto a piece of soap. Operation begins by preprocessing the image into a resized, grayscale version. The pixel values of the new image are then sent to a PIC32 microcontroller via UART. The PIC then controls three stepper motors for each of the X, Y, and Z axes and carves the soap with an end mill attached to a DC motor.

**Introduction**

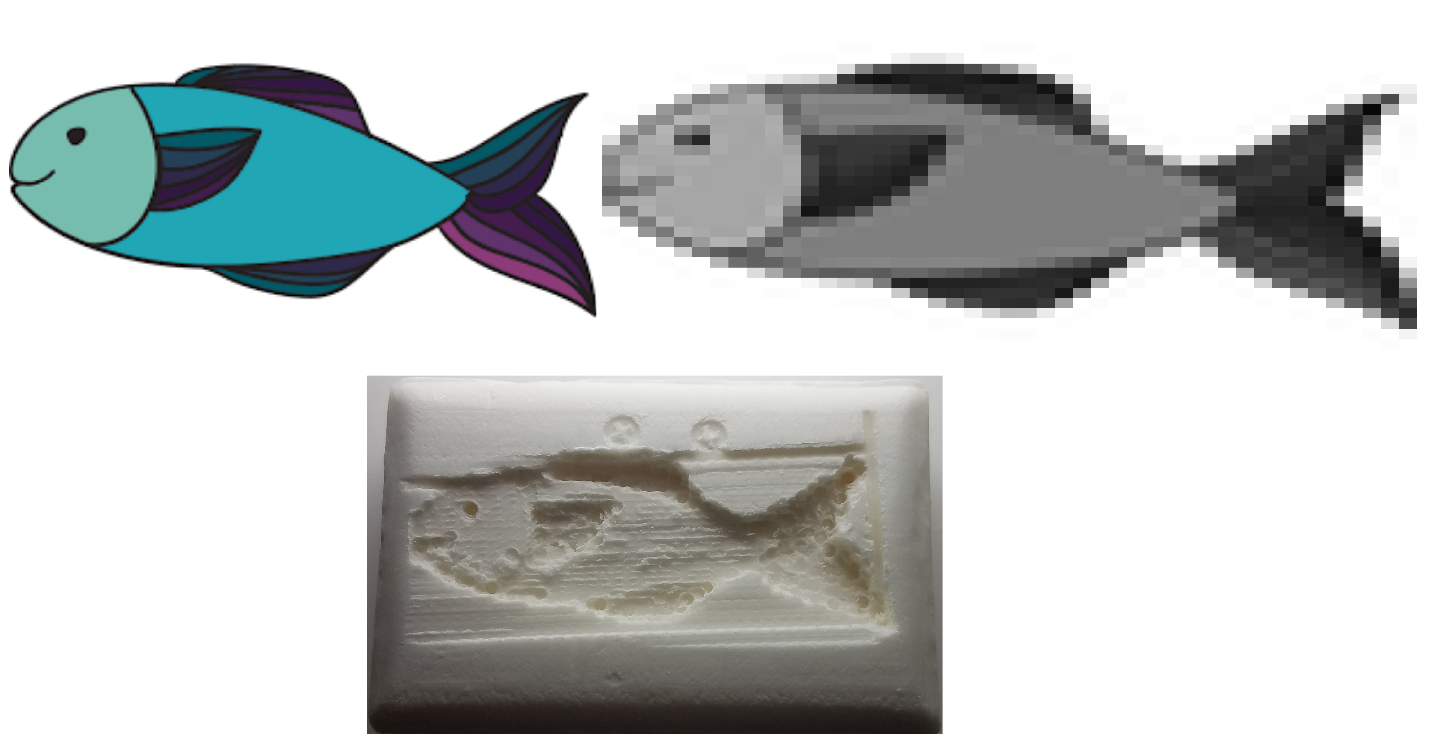
****

Figure 1 - User input image (top left) is processed (top right) then carved (bottom)!

