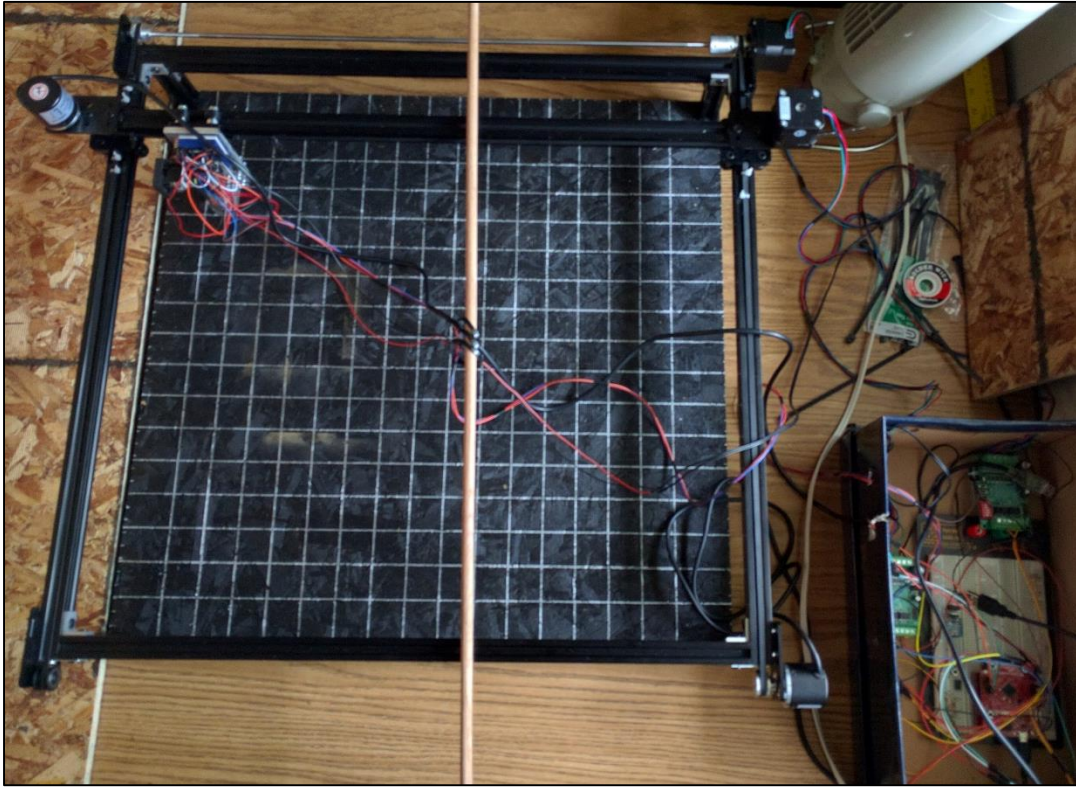


P.L.E.D
PLAQUE LASER ENGRAVER DEVICE



CASEY WOOD
ZACHARY GARRARD
JUSTIN COX
TATE SHORTHILL
INCASEYWOOD@GMAIL.COM
DDCBANZ@GMAIL.COM
JUSTIN.N.COX@GMAIL.COM
SHORTHILLT@GMAIL.COM

COLLEGE OF ENGINEERING
UTAH STATE UNIVERSITY

05/04/2016

TABLE OF CONTENTS

1.	EXECUTIVE SUMMARY	3
2.	INTRODUCTION	4
3.	REQUIREMENTS.....	5
3.1	Functional Requirements	5
3.2	Non-Functional Requirements	5
4.	SPECIFICATIONS	6
4.1	Performance Characteristics	6
4.2	Physical characteristics	6
4.3	Environment	6
4.4	Cleanliness	7
5.	METHODS	8
6.	RESULTS	14
7.	DISCUSSION	17
8.	CONCLUSION.....	18
9.	APPENDICES.....	19
9.1	Appendix 1 – Budget	19
9.2	Appendix 2 – Timeline	20
9.3	Appendix 3 – Bill of Materials	23
9.4	Appendix 4 – Schematics	24
9.5	Appendix 5 – Code	25
9.6	Appendix 6 – Mathematical Model	73
9.7	Appendix 7 – Machining	76
9.8	Appendix 8 – Assembly.....	89

1. EXECUTIVE SUMMARY

The Plaque Laser Engraver Device (PLED) is a system which engraves high quality images into wooden plaques. Our project can create quality decorations for homes and offices, the plaques could also be used for awards and recognition.

This report contains all information pertaining to the design and construction of a (PLED) system, as well as its capabilities. The introduction discusses the motivation behind the project and its purpose. The requirements and specifications sections contain specific functional and physical requirements of the PLED. The methods section explains the design steps and tasks involved in building the PLED system. The results section displays plaques that have been engraved and the progression of our plaques. The discussion section considers the plaques and the implications of our results. The conclusion discusses how well our team followed through with our plans and potential improvements or redesigns.

Additionally there is an appendix section, which provides our timeline, budget, and mathematical model. It also contains the design details for the construction of this system including the code, drawings for machined parts, bill of materials, and schematic.

After considering this document, the reader will be capable of not only understanding the design of the PLED, but recreating it.

2. INTRODUCTION

The first step in designing the Plaque Laser Engraver Device (PLED) was specifying our desired results and selecting specific design requirements to yield said results.

In specifying our desired results, the first considerations made were cost and quality. The team was familiar with laser engraving systems and designs that used a lot of software and materials that were cost prohibitive, especially for students and hobbyists. The team wanted to design something that would be significantly less cost prohibitive, but without sacrificing quality or capability. It was determined to create a very high quality, high fidelity design at a price significantly lower than that for similar systems. In order to do so, the team would have to write custom software for every portion of the system, from image processing to CNC control. The team would also have to find high quality components at a low cost and find a way to minimize custom machining, as many hobbyists and students cannot afford to have parts machined professionally.

The earliest decisions were in image resolution and in machine control. As it is simple to design a black and white laser engraver, it was determined to do something more difficult that could produce much higher quality reproductions of images. Initially the team wanted to design a system capable of engraving 256 shade grayscale images, however upon further investigation, it proved difficult to verify that high of resolution, and there were concerns about calibrating so many different shades. The team settled upon an 8 shade solution, which would provide much more detailed reproductions than a black and white system, but had fewer challenges than a 256 shade solution. Machine control was an easier decision to make. Industry standard CNC machines use g-code, which is a numerical control programming language capable of controlling all of the basic motion and burn systems for the PLED.

Additionally, we selected to use the Tiva-C family of microcontrollers from Texas instruments because we had experience with them, and because they are low cost.

A more in depth look at the details of this design is provided in section 5 of this document.

3. REQUIREMENTS

The specific requirements imposed upon the Plaque Laser Engraver Device (PLED) design follow.

3.1 Functional Requirements

- 3.1.1 PLED shall use a laser to burn grayscale images onto wooden plaques.
- 3.1.2 PLED shall operate like a CNC cutter to create facsimile images.
- 3.1.3 PLED laser shall turn off immediately upon system error.
- 3.1.4 PLED shall not function after the xy axis alignment is disturbed.
- 3.1.5 PLED shall use commercial laser driver to control laser diode.
- 3.1.6 PLED shall utilize a PC, microcontroller, and laser module.
- 3.1.7 PLED shall utilize G-Code for positioning.
- 3.1.8 PLED shall use no more than 7 Watts to drive diode and shall utilize a single 120 VAC power input.

3.2 Non-Functional Requirements

- 3.2.1 PLED shall create novelty plaques with wood burnt images at least 9"x12" in size.
- 3.2.2 PLED shall not require user input beyond an image and power to the system.
- 3.2.3 PLED shall operate independently after an image is provided and a plaque is placed.
- 3.2.4 PLED shall meet laser industry/government standard laser safety specifications.
- 3.2.5 PLED shall be less than 3 cubic feet in size.
- 3.2.6 PLED shall weigh less than 30 pounds.
- 3.2.7 PLED shall be capable of surviving gentle motion, but no drop tests are required.
- 3.2.8 PLED structure shall be made of aluminum.
- 3.2.9 PLED shall be capable of operating in a temperature range of 40° - 110° F.
- 3.2.10 PLED shall operate in regions with less than 90% humidity.
- 3.2.11 PLED shall use a noise-free power source.
- 3.2.12 PLED shall cost commercially no more than \$1000 (PC not included).

4. SPECIFICATIONS

Detailed specifications for the Plaque Laser Engraver Device (PLED) are laid out in this section.

4.1 Performance Characteristics

- 4.1.1 PLED will use a 445 nm laser at 2 watts power to keep engraving time under 6 hours.
- 4.1.2 PLED will burn images with a minimum resolution of 3 bits, 8 shades of grayscale.
- 4.1.3 PLED motion on a two dimensional axis will be accurate to a minimum of 0.5 mm.
- 4.1.4 PLED laser will turn off immediately upon system error.
- 4.1.5 PLED will utilize a microcontroller for processing and control.
- 4.1.6 PLED will utilize G-Code for position tracking and movement of the laser module.
- 4.1.7 PLED will have a minimum resolution of 85 dpi (dots per inch).
- 4.1.8 PLED will have a maximum image size of 1600 x 1600 pixels.
- 4.1.9 PLED will utilize 5 V logic.
- 4.1.10 PLED will utilize 5 VDC for the microcontroller.
- 4.1.11 PLED will utilize 18 VDC, 3.5 A max for the motor driver
- 4.1.12 PLED will utilize 5 VDC, 1.7 A max for the laser driver.

4.2 Physical characteristics

- 4.2.1 PLED will be less than 3 cubic feet in size.
- 4.2.2 PLED will weigh less than 30 pounds.
- 4.2.3 PLED structure will be made of aluminum.
- 4.2.4 PLED will not function after the xy axis alignment is disturbed.
- 4.2.5 PLED will use no more than 7.65 Watts to drive diode and 120 VAC power input.
- 4.2.6 PLED will meet laser industry/government standard laser safety specifications.

4.3 Environment

4.3.1 Natural Environments

- 4.3.1.1 PLED will be capable of operating in a temperature range of 40°-110° F.
- 4.3.1.2 PLED will operate in regions with less than 90% humidity.

4.3.2 Induced Environments

4.3.2.1 PLED will be capable of surviving 3 g's, but no drop tests are required.

4.3.2.2 PLED will use a power source with no more than 2% total harmonic distortion.

4.4 Cleanliness

4.4.1 Electromagnetic interference (EMI)

4.4.1.1 The unit will meet the EMI requirements for class IV equipment as specified in MIL-STD-461.

4.4.2 Ventilation

4.4.2.1 If smoke particles exceed 500 ppm concentration a fan will activate to vent fresh air

5. METHODS

A number of tasks, divided into different phases, must be accomplished in order for the Plaque Laser Engraver Device (PLED) project to be successful. A detailed look at the timeline for those tasks can be found in Appendix 2, while a detailed overview of each tasks and its specific execution follows after the block diagram in figure 1.