SCULPT carves any image onto a piece of soap with the help of a PIC32 microcontroller. There are three main modes of operation: Serial, Alignment, and Movement. Before using the machine, the user selects the image he or she would like carved and runs a Python script to process the image. The script is part of Serial mode as it uploads the processed image pixel data to the PIC32 via UART before completing. Once all the pixel data is transferred, Alignment mode begins. In Alignment mode, the X, Y, and Z axes are set to a known initial position by driving their respective stepper motors until a button placed at the “zero” position is hit. Once aligned, the user can load the soap onto the build plate. After the soap has been loaded, the user triggers the Movement mode by pressing a button on the device, allowing SCULPT to begin carving the image. When the image has been carved into the soap, the device is then ready to begin the cycle again with a new user image. A high level design can be seen in Figure 2.

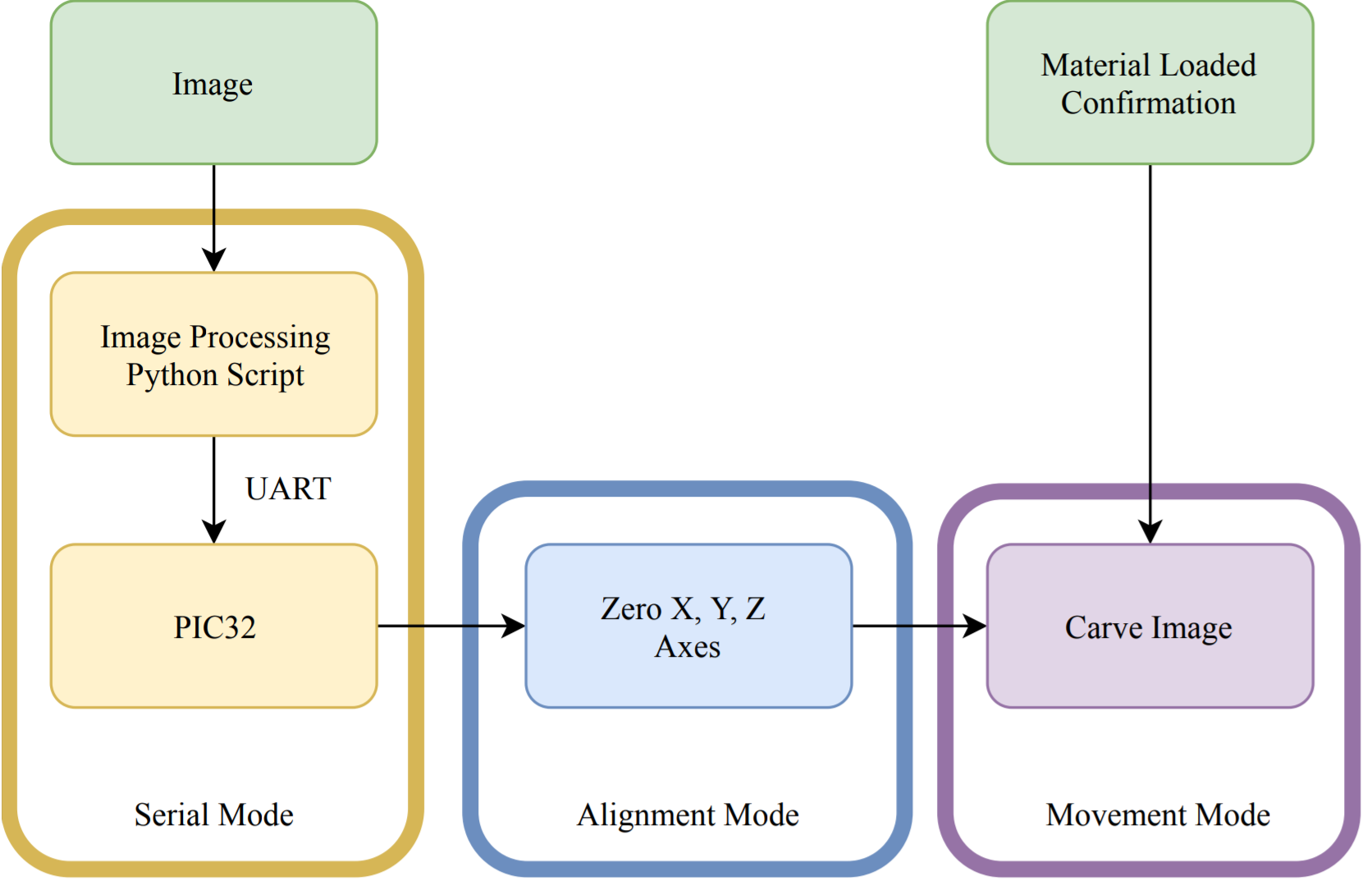


Figure 2 - High level overview for SCULPT, showing all 3 modes of operation as well as user input to the device (green)

**Mechanical Overview**

The physical components of SCULPT were laser cut, machined, or 3D-printed, with all supporting CAD work done using Autodesk Fusion 3601. An assembly modeling the final machine is shown below in Figure 3.

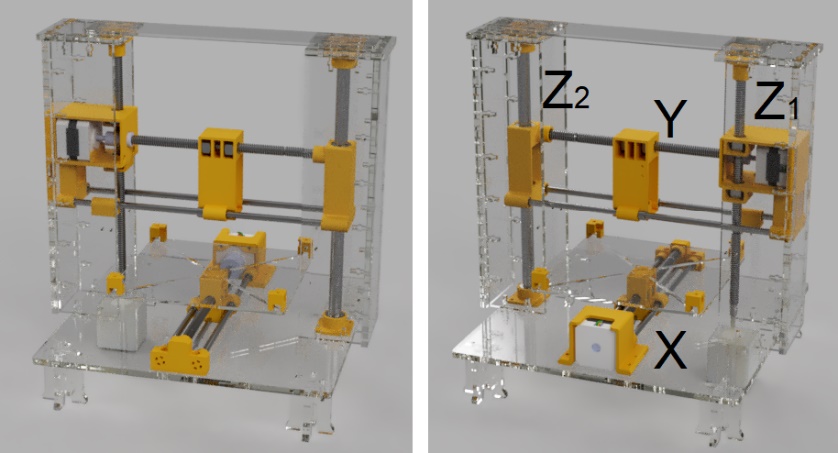


Figure 3 - Front view (left) and Back view (right, with axes labeled) render of SCULPT.

All 3D-printed parts are yellow, machined parts are gray, and laser cut parts are clear.

To provide rigidity, the walls, base, and legs of SCULPT were laser cut from ¼” acrylic. The top piece connecting the two walls was cut from ⅛” acrylic as it was not a structural piece. The ¼” acrylic was necessary to create a stable structure for a detailed carving. All acrylic and 3D-printed parts were connected with M3 screws.

The X, Y, and Z axes used ⅜” diameter steel lead screws and matching ⅜” hex nuts to travel along them. The lead screws and nuts were then housed in 3D printed components to produce linear translation when the lead screw spun. The lead screws were machined to desired lengths, and then modified at the ends to fit properly into housings and allow for proper set screw contact.

Each axis is made of a stepper motor attached to a lead screw through a rigid coupler. While one Z-axis (Z1) uses a lead screw, the other (Z2) is a ½” diameter linear shaft that was machined to length. Matching ½” diameter bronze sleeve bearings were used to travel along this shaft. To provide rigidity, the X and Y axes also have supporting ¼” diameter linear shafts machined to length with matching 1⁄4” diameter bronze sleeve bearings. Linear shafts and bearings were used as their tolerances are both within one thousandth of an inch, resulting in better alignment for the X and Y axes while moving and preventing the Z axis from sagging.

There are three NEMA 17 stepper motors, one to drive each of the X, Y, and Z axes. Since the Z-axis has more load and weight to move than the X and Y axes, the stepper motor has a higher torque rating than the other two motors. While the X and Y axes are rated for 40 Ncm, the Z-axis stepper is rated for 59 Ncm. As a result, the Z-axis stepper motor is larger than the other two and draws the most current during operation.