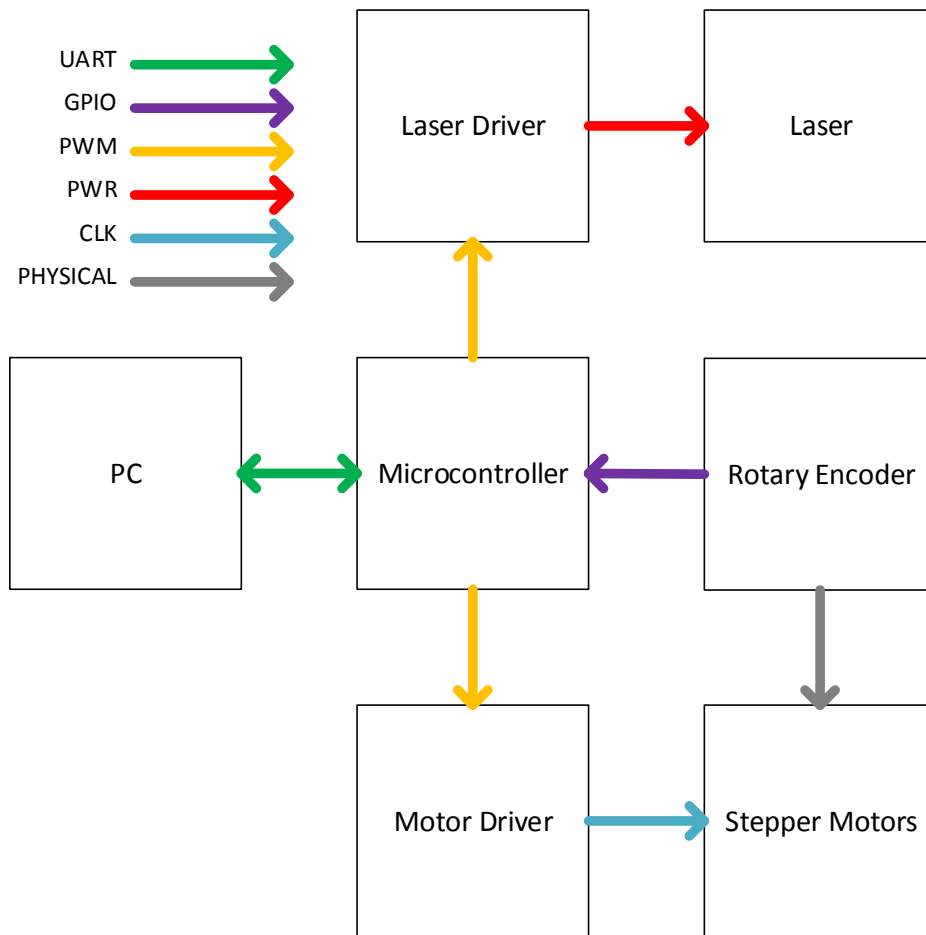


Figure 1. PLED System Block Diagram

Phase 1: Initial documentation

Specifications document – Determined program requirements and compiled them into a specifications document which drove the design of the PLED.

Proposal – A proposal was written to outline the design and details of the PLED project and how it was to be conducted.

Select specific components to be used – The PLED team used the specifications document and proposal to determine which components would be used for the PLED, then the specific manufacturers and models were identified.

Preliminary Design Review slides – In preparation for a preliminary design review, the last details of the design were determined and a presentation was prepared outlining those details and that design.

Preliminary Design Review – A preliminary design review was conducted with peers and engineering professionals to assure that the design was achievable and sufficient to yield the desired results. Feedback was taken and applied as the design matured.

Phase 2: Build prototype

Procure components – Components were procured in order to begin development of the PLED. A full list of those components and their descriptions can be found in Appendix 3.

Configure serial communications between PC and microcontroller – A serial communications line was configured and used to pass the g-codes to the interpreter housed in the Tiva-C microcontroller. This was accomplished by opening a UART serial communications port on the PC and on the microcontroller. Using a special USB serial cable, data could be passed via the UART protocol between the microcontroller and host PC. This code can be found under “Serial and Parser” in Appendix 5.

Write software to ingest and process image and convert to g-codes – Software was written to ingest the initial image, adjust dimensions according to board size, and convert the image to 8 shades of grayscale. At the same time, the image is turned into a series of g-codes which are later interpreted by the microcontroller to specify location and shade. Standard g-codes for position (G00 and G01) and dwell duration (G04) were used. In addition, codes specified for a milling machine were adjusted to meet our needs. The codes for spindle on and off (M04 and M05) were used to determine laser state (on or off). The code for spindle speed (S##) was used to set burn intensity (S00 is the darkest, S07 is the lightest). The milling machine code for termination (M02) was

also used to fulfill the same purpose. The code for this process can be found under “Image Ingestion” in Appendix 5.

Test components individually – Each electrical component (motor drivers and motors, encoders, laser driver and laser) was tested individually. We characterized the functionality and performance metrics of these components, including motor resolution, speed, torque and step sizes, encoder pulses and laser threshold and intensity capabilities. Our motors are specified to 400 pulses per revolution, however they are capable of as small as 1/32 steps using microstepping. We decided to use 1/16 steps, meaning that there are 6400 microsteps per revolution. Our encoder is capable of 600 pulses per revolution, however it is a quadrature encoder, and can therefore provide 2400 pulses per revolution. The laser, once set at threshold, can have its intensity set using a PWM. We configured our laser to have a 480 pulse duty cycle, so dividing that 480 between the 7 darkened shades (plus white for the 8th shade) allows us to generate varying shades of grayscale.

Configure communications from microcontroller to motor driver – The motors are controlled using a pulse width modulation. By varying the microstep sizes and frequency of pulses, we set the motor resolution and speed. The code for doing this can be found under “Microcontroller” in Appendix 5.

Design mechanical system – Our mechanical engineer designed a sturdy, reliable, and very structurally sound gantry for the PLED. The gantry can allow for images that are roughly 15x15 inches. It also prevents slippage on the motors and belts and assures adequate communication between the motors and encoders to allow for system feedback. An image of the working CAD model is included in figure 2 below.

Implement parser for g-code files – A g-code parser was designed on the host PC to parse and do a partial interpretation of g-codes on the host PC. This parser ingests the g-codes from the .gcode file and strips them down into a format easy to send via the UART serial protocol. These formatted codes are passed via the serial code mentioned above to the microcontroller. The code for this process can be found under “Serial and Parser” in Appendix 5.

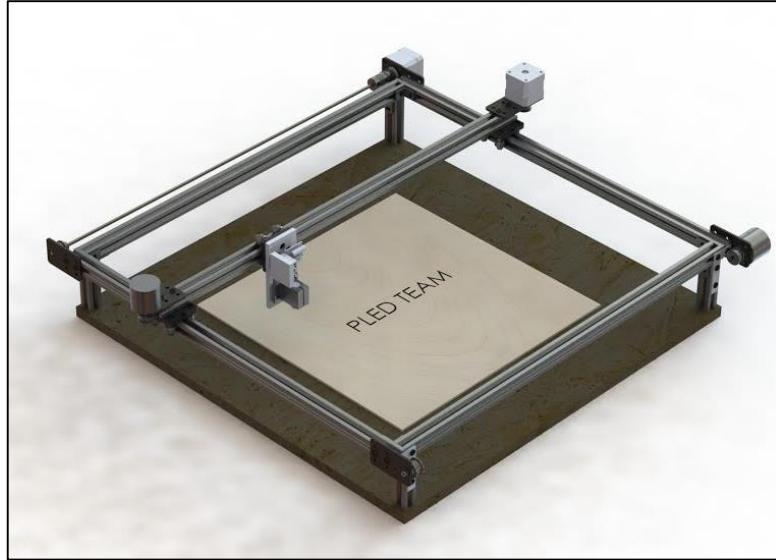


Figure 2. CAD model of the PLED system.

Write software GUI and interface – A graphical user interface (GUI) was written in C# to accept user inputs and host all of the PC protocols in the PLED system. This code first determines plaque size, then allows the user to ingest an image. The user can then select the image size and location on the plaque. The software gives a time estimate for completion of engraving. When the user submits the information, the GUI calls the serial protocol. Code for the GUI can be found under “Graphical User Interface” in Appendix 5. An example of the GUI is displayed in figure 3 below.

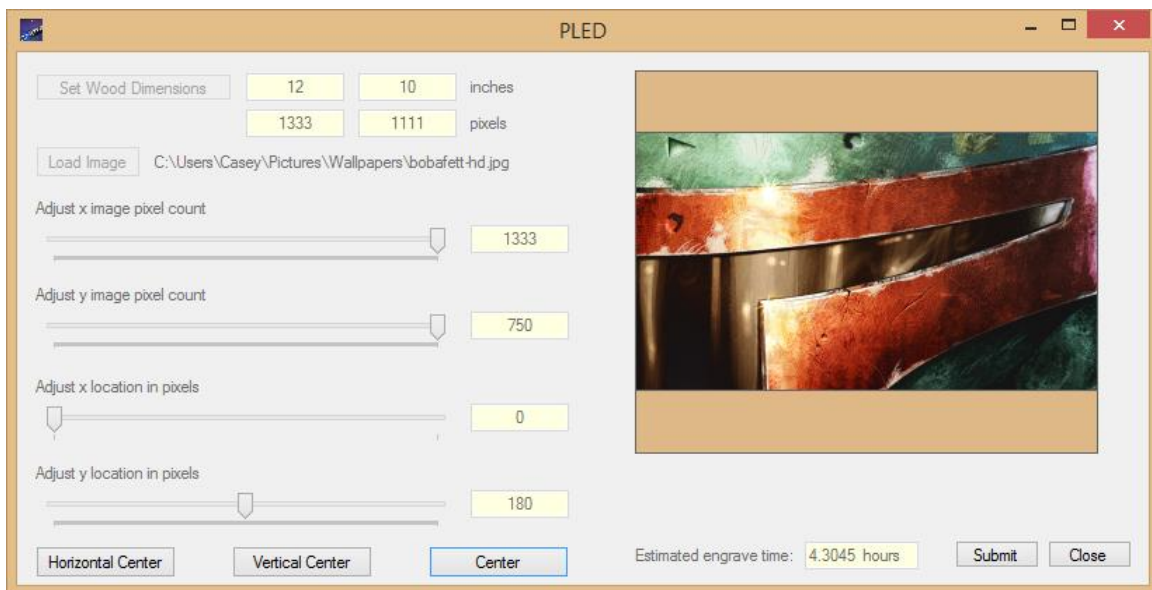


Figure 2. Example of the PLED GUI.

Procure mechanical system components – After the mechanical structure was designed and determined adequate, the components for the system were purchased.

Design and implement feedback system – The feedback control system of the PLED uses rotary encoders to determine the actual location of the laser head. Comparing the actual location to the expected/desired location allows us to correct for errors of alignment. The code for this feedback can be found under “Microcontroller” in Appendix 5.

Mathematically model system – A mathematical model was developed for the PLED system. This model was used to calculate the maximum torque, speed, and load that our motors could bear. The full mathematical model can be found in Appendix 6.

Implement and calibrate laser control system – The 2 Watt laser in our system had to be calibrated and focused, and we had to specify a control system. As outlined in the section “Test components individually,” our laser system is controlled by a PWM with a 480 pulse duty cycle. This allows us to specify intensity, and therefore burn contrast. The code outlining the control of the laser can be found under “Microcontroller” in Appendix 5.

Integrate g-code interpreter into microcontroller – A g-code interpreter was integrated into the microcontroller. This code received the parsed codes via the UART serial protocol and interpreted them to call the functions to actually control the PLED. The code for this procedure can be found under “Microcontroller” in Appendix 5.

Machine parts – Despite our efforts to avoid professional machining, some of our components did have to be machined to accommodate our design. Most of this machining was to enlarge holes, counter sink / counter bore screw holes, or to cut down parts that were too large in size. Additionally, one part, the laser mount, was custom designed and machined for our application. All of the drawings for these modifications can be found in Appendix 7.

Assemble system – Once all components were procured and modified according to our design specifications, we assembled the system according to our design. The physical system can be seen in the photograph below in figure 4 below.

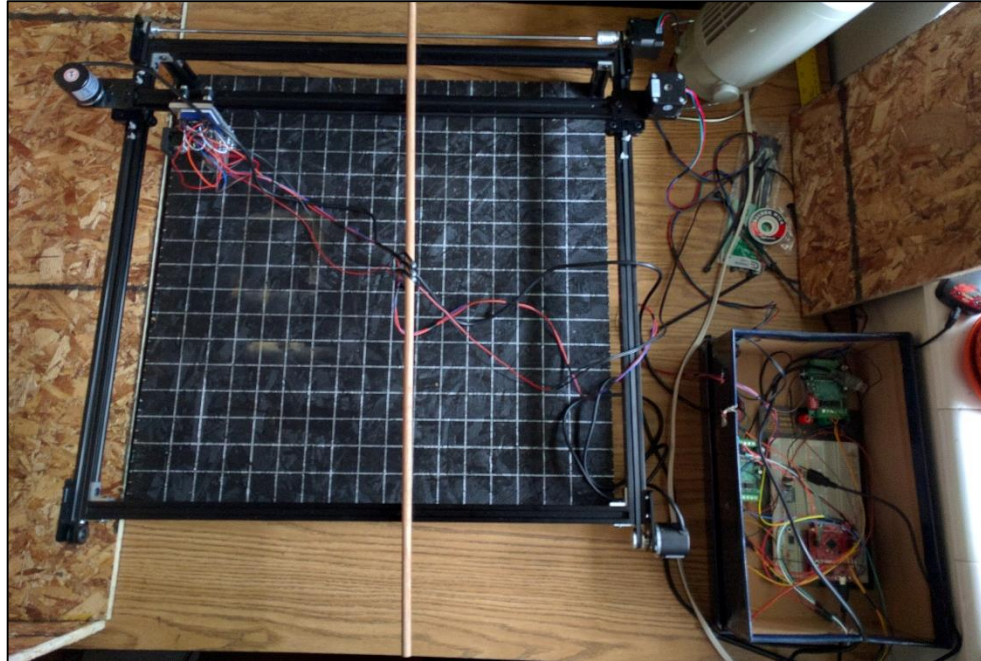


Figure 4. The PLED system

Phase 3: Testing, final documentation, additional features

Write user's manual – A user's manual has been written to outline how to configure and use the PLED. It explains how to install and launch the host software, as well as how to communicate between the host PC and the PLED system.