Both the X and Z axis limit switches are mounted to the acrylic wall and base plate respectively with M2 screws. Because the Z switch was too thin to be hit, a 3D printed adapter was added to the ¼” diameter linear shafts. The Y limit switch was mounted onto the 3D printed housing for the DC motor. This limit switch was too short to hit its zero position and also had a 3D printed adapter added to it to increase its reach.

**Electrical Overview**

A PIC32MX250F128B microcontroller handles all the logic for SCULPT. For this project, the PIC32 was mounted on a small breakout board provided by Cornell’s ECE 4760 course2.

The machine uses Pololu DRV88253 motor drivers for each of the X, Y, and Z axis stepper motors. This model of motor driver was chosen because its logic level input had a large range (2.5 V - 5.25 V which included the PIC32’s 3.3 V output), there was a simple control interface with convenient current-limiting features, and there was a large range for the voltage supply (8.2 V - 45 V).

The direction of stepper rotation (clockwise or counterclockwise) is managed by a connection from the PIC to the DIR pin on each of the DRV8825 drivers (Figure 4). To prevent excessive current consumption when the driver was not in use, the active-low SLEEP pin was also controlled by the PIC. The final connection from the microcontroller to the driver was for the STP pin, which allowed to PIC to manage when each motor was moved. Only full steps were used for the duration of operation since multiple full steps were small enough to carve detailed images.

Limit switches were placed at the “zero” positions for alignment. Another button is used to confirm the material has been loaded onto the build plate. The PIC32 has internal pullups enabled on all these pins so are driven low then pressed.

To make the system portable, a 110 V AC to 19 V DC power converter was used for power. A large kill switch was also added for safety. The 19 V input was within the input range for the DRV8825 stepper drivers, but the DC motor spinning the end mill required a smaller voltage to operate. To reduce the voltage even further, a LM2596 buck converter4 was used to regulate the 19 V input down to 12 V. The regulated output was then appropriate for the L298N5 motor driver which operated the DC motor.

The DC motor driver also needed 5V logic level inputs rather than 3.3V logic. To accomplish this, a 78056 5V voltage regulator and a logic level shifter was used to translate the PIC32’s 3.3V outputs to the appropriate level for the driver.

A full electronic schematic for SCULPT is shown in Figure 4.

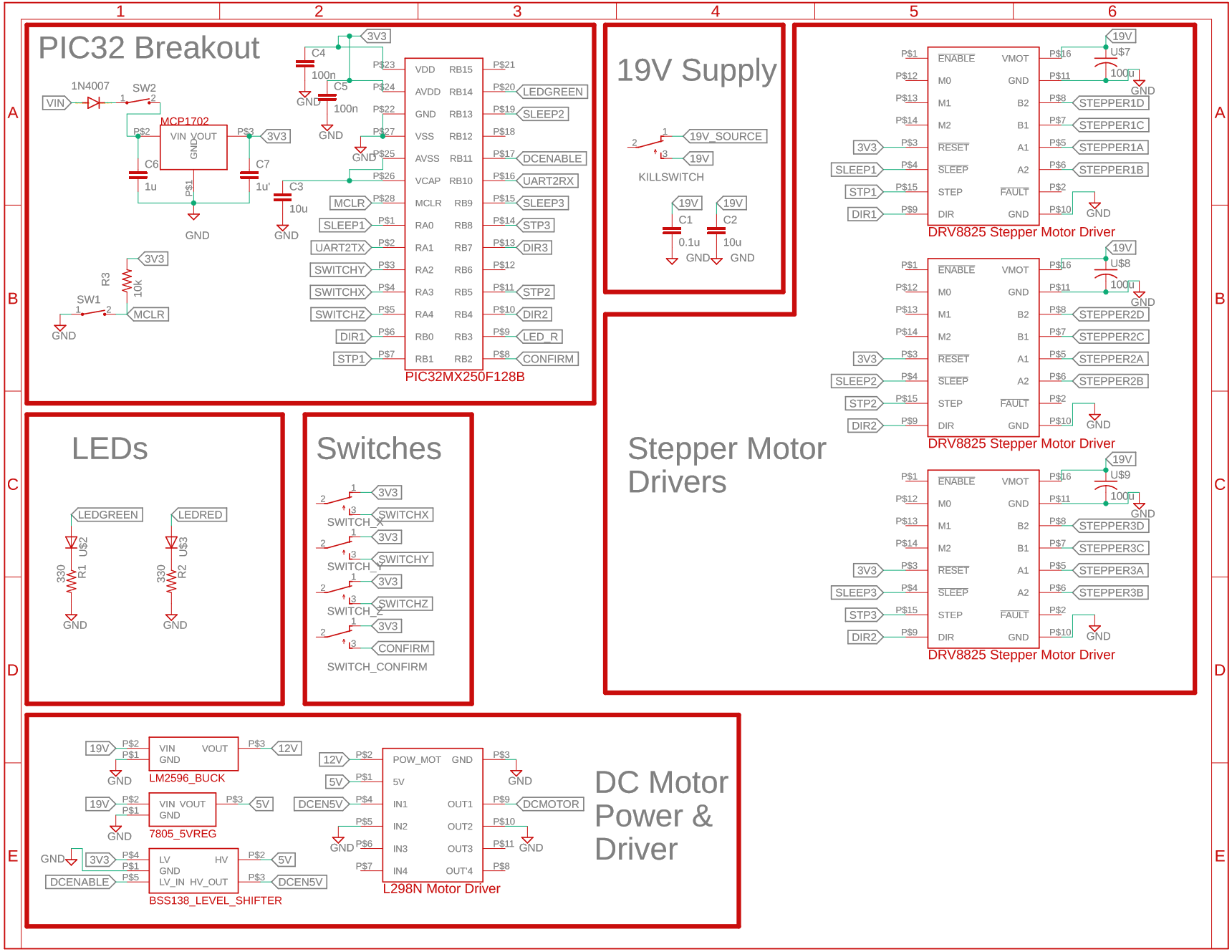


Figure 4 - Full Schematic for SCULPT with each of the major subsystems separated

**Programming Environment**

All programming was done using the MPLABX/XC32 environment from Microchip. The C32 Peripheral Library Guide7 and PIC32MX Peripheral Reference Manuals abstracts many register level manipulations.

In addition to these libraries, a lightweight C threading library, Protothreads8 , was used to schedule SCULPT’s operation. This library was slightly modified9 to allow for millisecond resolution thread yielding, UART receive threads, DMA-driven UART transmit threads. Using this library was advantageous as it allowed for a thread to “wait” for at least a specific number of milliseconds, to “yield” to other threads, and to wait until a variable is a certain value before continuing execution.

**Software Overview**

For this project, there are 3 main threads: protothread\_serial, protothread\_align, and protothread\_move. At a high level, protothread\_serial receives an image over UART and waits until the material is loaded; protothread\_align finds the zero positions for each axis; and protothread\_move controls the logic for carving the image. A detailed state machine for the interaction of SCULPT’s threads is presented in Figure 5.