Write final report – Time was set aside to write this report, detailing our design, progress, and results.

Testing and verification – Once all was assembled, our system had to be tested and calibrated to assure that it met our specifications. This testing was also a period of adjustment, as we tried to assure that our images fit the criteria that we had outlined.

Design poster – A poster has been designed to advertise and outline the PLED, its capabilities, and how it works.

6. RESULTS

The Plaque Laser Engraver Device (PLED) was able to reproduce 8 shade grayscale images, as specified and desired. This was verified primarily by inspection. Figure 5 shows an 8 shade image used to verify the different shades of grayscale engraved by the PLED. Additionally, a series of engraved images are shown below in figures 6 - 11 to further show the successful results of the PLED.



Figure 5. An 8-shade image used to verify the shades. The white shade is between the others.



Figure 6. Hack USU logo engraved into a piece of poplar. Our first successful engraving. Contrast was lacking and shades are too dark.



Figure 7. Utah State logo engraved into poplar. Second engraving. Contrast is still inadequate and there are visible alignment issues.



Figure 8. An image of Boba-Fett from Star Wars engraved on poplar. This was our first engraving with improved contrast, however the alignment had issues.



Figure 9. An image of the Logan Temple engraved on poplar. Contrast and alignment are good, however we learned the ill effects of engraving against the wood grain.

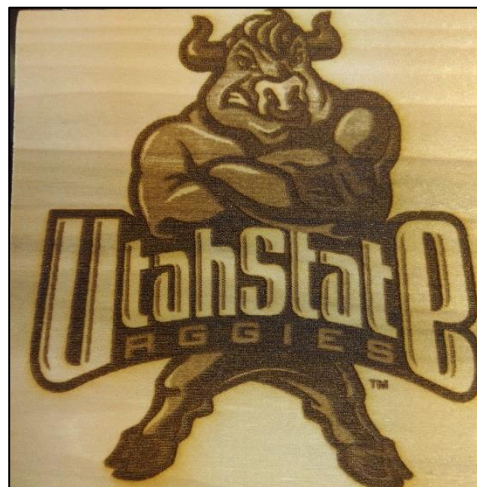


Figure 10. USU Aggies logo on poplar. Good contrast and alignment. Successful engraving.

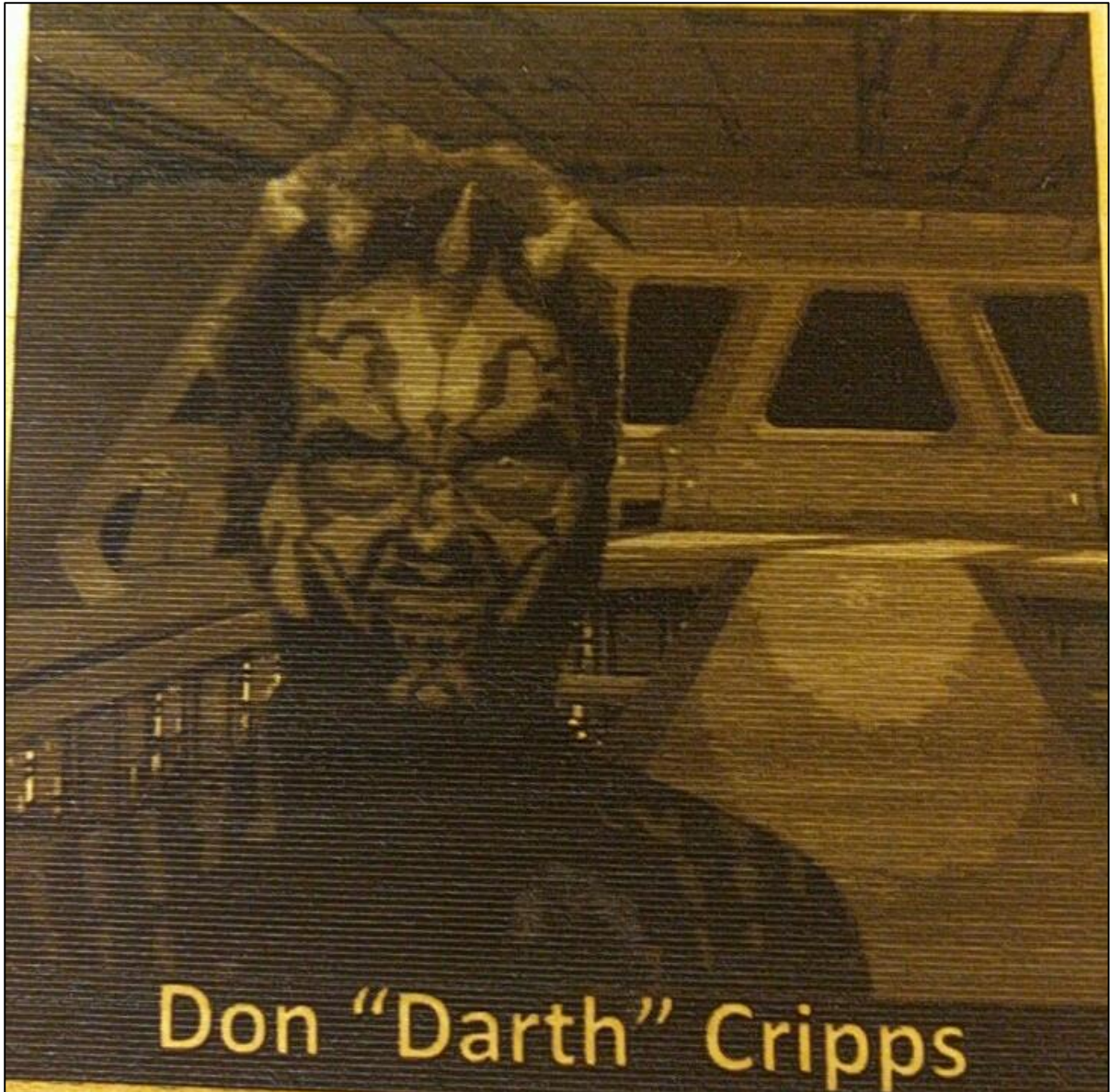


Figure 11. An engraving of Don "Darth" Cripps on the bridge of a star destroyer. Engraved on poplar, good alignment, contrast, and large in size (roughly 5.5 x 5.5 inches).

7. DISCUSSION

The Plaque Laser Engraver Device (PLED) has been successful in achieving our goals. We have written a capable and user friendly software package that simplifies the process of turning an image into 8 shades of grayscale. We have designed an extremely high quality, efficient, and reliable gantry and motion system.

There are significant improvements that can still be made in our system. We could improve the resolution of our images. With a higher number of dots per inch (DPI) we would take longer to engrave, but our images would be clearer. We could adjust the system to produce an 8-bit, 256 shade system. Additionally, the images are well aligned. The only issue in any image is physical alignment of the board on the machine, which we are going to improve by gluing down measuring sticks.

Despite those possible improvements, our system met our initial design goals and we have laid a foundation that will make it easy and possible to make improvements. With sufficient time and effort, all of the weak points in our design, mentioned above, could easily be addressed. A stronger laser could be used, increased engraving time could be permitted. Pixel size definitions, and therefore number of motor steps per pixel could be decreased, allowing for a tighter image.

PLED has been a success, and as it is a system that we are passionate about, we fully intend to continue building and improving the system so that at some point, PLED will be the best laser engraver possible for the price.

8. CONCLUSION

The Plaque Laser Engraver Device (PLED) has met our design goals. It has provided us with a design which can effectively engrave images according to our specifications.

Though we haven't met our initial budgeting goal of \$400, we have generated a design that is affordable for students and hobbyists compared to standard engravers on the market.

We have generated a software package that will either be open source, or available at a very low cost which can effectively process images and generate g-codes.

We have also gained worthwhile experience in program management and organization, as well as in working on a multi-disciplinary team. We have learned to design a system efficiently and then build up that system to achieve our needs. We learned the importance of setting and obtaining funds for a realistic budget. Extra is always better than too little. We learned the value in giving time and resources to the design of a quality mechanical structure. We learned the importance of calibration and of writing readable code that can be easily debugged. It is also important to be willing to look at problems in different ways.

Most importantly for us, we have met our requirements to be able to fulfill our credits for the courses we are enrolled in, and required to graduate from Utah State University.

9. APPENDICES

These appendices provide additional information in regards to the Plaque Laser Engraver Device (PLED).

9.1 Appendix 1 – Budget

The PLED budget is detailed below, with each subsystem budget specified in table 1.

PLED Budget	
Subsystem	Price
Laser System	\$150
Motors System	\$60
Feedback System	\$35
Microcontroller	\$15
Mechanical Structure	\$400
Cabling	\$35
Safety	\$35
Testing Materials	\$40
Total	\$770

Table 1. PLED Budget

9.2 Appendix 2 – Timeline

For the duration of the PLED program, work was divided into three phases, described below in Table 2. Our originally proposed Gantt chart is in figure 12 and the Gantt chart as followed is in figure 13.

PLED Schedule			
Task	Start date	Due date	Duration
Phase 1 - Initial documentation			
Specifications Document (Zach & Casey)	12/1/2015	12/15/2015	14
Proposal (Zach & Casey)	12/1/2015	12/15/2015	14
Select specific components to use (Team)	12/15/2015	1/25/2016	41
Preliminary Design Review Slides (Zach & Casey)	1/1/2016	1/25/2016	24
Preliminary Design Review (Team)	1/27/2016	1/28/2016	1
Phase 2 - Build prototype			
Procure components (Zach & Casey)	1/1/2016	1/30/2016	29
Configure serial communications PC / microcontroller (Zach & Justin)	1/15/2016	2/8/2016	24
Write software to ingest and process image and convert to g-codes (Casey)	1/25/2016	2/8/2016	14
Test components individually (Team)	1/15/2016	2/10/2016	26
Configure communications from microcontroller to motor driver (Zach)	2/9/2016	2/20/2016	11
Design mechanical system (Tate)	1/20/2016	2/29/2016	40
Impliment parser for g-code files (Justin)	2/20/2016	3/15/2016	24
Write software GUI and interface (Casey)	2/21/2016	3/15/2016	23
Procure mechanical system components (Tate & Casey)	3/4/2016	3/15/2016	11
Design and impliment feedback system (Tate, Zach, Casey)	3/5/2016	3/15/2016	10
Mathematically model system (Tate, Zach, Casey)	2/20/2016	3/25/2016	34
Impliment and calibrate laser control system (Zach)	2/21/2016	3/25/2016	33
Integrate G-Code interpreter into microcontroller (Justin)	3/15/2016	3/25/2016	10
Machine Parts (Tate)	3/15/2016	3/31/2016	16
Assemble system (Team)	4/1/2016	4/8/2016	7
Phase 3 - Testing, final documentation, additional features			
Write User's Manual (Zach & Casey)	4/1/2016	4/29/2016	28
Write Final Report (Zach & Casey)	4/1/2016	4/29/2016	28
Testing and Verification (Team)	4/8/2016	4/29/2016	21
Design poster (Zach & Casey)	4/15/2016	4/29/2016	14

Table 2. PLED phase schedule and tasks.

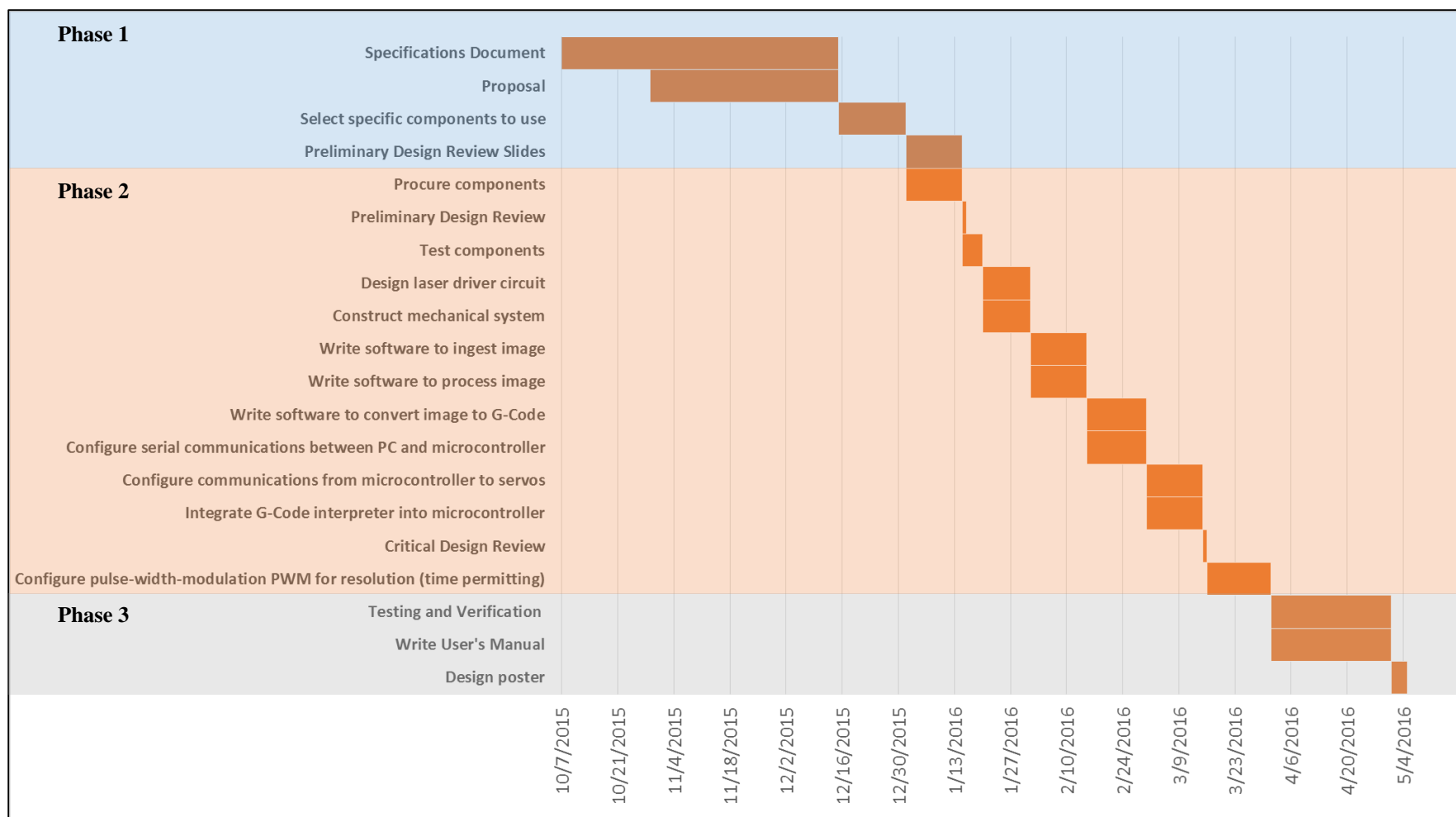


Figure 12. Proposed PLED Gantt Chart.

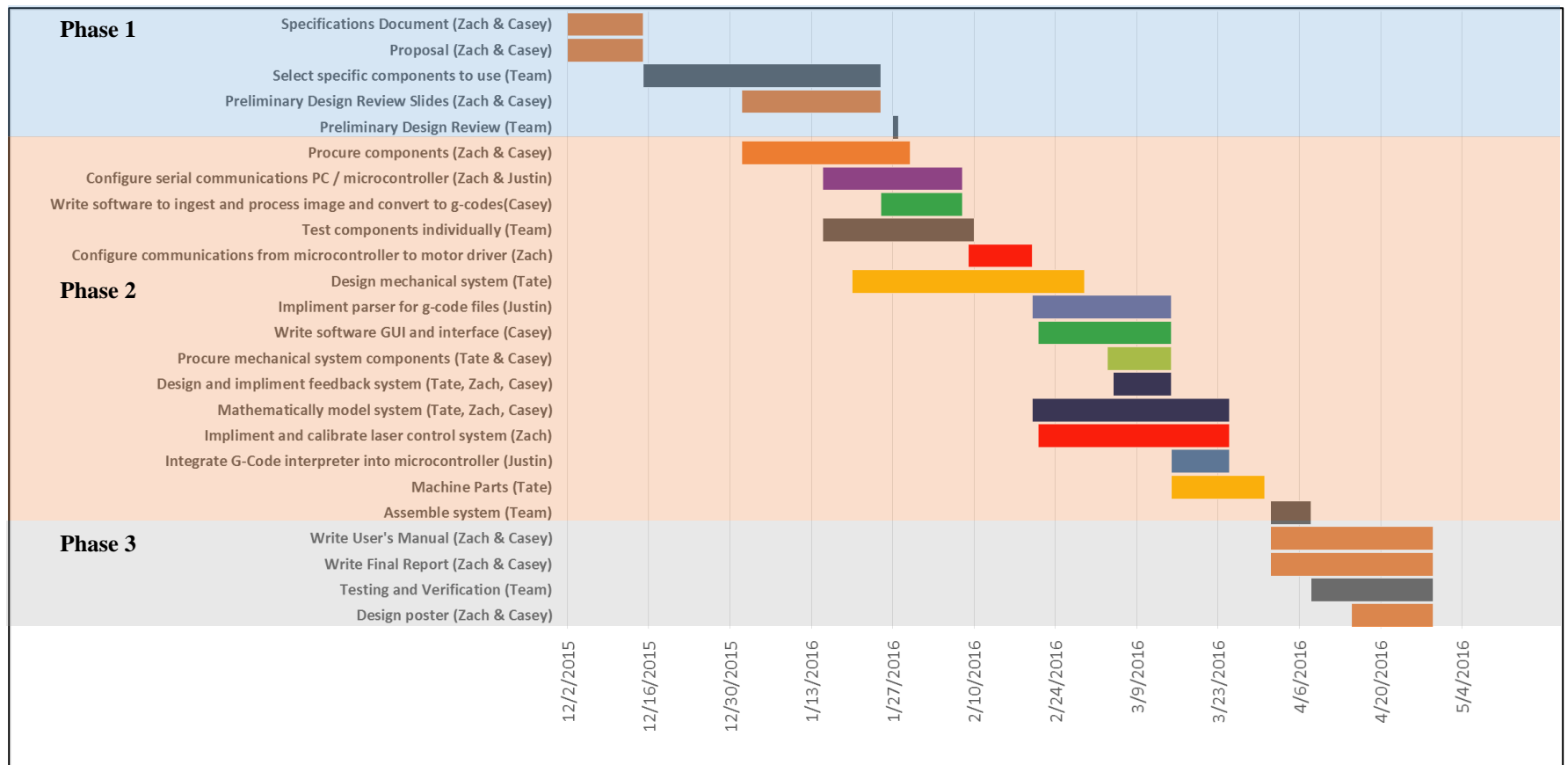


Figure 13. Actual PLED Gantt Chart.

9.3 Appendix 3 – Bill of Materials

This section details each individual component used to build a full PLED system. Table 3 contains the components, a description of them, the quantity used in the design, and its cost.

PLED Bill of Materials				
P/N	Description	Vendor	\$ / Ea	QTY
TTL-232R-3V3	USB UART Adapter	Amazon.com	\$ 19.95	1
DC5-24V	600P/R Rotary Encoder	Amazon.com	\$ 17.45	2
TB6560	Motor Driver	Amazon.com	\$ 14.99	2
Nema 17	Stepper Motors	Amazon.com	\$ 13.99	2
O1 rod	36", 5mm Rod	Amazon.com	\$ 10.78	1
Gdstime 3007	Cooling Fan	Amazon.com	\$ 8.59	1
WYPH IIC I2C	Level Converter 3.3 to TTL	Amazon.com	\$ 5.87	1
SK12	Laser Heat Sink	Amazon.com	\$ 5.15	1
Coupler	5x5mm Coupler	Amazon.com	\$ 5.09	1
TO-220/TO-202	Heat Sink	Amazon.com	\$ 5.00	1
VIO 1.5G	Thermal Grease	Amazon.com	\$ 4.95	1
445 M140 Module	Laser Module	DTR's Laser Shop	\$ 72.00	1
381531274978	LASORB	eBay.com	\$ 11.00	1
Fasteners	Fasteners/washers/bolts	Fastenal	\$ 10.67	1
Spacers	Washers and spacers	Home Depot	\$ 7.35	1
Corner Brace	L-Beam for Frame (4 pk)	Home Depot	\$ 2.84	1
m3-wash	Motor/Encoder Spacers (6 pk)	Home Depot	\$ 0.48	6
m3-0.5x10mm	Motor Fasteners (3 pk)	Home Depot	\$ 0.48	3
FlexMod P3	Laser Driver Heat Sink	Innolasers	\$ 35.99	1
Yard Stick	Straight edge for grid	Lowes	\$ 3.18	1
1185	Mini V Gantry Set	OpenBuilds.org	\$ 28.95	3
230-LP	V-Slot Linear Rail (1500 mm)	OpenBuilds.org	\$ 16.50	2
170-LP	V-Slot Linear Rail (1000 mm)	OpenBuilds.org	\$ 11.00	1
200	GT2 30-Tooth Pulley	OpenBuilds.org	\$ 6.95	5
570	Idler pulley Plate	OpenBuilds.org	\$ 6.95	4
575	Motor mount Plate	OpenBuilds.org	\$ 6.95	2
550	Smooth Idler Pulley	OpenBuilds.org	\$ 5.45	1
50	Tee Nuts (25 Pack)	OpenBuilds.org	\$ 4.95	1
470	GT2 Timing Belt	OpenBuilds.org	\$ 2.50	14
545	L Bracket	OpenBuilds.org	\$ 1.00	8
30	Ball Bearing	OpenBuilds.org	\$ 1.00	1
60	Double Tee Nut	OpenBuilds.org	\$ 0.85	10
730	Belt Clamp	OpenBuilds.org	\$ 0.60	6
115	Low Profile M5 Screws	OpenBuilds.org	\$ 0.15	36

Table 3. PLED BOM

9.4 Appendix 4 – Schematics

Schematics for the Plaque Laser Engraver Device (PLED) appear in this appendix in figure 14.