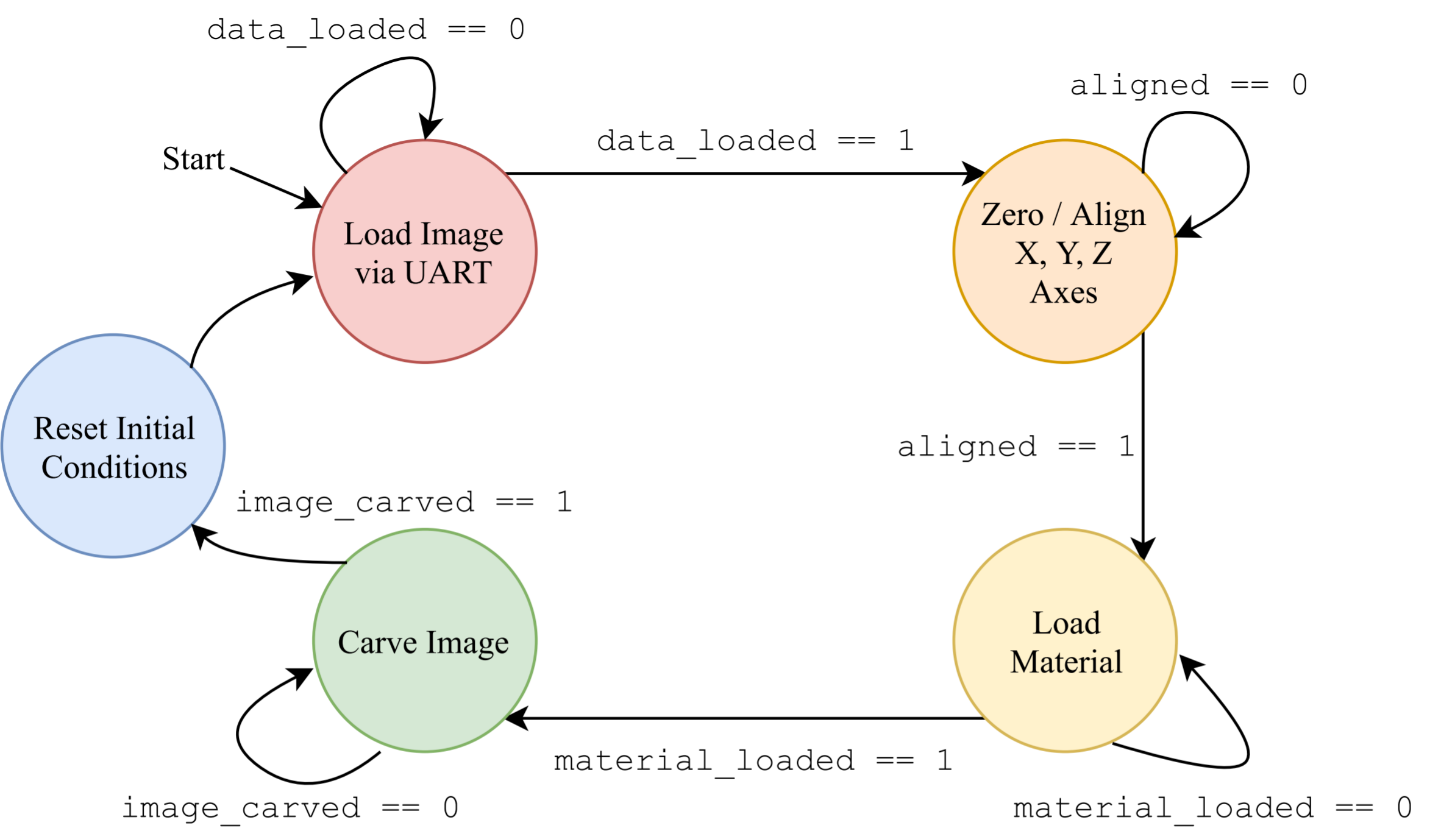


Figure 5 - State Machine for SCULPT

A user created stepper motor struct stepper\_t manages several aspects for each stepper and its driver: which PIC pins were connected to the STP, DIR, and SLEEP pins on the driver; which axis the stepper controls, stp\_num; the current position of an axis, pos; the direction the motor is moving, dir\_move; and how many steps the motor must still move, stps\_left.

From their zero positions, the stepper motors’ limits of movement were decided by hard coding maximum and minimum positions for each axis. If a stepper attempted to move beyond its limit, any movement is disabled to prevent any case of the device damaging itself.

The first stage begins when a user selects an image and processes it with the Python3 script given in Listing 1. To cut the image into a user-defined size grid and assign greyscale values to each pixel, the Python Imaging Library (PIL) and numpy libraries were used. The X and Y dimensions of the rescaled image can set by the user while the Z coordinates is set by the image’s grayscale value (0 - 255) for that pixel. While the PIC32 can support images up to a 100 by 100 size, the current implementation is limited to 39 by 47 pixels. This is because the end mill carving the image has a fixed ~2 mm diameter that determines the resolution of each step and how many pixels can fit on soap bar. The grid size limitation cannot be improved in software and can be improved by making the soap larger or making the end mill smaller.

Once the image is done being processed, the data is sent to the PIC32 via a UART connection. The script opens, writes to, and closes a serial port using Python’s Serial library. First, information about the image is sent. The first transmission for valid data messages of image information was sent with the char ‘s’ first, followed by the size of the x dimension, size of the y dimension, and the largest z coordinate seen in the image (e.g. s 39 47 200 for a x dimension of 39 pixels, a y dimension of 47 pixels, and a largest greyscale value of 200 for the whole image). After this first step, pixel data is sent. Valid data messages for a “pixel” were sent with the char ‘p’ in first, followed by the x index, y index, and the grayscale value (e.g p 0 1 27 to encode the (0, 1) pixel with a grayscale value of 27). The transmission is terminated when the char ‘e’ is sent. The PIC32 handles all input in the protothread\_serial thread at a 9600 baud rate.

To run the script, another Python library called argparse was used to easily incorporate variable inputs to the script. The user must specify the port of the UART connection with the -p flag and the absolute path for the image file with the -i flag. The user can also optionally turn off debug mode for the script with the -d flag; specify the size of the x dimension with the -x flag; and specify the size of the y dimension with the -y flag. By default, debug mode is turned on, the x dimension is set its largest value (39), and the y dimension is set to its largest value (47).

To input an image stored in /home/user/Downloads/image.png to /dev/ttyUSB0 with debug mode off, the following command would be typed into the terminal. The order the arguments are given does not matter, but the -p and -i arguments are required.

./processImage.py -p /dev/tty/USB0 -i /home/user/Downloads/image.png -d

An example image before and after processing the image is shown in Figure 6.



Figure 6 - An example image before (left) and after (right) processing at the largest resolution of 39 by 47 pixels

As the image is loaded, SCULPT does not move. Once the image is finished uploading over UART, a green LED lights up, the global variable data\_loaded is set to 1, and the protothread\_serial thread yields until the current image is carved (image\_carved == 1) before being scheduled again.

Now that the image values are loaded, SCULPT zeros the three axes by moving each stepper motor until a limit switch is pressed. A pulse duration of 600 us to the DRV8825 drivers’ STP pin was found to be a good rate for each stepper motor to operate at a reasonable speed. This timing management was done with the PIC32’s Timer2 and a user-implemented interrupt service routine (ISR). When Timer2’s ISR is called, it checks if there are any stps\_left for a stepper and if it is enabled. If so, the STP pin is toggled and the motor’s tracked position pos is updated. When there are no more stps\_left, or if any limits are going to be exceeded, the stepper is disabled.

The motors move until their zero positions are found, and the thread waits until the material\_loaded button is pressed. Before pressing the button, the user should place their soap onto the build plate and make sure it does not move by tightening the fasteners. The protothread\_align thread then yields until another image is loaded in protothread\_serial.

Once SCULPT has the image to carve, has aligned its axes, and has material loaded, the stepper motors move according to the image’s pixel coordinates (X and Y axes) and grayscale values (Z axis) mentioned earlier. Originally, SCULPT directly carved the z position from the grayscale value directly. This placed lots of force on the end mill if a large, deep line or square was carved since the end mill would have to go through massive amounts of material sideways. To negate this issue, the implementation was changed to carve all the locations with a specific Z layer depth before incrementing the Z depth.

The carving begins at the uppermost Z height and carves all pixels that have a grayscale value whose final Z height is lower than this uppermost value. After going through the whole image for this Z height, the Z height target is changed to be the current Z height + 10 steps further into the material (effectively assigning 10 steps to each of the 255 grayscale levels), and the whole image is carved again. The image is carved by moving sequentially through each column (y-coordinate) for a given row (x-coordinate).