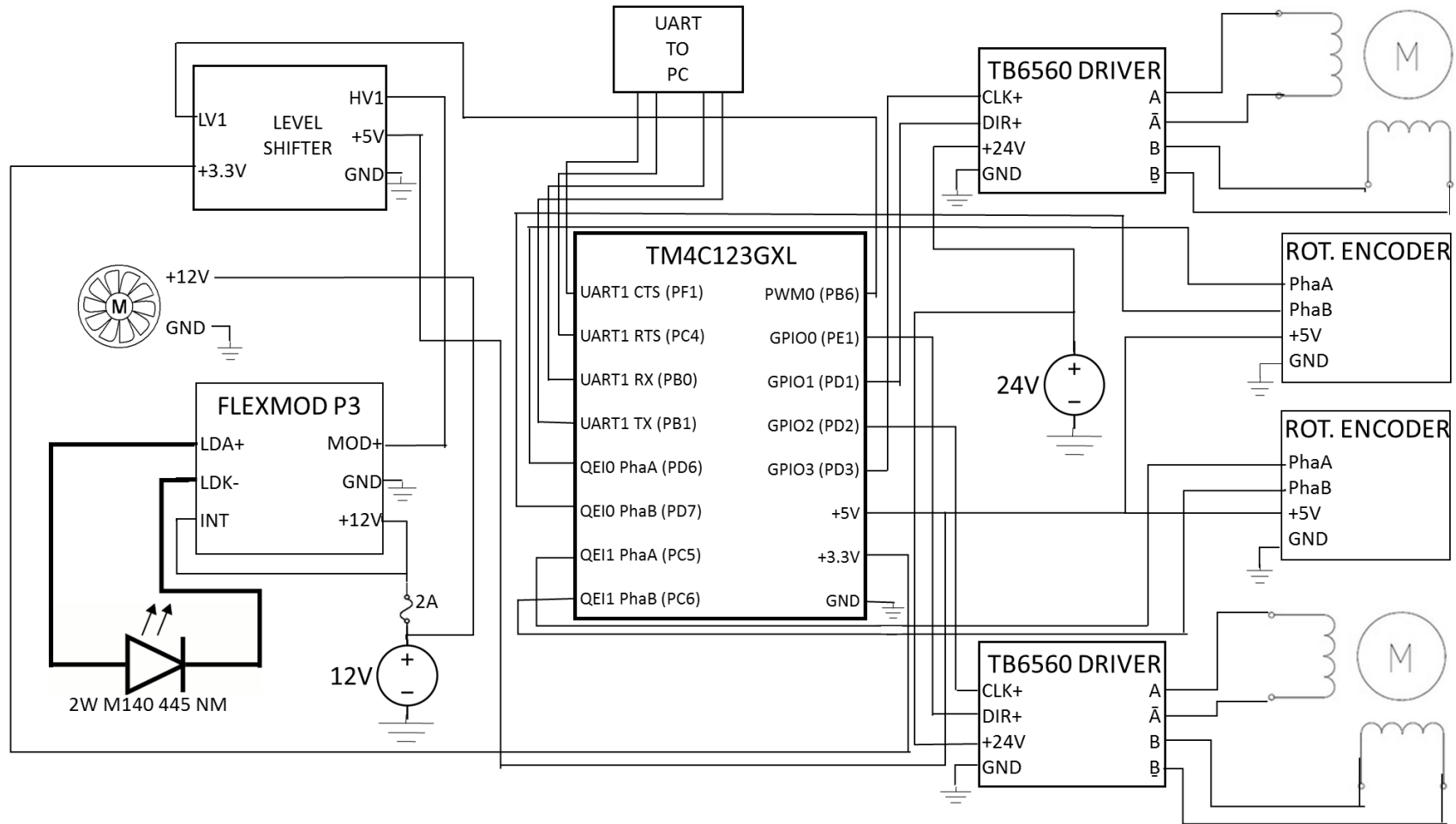


Figure 14. PLED schematic.

9.5 Appendix 5 – Code

Serial and Parser

“main.c”

```
//-----  
//  
//    Code: PC to uController Serial (main.cpp)  
//    Authors: Zachary Garrard, Justin Cox, Casey Wood  
//    Date: 3/10/2016  
//  
//-----  
  
#pragma warning(disable:4996)  
#include <stdio.h>  
#include <stdint.h>  
#include <stdlib.h>  
#include <Windows.h>  
#include "Serial.h"  
#include <string>  
#include <iostream>  
#include <fstream>  
  
#define G0 0x4730  
#define G1 0x4731  
#define G4 0x4734  
#define M2 0x4D32  
#define M4 0x4D34  
#define M5 0x4D35  
#define S0 0x5330  
#define S1 0x5331  
#define S2 0x5332  
#define S3 0x5333  
#define S4 0x5334  
#define S5 0x5335  
#define S6 0x5336  
#define S7 0x5337  
  
COMSTAT comStat;  
DWORD    dwErrors;  
  
void readUart(void);  
void splitShort(uint16_t short2Split, uint8_t &msb, uint8_t &lsb, uint8_t buffer[]);  
bool getRowData(FILE *inFile, uint16_t xCoor[], uint16_t yCoor[], uint16_t gCode[],  
uint16_t pauseP[], uint16_t size);  
void sendRowData(uint16_t xCoor[], uint16_t yCoor[], uint16_t gCode[], uint16_t pauseP[],  
uint16_t size, int jay, uint16_t rows);  
void clearErrs();  
bool checkResend();  
  
//-----  
//    FUNCTION MAIN  
//-----  
  
CSerial serial;  
  
int main(){
```

```

//*****
//      Serial Vars
//*****
int port = 5;//3;
int baudRate = 115200;//9600; //115200
int dispType = 0;
int nBytesSent = 0;

std::cout << "Please insert your UART COM port number: ";
std::cin >> port;

std::cout << "Please insert your UART baud rate (default is 115200): ";
std::cin >> baudRate;

uint8_t buffer[2];

uint8_t shortMSB;
uint8_t shortLSB;
//*****
//      File IO Vars
//*****
FILE *fp;

uint16_t xInit = 0;
uint16_t yInit = 0;
uint16_t pInit = 0;
uint16_t g00 = G0;
uint16_t g4 = G4;

uint8_t garbage[100] = {0};
uint8_t chG;

uint16_t *xCoors;
uint16_t *yCoors;
uint16_t *gCodes;
uint16_t *pauseP;

uint16_t sizeCol = 0;
uint16_t sizeRow = 0;

//-----
//      GET GCODE LOCATION
//-----

//define G-Code location path string and file stream
std::string gcodeloc;
std::ifstream gcloc("C:\\PLED\\PLEDpath.txt");

//get G-Code Location
std::getline(gcloc, gcodeloc);

//close file stream
gcloc.close();

//-----
//      PARSE CODE

```

```

//-----

//convert string to const char
const char * gpath = gcodeLoc.c_str();
fp = fopen(gpath, "r");

if(!fp){
    printf("Error opening file! \n");
    return 0;
}

//Get Size of image for arrays
fscanf(fp, "size %hd,%hd", &sizeCol, &sizeRow);
chG = fgetc(fp);
//printf("SizeX is: %hd and SizeY is: %hd \n", sizeX, sizeY);

//Dynamically set size of arrays
xCoors = new uint16_t [sizeCol];
yCoors = new uint16_t [sizeCol];
pauseP = new uint16_t [sizeCol];
gCodes = new uint16_t [(sizeCol * 5)];

//Get initial position
fscanf(fp, "G00 X%hd Y%hd", &xInit, &yInit);
chG = fgetc(fp);

fscanf(fp, "G04 P0.%hd", &pInit);
chG = fgetc(fp);

//Open Serial Port
if (!serial.Open(port, baudRate)){
    printf("Error opening COM port! \n");
    return 0;
}

for(int j = 0; j < sizeRow + 1; j++)
{
    if(getRowData(fp, xCoors, yCoors, gCodes, pauseP, sizeCol)){
        //for(int i = 0; i < sizeCol; i++){
            //printf("X:%hd Y:%hd \n", xCoors[i], yCoors[i]);
            //printf("P value: %hd \n", pauseP[i]);
        //}

        //-----
        //Send initial values
        //-----

        if(j == 0)
        {
            //      ZZ sizeCol, sizeRow
            buffer[0] = 'Z';
            buffer[1] = 'Z';
            serial.SendData(buffer, 2);
            splitShort(sizeCol, shortMSB, shortLSB, buffer);
            serial.SendData(buffer, 2);
            splitShort(sizeRow, shortMSB, shortLSB, buffer);

```

```

        serial.SendData(buffer, 2);
        // G0, Xinit, Yinit
        splitShort(g00, shortMSB, shortLSB, buffer);
        serial.SendData(buffer, 2);
        splitShort(xInit, shortMSB, shortLSB, buffer);
        serial.SendData(buffer, 2);
        splitShort(yInit, shortMSB, shortLSB, buffer);
        serial.SendData(buffer, 2);
        // G4, Pinit
        splitShort(g4, shortMSB, shortLSB, buffer);
        serial.SendData(buffer, 2);
        splitShort(pInit, shortMSB, shortLSB, buffer);
        serial.SendData(buffer, 2);
    }

    //-----
    //Send row
    //-----
    sendRowData(xCoors, yCoors, gCodes, pauseP, sizeCol,j,sizeRow);
    //check to see if the row was recieved correctly, if not resend, if
so, wait for the GO-ahead to send the next row
    while (checkResend())
    {
        //resend row
        sendRowData(xCoors, yCoors, gCodes, pauseP, sizeCol, j,
sizeRow);
    }

    //Read GO to send next row
    readUart();
    for (int j = 0; j < 240000000; j++)
    {
        int m = 0;
    }
}
else{
    buffer[0] = 'M';
    buffer[1] = '2';
    serial.SendData(buffer, 2);
    clearErrs();
    buffer[0] = 'R';
    buffer[1] = 'D';
    serial.SendData(buffer, 2);
}

} //End of for()

serial.Close();

delete [] xCoors;
delete [] yCoors;
delete [] gCodes;
delete [] pauseP;

return 0;
}

```

```

bool getRowData(FILE *inFile, uint16_t xCoor[], uint16_t yCoor[], uint16_t gCode[],
uint16_t pauseP[], uint16_t size){
    uint8_t firstChar;
    uint8_t secondChar;
    uint8_t thirdChar;

    uint8_t chG;
    uint32_t iPix = 0;
    uint32_t counter = 1;

    for(int i = 0; i < (size * 5); i++){

        firstChar = fgetc(inFile);
        //printf("Character in question: %c \n",firstChar);

        switch(firstChar){
            case 'G':
                secondChar = fgetc(inFile);
                thirdChar = fgetc(inFile);
                if(thirdChar == '1'){
                    fscanf(inFile, " X%hd Y%hd", &xCoor[iPix],
&yCoor[iPix]);
                    //printf("X and Y: %hd %hd \n",xCoor[iPix],
yCoor[iPix]);

                    chG = fgetc(inFile);
                    gCode[i] = G1;
                }
                else if(thirdChar == '4'){
                    fscanf(inFile, " P0.0%hd", &pauseP[iPix]);
                    chG = fgetc(inFile);
                    gCode[i] = G4;
                }
                else{
                    printf("Unkown Gcode with 'G' \n");
                }
                break;
            case 'S':
                secondChar = fgetc(inFile);
                switch(secondChar){
                    case '0':
                        gCode[i] = S0;
                        break;
                    case '1':
                        gCode[i] = S1;
                        break;
                    case '2':
                        gCode[i] = S2;
                        break;
                    case '3':
                        gCode[i] = S3;
                        break;
                    case '4':
                        gCode[i] = S4;
                        break;
                    case '5':
                        gCode[i] = S5;
                        break;
                    case '6':

```

```

        gCode[i] = S6;
        break;
    case '7':
        gCode[i] = S7;
        break;
    default:
        printf("Unkown S code \n");
        break;
} //end of switch
chG = fgetc(inFile);
break;
case 'M':
    secondChar = fgetc(inFile);
    thirdChar = fgetc(inFile);

    if(thirdChar == '4'){
        gCode[i] = M4;
    }
    else if(thirdChar == '5'){
        gCode[i] = M5;
    }
    else if(thirdChar == '2'){
        return false;
    }
    else {
        printf("Unknown M code \n");
    }
    chG = fgetc(inFile);
    break;
default:
    printf("Unknown line from file \n");
    break;
} //end of switch

if(counter == 5){
    iPix++;
    counter = 0;
}

counter++;
} // end of for

return true;
}

void clearErrs()
{
    ClearCommError(serial.m_hIDComDev, &dwErrors, &comStat);
    while ((comStat.fCtsHold))
    {
        //wait until the CTS line says we can send
        ClearCommError(serial.m_hIDComDev, &dwErrors, &comStat);
    }
    return;
}

void sendRowData(uint16_t xCoor[], uint16_t yCoor[], uint16_t gCode[], uint16_t pauseP[],
uint16_t size, int jay, uint16_t rows){

```

```

uint8_t shortMSB;
uint8_t shortLSB;
uint8_t buffer[2];

uint32_t nBytesSent = 0;

uint32_t iPix = 0;

for(int i = 0; i < (size * 5); i = i + 5){
    //Order to send
    /*for (int j = 0; j < 12000000; j++){
        {
            int m = 0;
        }*/
    //    Gcode[i], X[iPix], Y[iPix] <--- G00 X#,Y#
    clearErrs();
    splitShort(gCode[i], shortMSB, shortLSB, buffer);
    serial.SendData(buffer, 2);
    clearErrs();
    splitShort(xCoor[iPix], shortMSB, shortLSB, buffer);
    serial.SendData(buffer, 2);
    clearErrs();
    splitShort(yCoor[iPix], shortMSB, shortLSB, buffer);
    serial.SendData(buffer, 2);
    //    Gcode[i+1] <--- S#
    clearErrs();
    splitShort(gCode[i+1], shortMSB, shortLSB, buffer);
    serial.SendData(buffer, 2);
    //    Gcode[i+2] <--- M04
    clearErrs();
    splitShort(gCode[i+2], shortMSB, shortLSB, buffer);
    serial.SendData(buffer, 2);
    //    Gcode[i+3], P[iPix] <--- G04 P#
    clearErrs();
    splitShort(gCode[i+3], shortMSB, shortLSB, buffer);
    serial.SendData(buffer, 2);
    clearErrs();
    splitShort(pauseP[iPix], shortMSB, shortLSB, buffer);
    serial.SendData(buffer, 2);
    //    Gcode[i+4] <--- M05
    clearErrs();
    splitShort(gCode[i+4], shortMSB, shortLSB, buffer);
    serial.SendData(buffer, 2);

    iPix++;
}

//    RD when row has been sent
clearErrs();
if (j != rows) //only send on all but the last row
{
    buffer[0] = 'R';
    buffer[1] = 'D';
    serial.SendData(buffer, 2);
}

//nBytesSent = serial.SendData(buffer, SIZE);

```

```

}

void splitShort(uint16_t short2Split, uint8_t &msb, uint8_t &lsb, uint8_t buffer[]){
    msb = short2Split >> 8;
    lsb = short2Split & 0x00FF;

    buffer[0] = msb;
    buffer[1] = lsb;
}

void readUart(void){

    int nBytesRead = 0;
    int curT = 0;
    int oldT = 0;
    uint8_t Gflag = 0;
    uint8_t Oflag = 0;

    char readBuffer[1];

    while(Gflag == 0 && Oflag == 0)
    {
        curT = GetTickCount();

        if (curT - oldT > 5)
        {
            nBytesRead = serial.ReadData(readBuffer, sizeof(readBuffer));

            if (nBytesRead > 0)
            {
                for(int i = 0; i < nBytesRead; i++){
                    if(readBuffer[i] == 'G'){
                        Gflag = 1;
                    }
                    else if(readBuffer[i] == 'O'){
                        Oflag = 1;
                    }
                    printf("%c", readBuffer[i]);
                }

            }

            oldT = curT;
        }
    }

    for (int j = 0; j < 50; j++)
    {
        serial.ReadData(readBuffer, sizeof(readBuffer));
    }
    readBuffer[0] = '\0';
}

bool checkResend(){
    //RS=resend MB=muy bueno
    int nBytesRead = 0;

```



```

int curT = 0;
int oldT = 0;
uint8_t Rflag = 0;
uint8_t Sflag = 0;
uint8_t Mflag = 0;
uint8_t Bflag = 0;
bool RS = 0;
bool MB = 0;

char readBuffer[1];

while (RS==false && MB==false)
{
    curT = GetTickCount();

    if (curT - oldT > 5)
    {
        nBytesRead = serial.ReadData(readBuffer, sizeof(readBuffer));

        if (nBytesRead > 0)
        {
            for (int i = 0; i < nBytesRead; i++){
                if (readBuffer[i] == 'R'){
                    Rflag = 1;
                }
                else if (readBuffer[i] == 'S'){
                    Sflag = 1;
                }
                else if (readBuffer[i] == 'M'){
                    Mflag = 1;
                }
                else if (readBuffer[i] == 'B'){
                    Bflag = 1;
                }
                printf("%c", readBuffer[i]);
            }

            oldT = curT;
        }
        if (Rflag == 1 && Sflag == 1)
            RS = true;
        if (Mflag == 1 && Bflag == 1)
            MB = true;
    }
    for (int j = 0; j < 50; j++)
    {
        serial.ReadData(readBuffer, sizeof(readBuffer));
    }
    readBuffer[0] = '\0';
    if (Mflag == 1 && Bflag == 1)
        return false;
    else
        return true;
}

```

Microcontroller

“main.c”

```
//#include "inc\tm4c123gh6pm.h"
#include <stdbool.h>
#include <stdint.h>
#include <stddef.h>
#include <math.h>
//#include <string.h>
#include "driverlib\pwm.h"
#include "inc\hw_memmap.h"
#include "driverlib\gpio.h"
#include "driverlib\pin_map.h"
#include "driverlib\sysctl.h"
#include "inc\hw_ints.h"
#include "driverlib\debug.h"
#include "driverlib\fpu.h"
#include "driverlib\interrupt.h"
#include "driverlib\rom.h"
#include "driverlib\qei.h"
#include "driverlib\uart.h"
#include "inc\hw_gpio.h"
#include "inc\hw_types.h"
#include "driverlib\fpu.h"

#define FEEDBACK                                //<---- Uncomment this for Feedback

volatile unsigned int *UART1=(unsigned int *) 0x4000D000; //This points to the base
address for UART1
int timeEngrave=67; //This is the denominator for the fraction of a second we are
engraving during testing
#define timeRest 0.5 //This is the denominator for the fraction of a second we rest
between burns
#define QEIMaxPosition 0xFFFFFFFF //This is the maximum count for the encoder(s) to
accumulate pulses to.
#define motorStepDuration 8 //This is the denominator for the fraction of a second we wait
between sending stepper motor pulses
#define pulsesPerPixel 12

signed short positiveXPixels=0;    //This keeps track of where we think we are in the +X
direction
signed short positiveYPixels=0;    //This keeps track of where we think we are in the +Y
direction
int encoderPositionX=0;            //EACH PIXEL IS ABOUT 9.1098 ENCODER PULSES - this is
feedback of absolute +X direction position
int encoderPositionY=0;            //EACH PIXEL IS ABOUT 9.135 ENCODER PULSES - this is
feedback of absolute +Y direction position
char engraveReady=0;              //This flag lets us know when we have a full row's worth of
data to engrave

char identifier[2];                //This is a temporary value used to compare incoming G-Codes
for sorting
uint32_t tempPosition;
short moveX;
short moveY;
short dwellTime;
```

```

char numberOfRows=0;
char firstRun=1;

char command[10];
int commandIndex=0;
uint32_t size=0;
short sizeColumns=0;
short sizeRows=0;
int i=0;
int j=0;
char stop[5]="stop"; //currently not used
char go[3]="G0";      //go ahead and send another row
char rs[3]="RS";      //resend row
char mb[3]="MB";      //muy bueno
char done=0;
char start=0;
char readyToGo=0;
char temp;            //used to filter out the null terminating character '\0' in UART
communication
char wellShit=0;
uint32_t ui32Status;
int s;
char rowGood=1;
int openLoopCorrectionCount=0;
char openLoopCountTrigger=0;

int32_t mytest;
uint32_t positionX;
uint32_t positionY;
short xCommands[1612]; //holds the desired x coordinates for an entire row
int xCommandsEnd;      //points to the end of the xCommand data
int xCommandsIndex;    //points to the current x command to be moved to
short yCommands[1612]; //holds the desired y coordinates for an entire row
int yCommandsEnd;      //points to the end of the yCommand data
int yCommandsIndex;    //points to the current y command to be moved to
char gCode[16012];     //enough space to store 2 chars per gcode*(1600 gcodes possible per
row + jog to home command + end of program/row command + initial G04 code)
int gCodeEnd;          //points to the end of the gCode data
int gCodeIndex=0;      //points to the current gCode to be executed
short pauseValues[1612]; //holds the pause durations for the G04 Commands
int pauseValuesEnd=0;  //points to the end of the pauseValues data
int pauseValuesIndex=0; //points to the current pause time duration to dwell
int pixelsCount=0;
int pixelCorrectY=0;
int pixelCorrectX=0;

short burnDurVal = 0;
short zephyr=0;

//*****
//
// The UART interrupt handler.
//
//*****
void
UART1_Handler(void)
{

```

```

//
// Get the interrupt status.
//
ui32Status = UARTIntStatus(UART1_BASE, true);
//
// Clear the asserted interrupts.
//
UARTIntClear(UART1_BASE, ui32Status);
    UARTRxErrorClear(UART1_BASE);
    // Grab the first byte of the identifier that tells us what type of G Code
instruction we are getting
    firstIdentifier:
        temp=UARTCharGet(UART1_BASE);
        if ((0xFF-temp)!=0) //Remove potential null terminating
characters
            identifier[0]=temp;
        else
            goto firstIdentifier;
            //identifier[0]=UARTCharGetNonBlocking(UART1_BASE);
        // Grab the second part of the identifier
    secondIdentifier:
        temp=UARTCharGet(UART1_BASE);
        if ((0xFF-temp)!=0) //Remove potential null terminating
characters
            identifier[1]=temp;
        else
            goto secondIdentifier;
    //Put data in the appropriate arrays
    if (identifier[0]=='Z' && identifier[1]=='Z') //If we recieve the command
to collect the size of the image...
    {
        size=0x00000000;
        sizeRows=0x0000;
        sizeColumns=0x0000;
        i=0;
        while(i<4) //check to see if we have filled up our size integer with
4 bytes, i keeps count
        {
            size|=UARTCharGet(UART1_BASE); //Grab a byte from the fifo
buffer
            if (i==3) //When we know all the parts of the sizing
dimension information...
            {
                sizeRows=size&~0xFFFF0000; //grab the size of the rows
                sizeColumns=size>>16; //grab the size of the
columns
            }
            size=size<<8; //make room for the next byte
            i++; //increment the number of bytes
        }
        i=0; //reset i for next use
        return;
    }

    else if (identifier[0]=='G' && identifier[1]=='0') //If we recieve the
command to jog to a given coordinate...
    {
        tempPosition=0x00000000;

```

```

        moveY=0x0000;
        moveX=0x0000;
        i=0;
        while(i<4) //check to see if we have filled up our size integer with
4 bytes, i keeps count
        {
            tempPosition|=UARTCharGet(UART1_BASE);    //Grab a byte from
the fifo buffer
            if (i==3)    //When we know all the parts of the sizing
dimension information...
            {
                moveY=tempPosition&~0xFFFF0000;    //grab the size of
the rows
                moveX=tempPosition>>16;    //grab the size of the
columns
            }
            tempPosition=tempPosition<<8;    //make room for the next
byte
            i++;    //increment the number of bytes
        }
        if ((moveX>2600||moveX<0) || (moveY>2600||moveY<0))
        {
            rowGood=0;
        }
        xCommands[xCommandsEnd]=moveX;    //Store the x command in
the buffer
        yCommands[yCommandsEnd]=moveY;    //Store the y command in
the buffer
        xCommandsEnd++;    //move the index of the end of the
xCommandBuffer to point to one past the last entry
        yCommandsEnd++;    //move the index of the end of the
yCommandBuffer to point to one past the last entry
        gCode[gCodeEnd]=identifier[0];    //Store the first character
of the G code
        gCode[gCodeEnd+1]=identifier[1];    //Store the second character of
the G code
        gCodeEnd=gCodeEnd+2;    //move the index of the gCode buffer to
point to one past the last entry
        i=0;    //reset i for next use
        return;
    }

    else if (identifier[0]=='G' && identifier[1]=='1')    //If we recieve the
command to interpolate to the next position...
    {
        tempPosition=0x00000000;
        moveX=0x0000;
        moveY=0x0000;
        i=0;
        while(i<4) //check to see if we have filled up our size integer with
4 bytes, i keeps count
        {
            tempPosition|=UARTCharGet(UART1_BASE);    //Grab a byte from
the fifo buffer
            if (i==3)    //When we know all the parts of the sizing
dimension information...
            {