Because the image is carved layer by layer, the Z axis has to be reset to its starting position above the soap whenever the next pixel position is not carved. For instance, carving a white line in the middle of a black square requires the Z axis to carve deep into the soap for the black regions, raise itself to avoid carving or bumping into the white line, and then lower itself to carve the remaining black region. This Z raising is done whenever a new row is being carved, the next pixel’s y-coordinate is more than one spot away, and at the end of the carving.

Although all the axes were zeroed in the Alignment phase, the axes drifted while the image was carved. Because of this, each axis had to be re-zeroed during the carving. Since the Y axis moves the most, it had the worst drift and is re-zeroed after going through each row. The X axis was recalibrated whenever next pixel’s x-coordinate was more than 1 position away from the current pixel. The Z axis was recalibrated every time the X axis was calibrated or every time the Y axis was calibrated if the Z position corresponded to a grayscale value of less than 30 or more than 120. The latter condition was done because the Z axis stepper motor position drifts the most when the difference between the target and start position was too little or too small. With all the recalibrating, the carving time was greatly increased, but it could be solved with mechanical fixes such as replacing all hex nuts with anti-backlash nuts.

Once the current image had finished being carved, SCULPT was ready to accept another one through UART. The Z axis raised the end mill from the last pixel to it start position so that the user can pull the soap out from the build plate. After sending the image over UART, SCULPT moved into Alignment phase again.

**Conclusion**

A 2.5D carving device was created from the ground up. An example user image and the result from SCULPT is shown in Figure 7. The image below took less than a minute to upload via UART and about 4.5 hours to carve.



Figure 7 - An example carving’s input image (top left), processed image (top right), and the final carving (bottom)

Despite the success in carving an image, SCULPT could be improved in future iterations. Users found uploading the image over UART took too long. While sending small images (e.g. a 20 x 20 grid) take less than 30 seconds to upload, the largest soap image (37 x 49) takes about a minute to send. Also, the start position (ie the (0, 0, 0) coordinate) was found manually and programmed into the software. Having a controller to move then set the X, Y, and Z start positions would be more practical than finding each number by hand. A final major improvement that could be made to SCULPT would be using anti-backlash nuts instead of hex nuts for each of the parts traveling on the threaded rods. Doing this would reduce the recalibrating done while carving the image, make each stepper movement more accurate, and make the process overall quicker.

**Resources**

* Microchip ([www.microchip.com](http://www.microchip.com))
* Pololu ([www.pololu.com](http://www.pololu.com))

We would also like to thank Matthew Sherman for helping machine many of the parts and providing great suggestions on making SCULPT mechanically stable. We would also like to thank Bruce Land and Joe Skovira also for providing great electrical and mechanical suggestions.

**Reference List**

[1] <https://www.autodesk.com/products/fusion-360/overview#banner>

[2] <http://people.ece.cornell.edu/land/courses/ece4760/PIC32/target_board.html>

[3] [www.pololu.com/product/2133](http://www.pololu.com/product/2133).

[4] <https://www.amazon.com/Controller-H-Bridge-Stepper-Mega2560-Duemilanove/dp/B01BWLICV4/ref=sr_1_2?keywords=ln298n&qid=1554886797&s=electronics&sr=1-2>

[5] <https://www.amazon.com/LM2596s-Converter-Step-down-Regulator-Stabilizer/dp/B07CVBG8CT/ref=sr_1_12?keywords=dc+dc+buck+converter&qid=1554882908&s=electronics&sr=1-12>

[6] <https://www.sparkfun.com/datasheets/Components/LM7805.pdf>

[7] <http://ww1.microchip.com/downloads/en/DeviceDoc/32bitPeripheralLibraryGuide.pdf>

[8] <http://dunkels.com/adam/pt/>

[9] <http://people.ece.cornell.edu/land/courses/ece4760/PIC32/index_Protothreads.html>

To zip

* All images
* Python script
* My jacket