```

```

        moveY=tempPosition&~0xFFFF0000;    //grab the size of
the rows
        moveX=tempPosition>>16;    //grab the size of the
columns
    }
    tempPosition=tempPosition<<8;    //make room for the next
byte
    i++;    //increment the number of bytes
}
if ((moveX>2600||moveX<0) || (moveY>2600||moveY<0))
{
    rowGood=0;
}
xCommands[xCommandsEnd]=moveX;    //Store the x command in
the buffer
yCommands[yCommandsEnd]=moveY;    //Store the y command in
the buffer
xCommandsEnd++;    //move the index of the end of the
xCommandBuffer to point to one past the last entry
yCommandsEnd++;    //move the index of the end of the
yCommandBuffer to point to one past the last entry
gCode[gCodeEnd]=identifier[0];    //Store the first character
of the G code
gCode[gCodeEnd+1]=identifier[1];    //Store the second character of
the G code
gCodeEnd=gCodeEnd+2;    //move the index of the gCode buffer to
point to one past the last entry
i=0;    //reset i for next use
return;
}

else if (identifier[0]=='G' && identifier[1]=='4')    //If we recieve the
command to dwell at the current position...
{
    dwellTime=0x0000;
    dwellTime|=UARTCharGet(UART1_BASE);    //Grab a byte from the fifo
buffer
    dwellTime=dwellTime<<8;    //make room for the next byte
    dwellTime|=UARTCharGet(UART1_BASE);    //Grab the next byte from
the fifo buffer
    if ((dwellTime<0 || dwellTime>100000))
    {
        rowGood=0;
    }
    pauseValues[pauseValuesEnd]=dwellTime;
    pauseValuesEnd++;
    gCode[gCodeEnd]=identifier[0];    //Store the first character
of the G code
    gCode[gCodeEnd+1]=identifier[1];    //Store the second character of
the G code
    gCodeEnd=gCodeEnd+2;    //move the index of the gCode buffer to
point to one past the last entry
    return;
}
else if (identifier[0]=='M' && identifier[1]=='4')    //If we recieve the
command to turn the laser on...
{

```

```

                                gCode[gCodeEnd]=identifier[0];           //Store the first character
of the G code
                                gCode[gCodeEnd+1]=identifier[1]; //Store the second character of
the G code
                                gCodeEnd=gCodeEnd+2;           //move the index of the gCode buffer to
point to one past the last entry
                                return;
                                }

                                else if (identifier[0]=='M' && identifier[1]=='5') //If we recieve the
command to turn the laser off...
                                {
of the G code
                                gCode[gCodeEnd]=identifier[0];           //Store the first character
the G code
                                gCode[gCodeEnd+1]=identifier[1]; //Store the second character of
point to one past the last entry
                                gCodeEnd=gCodeEnd+2;           //move the index of the gCode buffer to
                                return;
                                }

                                else if (identifier[0]=='S' && identifier[1]=='0') //If we recieve the
command to set the laser at full intensity...
                                {
of the G code
                                gCode[gCodeEnd]=identifier[0];           //Store the first character
the G code
                                gCode[gCodeEnd+1]=identifier[1]; //Store the second character of
point to one past the last entry
                                gCodeEnd=gCodeEnd+2;           //move the index of the gCode buffer to
                                return;
                                }

                                else if (identifier[0]=='S' && identifier[1]=='1') //If we recieve the
command to set the laser 6/7 intensity...
                                {
of the G code
                                gCode[gCodeEnd]=identifier[0];           //Store the first character
the G code
                                gCode[gCodeEnd+1]=identifier[1]; //Store the second character of
point to one past the last entry
                                gCodeEnd=gCodeEnd+2;           //move the index of the gCode buffer to
                                return;
                                }

                                else if (identifier[0]=='S' && identifier[1]=='2') //If we recieve the
command to set the laser at 5/7 intensity...
                                {
of the G code
                                gCode[gCodeEnd]=identifier[0];           //Store the first character
the G code
                                gCode[gCodeEnd+1]=identifier[1]; //Store the second character of
point to one past the last entry
                                gCodeEnd=gCodeEnd+2;           //move the index of the gCode buffer to
                                return;
                                }

```

```

        else if (identifier[0]=='S' && identifier[1]=='3')    //If we recieve the
command to set the laser at 4/7 intensity...
        {
            gCode[gCodeEnd]=identifier[0];                //Store the first character
of the G code
            gCode[gCodeEnd+1]=identifier[1];    //Store the second character of
the G code
            gCodeEnd=gCodeEnd+2;                //move the index of the gCode buffer to
point to one past the last entry
            return;
        }

        else if (identifier[0]=='S' && identifier[1]=='4')    //If we recieve the
command to set the laser at 3/7 intensity...
        {
            gCode[gCodeEnd]=identifier[0];                //Store the first character
of the G code
            gCode[gCodeEnd+1]=identifier[1];    //Store the second character of
the G code
            gCodeEnd=gCodeEnd+2;                //move the index of the gCode buffer to
point to one past the last entry
            return;
        }

        else if (identifier[0]=='S' && identifier[1]=='5')    //If we recieve the
command to set the laser at 2/7 intensity...
        {
            gCode[gCodeEnd]=identifier[0];                //Store the first character
of the G code
            gCode[gCodeEnd+1]=identifier[1];    //Store the second character of
the G code
            gCodeEnd=gCodeEnd+2;                //move the index of the gCode buffer to
point to one past the last entry
            return;
        }

        else if (identifier[0]=='S' && identifier[1]=='6')    //If we recieve the
command to set the laser at 1/7 intensity...
        {
            gCode[gCodeEnd]=identifier[0];                //Store the first character
of the G code
            gCode[gCodeEnd+1]=identifier[1];    //Store the second character of
the G code
            gCodeEnd=gCodeEnd+2;                //move the index of the gCode buffer to
point to one past the last entry
            return;
        }

        else if (identifier[0]=='S' && identifier[1]=='7')    //If we recieve the
command to set the laser at no power...
        {
            gCode[gCodeEnd]=identifier[0];                //Store the first character
of the G code
            gCode[gCodeEnd+1]=identifier[1];    //Store the second character of
the G code
            gCodeEnd=gCodeEnd+2;                //move the index of the gCode buffer to
point to one past the last entry
            return;
        }

```



```

    }

    else if (identifier[0]=='M' && identifier[1]=='2')           //If we
recieve the command that the picture is complete...
    {
        gCode[gCodeEnd]=identifier[0];           //Store the first character
of the G code
        gCode[gCodeEnd+1]=identifier[1]; //Store the second character of
the G code
        gCodeEnd=gCodeEnd+2;           //move the index of the gCode buffer to
point to one past the last entry
        return;
    }
    else if (identifier[0]=='R' && identifier[1]=='D')           //If we
recieve the command that the picture is complete...
    {
        gCode[gCodeEnd]=identifier[0];           //Store the first character
of the G code
        gCode[gCodeEnd+1]=identifier[1]; //Store the second character of
the G code
        numberOfRows++;
        gCodeEnd=gCodeEnd+2;           //move the index of the gCode buffer to
point to one past the last entry
        readyToGo = 1;
        SysCtlDelay(SysCtlClockGet()*20/3000); //wait 20 milliseconds
        if (rowGood==1)
        {
            for (i=0;i<2;i++)
            {
                UARTCharPut(UART1_BASE,mb[i]);
                SysCtlDelay(SysCtlClockGet()*20/3000); //wait 20
milliseconds
            }
        }
        return;
    }
    else
    {
        rowGood=0;
    }
}

void Sys_Clock_Set()
{
    int i;
    //
    // Set the clocking to run directly from the external crystal/oscillator.
    //
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_INT |
SYSCTL_XTAL_16MHZ);
    for (i=0;i<2000;i++)
    {
        //Wait while the clock frequency change settles
    }
    //
    // Can be used to check the clock frequency
    //
    mytest=SysCtlClockGet();
}

```

```

        return;
    }

void PWM_Setup()
{
    //
    // Set the PWM clock to the system clock.
    //
    SysCtlPWMClockSet(SYSCTL_PWMDIV_1);
    //
    // The PWM peripheral must be enabled for use.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
    //
    // PWM0 is used with PB6 and PB7.
    // GPIO port B needs to be enabled so these pins can be used.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    //
    // Configure the GPIO pin muxing to select PWM00/PWM01 functions for these pins.
    // This step selects which alternate function is available for these pins.
    //
    GPIOPinConfigure(GPIO_PB6_M0PWM0);
    // | GPIO_PB7_M0PWM1
    //
    // Configure the PWM function for these pins.
    //
    GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_6);
    //
    // Configure the PWM generator for count down mode with immediate updates
    // to the parameters.
    //
    PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
    //
    // Set the PWM period to 250Hz. To calculate the appropriate parameter
    // use the following equation:  $N = (1 / f) * SysClk$ . Where N is the
    // function parameter, f is the desired frequency, and SysClk is the
    // system clock frequency.
    // In this case you get:  $(1 / 250\text{Hz}) * 16\text{MHz} = 64000$  cycles. Note that
    // the maximum period you can set is  $2^{16}$ .
    //
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, 480); //3813
    //
    // Set PWM0 to a duty cycle of 25%. You set the duty cycle as a function
    // of the period. Since the period was set above, you can use the
    // PWMGenPeriodGet() function. For this example the PWM will be high for
    // 25% of the time or 16000 clock ticks ( $64000 / 4$ ).
    //
    PWMPPulseWidthSet(PWM0_BASE, PWM_OUT_0, 1); //1907 //
    // Enable the PWM0 Bit0 (PB6) and PWM1 Bit0 (PB7) output.
    PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT, true);
    //
    // Enable the PWM generator block.
    //
    PWMGenEnable(PWM0_BASE, PWM_GEN_0);
    return;
}

```

```

void UART1_Setup()
{
    //
    // Enable the GPIO port that is used for the on-board LED. (and UART1 CTS)
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    //
    // Enable the GPIO pins for the LED (PF2).
    //
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);
    //
    // Enable the peripherals for the LED and UART1.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
    //
    // Enable processor interrupts.
    //
    IntMasterEnable();
    //
    // Set GPIO PB0 and PB1 as RX and TX on UART1, and
    // PC4 and PF1 as RTS and CTS for UART1
    //
    GPIOPinConfigure(GPIO_PB0_U1RX);
    GPIOPinConfigure(GPIO_PB1_U1TX);
    GPIOPinConfigure(GPIO_PF1_U1CTS);
    GPIOPinConfigure(GPIO_PC4_U1RTS);
    GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_4);
    GPIOPinTypeUART(GPIO_PORTF_BASE, GPIO_PIN_1);
    //
    // Configure the UART for 9600 8-N-1 operation.
    //
    UARTClockSourceSet(UART1_BASE, UART_CLOCK_PIOSC);

    UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 115200,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
UART_CONFIG_PAR_NONE));
    //
    // Configure the UART for interrupts on < 7/8 of the TX buffer empty and > 1/2 RX
full.
    //
    UARTFIFOEnable(UART1_BASE);
    UARTFIFOLevelSet(UART1_BASE, UART_FIFO_TX7_8, UART_FIFO_RX4_8);
    //
    // Configure the UART to use RTS and CTS handshaking.
    //
    UARTFlowControlSet(UART1_BASE, UART_FLOWCONTROL_RX | UART_FLOWCONTROL_TX);
    //
    // Enable the UART interrupt.
    //
    UARTFIFOEnable(UART1_BASE);
    IntEnable(INT_UART1);
    UARTIntEnable(UART1_BASE, UART_INT_RX | UART_INT_RT);
    UARTEnable(UART1_BASE);
}

```

```

}

void QEI_Setup()
{
    //
    // QEI0 is used with PC5 and PC6 and QEI1 is used with PD6 and PD7.
    // GPIO ports C and D need to be enabled so these pins can be used.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC); //already done in UART1 setup
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    //
    // Enable the peripherals for QEI0 and QEI1.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_QEI0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_QEI1);
    //
    // Unlock GPIO D7
    //
    HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTD_BASE + GPIO_O_CR) |= 0x80;
    HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = 0;
    //
    // Set GPIO PD6 and PD7 as PhA and PhB on QEI0, and
    // PC5 and PC6 as PhA and PhB for QEI1
    //
    GPIOPinConfigure(GPIO_PD6_PHA0);
    GPIOPinConfigure(GPIO_PD7_PHB0);
    GPIOPinConfigure(GPIO_PC5_PHA1);
    GPIOPinConfigure(GPIO_PC6_PHB1);
    GPIOPinTypeQEI(GPIO_PORTD_BASE, GPIO_PIN_6 | GPIO_PIN_7);
    GPIOPinTypeQEI(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_6);
    //
    // Disable the encoders before configuration
    //
    QEIDisable(QEI0_BASE);
    QEIDisable(QEI1_BASE);
    //
    // Configure the UART for 9600 8-N-1 operation.
    //
    QEIConfigure(QEI0_BASE, QEI_CONFIG_CAPTURE_A_B | QEI_CONFIG_NO_RESET |
QEI_CONFIG_QUADRATURE | QEI_CONFIG_NO_SWAP, QEIMaxPosition);
    QEIConfigure(QEI1_BASE, QEI_CONFIG_CAPTURE_A_B | QEI_CONFIG_NO_RESET |
QEI_CONFIG_QUADRATURE | QEI_CONFIG_NO_SWAP, QEIMaxPosition);
    //
    // Enable the encoders
    //
    QEIEnable(QEI0_BASE);
    QEIEnable(QEI1_BASE);
}

void GPIO_Setup()
{
    //
    // GPIO PD0:PD1 are used for toggling the forward and reverse step on the stepper
motor drivers.
    // GPIO PD2:PD3 are used for sending the step signal to the stepper motor drivers.
    // Enable port D so these pins can be used.
    //

```

```

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
//
// Enable the GPIO pins for controlling the stepper motors (PD0:PD3).
//
GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE,GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE,GPIO_PIN_1);
//
// Set the pins as open drain
//
GPIOPadConfigSet(GPIO_PORTD_BASE,GPIO_PIN_1 | GPIO_PIN_2 |
GPIO_PIN_3,GPIO_STRENGTH_8MA,GPIO_PIN_TYPE_OD);
GPIOPadConfigSet(GPIO_PORTE_BASE,GPIO_PIN_1,GPIO_STRENGTH_8MA,GPIO_PIN_TYPE_OD);
}

void dwell(short dwellDuration) //USED WITH the initial G04
{
    //THIS FUNCTION CAUSES THE LASER TO DWELL IN ITS CURRENT STATE FOR THE G04 COMMAND
    SysCtlDelay((int) ((SysCtlClockGet()*dwellDuration)/3000.0));
    return;
}

void correctPlacement(short curPosX,short curPosY)
{
    //CurPosX is where we THINK we are, based on the number of positive pixels sent
    out
    //Assign positiveXPixels to curPosX when calling step()
    //positiveXPixels is also where we THOUGHT we were, but gets updated to match
    where we REALLY are in this function
    //encoderPositionX is where we REALLY are
    //There are 9.1098 average x counts per pixel from the encoder and 9.135 average y
    counts per pixel from the encoder.
    #ifdef FEEDBACK
    encoderPositionX=QEIPositionGet(QEI0_BASE);
    while (curPosX*8.8938-encoderPositionX>20) //if we are more than 2 pixels
    left of where we need to be, fix it before moving on
    {
        //set the X axis stepper motor direction to forward
        GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, 0);

        //set the clock output high - the motor steps on rising clock edges
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_PIN_2);
        //wait a certain amount of time before changing the stepper motor clock
    state
        SysCtlDelay(SysCtlClockGet()*8 / (motorStepDuration * 6000)); //if we are
    just jogging then go ahead and move fast
        //Set the clock output low
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
        //wait a certain amount of time before changing the stepper motor clock
    state
        SysCtlDelay(SysCtlClockGet()*8 / (motorStepDuration * 6000)); //if we are
    just jogging then go ahead and move fast

        //Recalibrate the number of steps actually sent to match the actual number
    of steps recorded by the encoder
        encoderPositionX=QEIPositionGet(QEI0_BASE);
        encoderPositionX=QEIPositionGet(QEI0_BASE); //grab the current encoder
    position
    }
}

```

```

    while (encoderPositionX-curPosX*8.8938>20)          //if we are more than 2
pixels right of where we need to be, fix it before moving on
    {
        //set the X axis stepper motor direction to reverse
        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_1, GPIO_PIN_1);

        //set the clock output high - the motor steps on rising clock edges
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_PIN_2);
        //wait a certain amount of time before changing the stepper motor clock
state
        SysCtlDelay(SysCtlClockGet()*8 / (motorStepDuration * 6000)); //if we are
just jogging then go ahead and move fast
        //Set the clock output low
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
        //wait a certain amount of time before changing the stepper motor clock
state
        SysCtlDelay(SysCtlClockGet()*8 / (motorStepDuration * 6000)); //if we are
just jogging then go ahead and move fast

        //Recalibrate the number of steps actually sent to match the actual number
of steps recorded by the encoder
        encoderPositionX=QEIPositionGet(QEI0_BASE);    //grab the current encoder
position
    }
    encoderPositionY=QEIPositionGet(QEI1_BASE);    //Now see if the Y alignment is
correct

    while (curPosY*8.83568-encoderPositionY>20)        //if we are more than 2
pixels above where we need to be, fix it before moving on
    { //8.83568
        //set the Y axis stepper motor direction to forward
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1, 0);

        //set the clock output high - the motor steps on rising clock edges
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, GPIO_PIN_3);
        //wait a certain amount of time before changing the stepper motor clock
state
        SysCtlDelay(SysCtlClockGet()*8 / (motorStepDuration * 6000)); //if we are
just jogging then go ahead and move fast
        //Set the clock output low
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, 0);
        //wait a certain amount of time before changing the stepper motor clock
state
        SysCtlDelay(SysCtlClockGet()*8 / (motorStepDuration * 6000)); //if we are
just jogging then go ahead and move fast

        //Recalibrate the number of steps actually sent to match the actual number
of steps recorded by the encoder
        encoderPositionY=QEIPositionGet(QEI1_BASE);    //
        encoderPositionY=QEIPositionGet(QEI1_BASE);    //Now see if the Y
alignment is correct
    }

    while (encoderPositionY-curPosY*8.83568>20)        //if we are more than 2
pixels below where we need to be, fix it before moving on
    { //8.83568
        //set the Y axis stepper motor direction to reverse

```

```

        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1, GPIO_PIN_1);
    {
        //set the clock output high - the motor steps on rising clock
edges
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, GPIO_PIN_3);
        //wait a certain amount of time before changing the stepper
motor clock state
        SysCtlDelay(SysCtlClockGet()*8 / (motorStepDuration * 6000));
        //if we are just jogging then go ahead and move fast
        //Set the clock output low
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, 0);
        //wait a certain amount of time before changing the stepper
motor clock state
        SysCtlDelay(SysCtlClockGet()*8 / (motorStepDuration * 6000));
        //if we are just jogging then go ahead and move fast
    }

    //Recalibrate the number of steps actually sent to match the actual number
of steps recorded by the encoder
    encoderPositionY=QEIPositionGet(QEI1_BASE);    //Now see if the Y
alignment is correct
    }

    curPosX=positiveXPixels;
    curPosY=positiveYPixels;

    #endif
}
void step(short curPosX,short curPosY,short desPosX,short desPosY, short burnDuration)
{
    //The step duration is the count of 16000000*(0.001*burnDuration)/((3 assembly
instructions per SysCtlDelay)*pulsesPerPixel)
    //this way we have the laser ON while we microstep across the width of a pixel so we dont
have to pause to engrave
    if (burnDuration!=0)
        openLoopCountTrigger=1;
    while (desPosX-curPosX>0)    //if we need to move in the +X direction...
    {
        pixelsCount++;
        //set the stepper motor direction to forward
        GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, 0);

        for(i = 0; i < (pulsesPerPixel); i++)    //Move one full pixel width
        {
            //set the clock output high - the motor steps on rising clock
edges
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_PIN_2);
            //wait a certain amount of time before changing the stepper
motor clock state
            if (burnDuration!=0) //if we are burning and not just moving

            SysCtlDelay((int)((SysCtlClockGet()*burnDuration)/(6000*pulsesPerPixel)));
            //pulse high for half the duration of 1/12th of the pixel width
            else

            SysCtlDelay((int)(SysCtlClockGet()/(motorStepDuration*6000))); //if we are just
jogging then go ahead and move fast
            //Set the clock output low

```

```

        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
        //wait a certain amount of time before changing the stepper
motor clock state
        if (burnDuration!=0) //if we are burning and not just moving

        SysCtlDelay((int)((SysCtlClockGet()*burnDuration)/(6000*pulsesPerPixel)));
        //pulse high for half the duration of 1/12th of the pixel width
        else

        SysCtlDelay((int)(SysCtlClockGet()/(motorStepDuration*6000))); //if we are just
jogging then go ahead and move fast
    }
    positiveXPixels++; //increment the number of x pixels moves in the +x
direction
    curPosX++; //increment the temporary count of the number of x pixels
moves in the +x direction
    encoderPositionX=QEIPositionGet(QEI0_BASE); //grab the current
encoder position
    }
    while (curPosX-desPosX>0) //if we got the command to move in the -X direction...
    {
        pixelsCount++;
        //set the stepper motor direction to reverse
        GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, GPIO_PIN_1);

        for(i = 0; i < (pulsesPerPixel); i++) //Move one full pixel width
        {
            //set the clock output high - the motor steps on rising clock
edges
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_PIN_2);
            //wait a certain amount of time before changing the stepper
motor clock state
            if (burnDuration!=0) //if we are burning and not just moving

            SysCtlDelay((int)((SysCtlClockGet()*burnDuration)/(6000*pulsesPerPixel)));
            //pulse high for half the duration of 1/12th of the pixel width
            else

            SysCtlDelay((int)(SysCtlClockGet()/(motorStepDuration*6000))); //if we are just
jogging then go ahead and move fast
            //Set the clock output low
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
            //wait a certain amount of time before changing the stepper
motor clock state
            if (burnDuration!=0) //if we are burning and not just moving

            SysCtlDelay((int)((SysCtlClockGet()*burnDuration)/(6000*pulsesPerPixel)));
            //pulse high for half the duration of 1/12th of the pixel width
            else

            SysCtlDelay((int)(SysCtlClockGet()/(motorStepDuration*6000))); //if we are just
jogging then go ahead and move fast
        }
        positiveXPixels--; //decrement the number of of x pixels moves in the +x
direction
        curPosX--; //the temporary count of the number of x pixels moves in the
+x direction

```



```

        encoderPositionX=QEIPositionGet(QEI0_BASE);        //Grab the current encoder
position
    }
    while (desPosY-curPosY>0)    //if we got the command to move in the +Y direction...
    {
        //set the stepper motor direction to forward
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1, 0);
        //SysCtlDelay((int)(SysCtlClockGet()/(8000)));

        for(i = 0; i < (pulsesPerPixel); i++)    //Move one full pixel width
        {
            //set the clock output high - the motor steps on rising clock
edges
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, GPIO_PIN_3);
            //wait a certain amount of time before changing the stepper
motor clock state
            if (burnDuration!=0) //if we are burning and not just moving

            SysCtlDelay((int)((SysCtlClockGet()*burnDuration)/(6000*pulsesPerPixel)));
            //pulse high for half the duration of 1/12th of the pixel width
            else

            SysCtlDelay((int)(SysCtlClockGet()/(motorStepDuration*6000))); //if we are just
jogging then go ahead and move fast
            //Set the clock output low
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, 0);
            //wait a certain amount of time before changing the stepper
motor clock state
            if (burnDuration!=0) //if we are burning and not just moving

            SysCtlDelay((int)((SysCtlClockGet()*burnDuration)/(6000*pulsesPerPixel)));
            //pulse high for half the duration of 1/12th of the pixel width
            else

            SysCtlDelay((int)(SysCtlClockGet()/(motorStepDuration*6000))); //if we are just
jogging then go ahead and move fast
            if (openLoopCountTrigger==1)
                openLoopCorrectionCount++;
            if (openLoopCorrectionCount%26==0)
            {
                //set the X stepper motor direction to reverse
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1, GPIO_PIN_1);
                //set the clock output high - the motor steps on rising
clock edges
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_PIN_2);

                SysCtlDelay((int)(SysCtlClockGet()/(motorStepDuration*6000))); //if we are just
jogging then go ahead and move fast
                //Set the clock output low
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);

                SysCtlDelay((int)(SysCtlClockGet()/(motorStepDuration*6000))); //if we are just
jogging then go ahead and move fast
            }
        }
        positiveYPixels++;    //increment the number of y pixels moves in the +y
direction

```

```

        curPosY++;    //increment the temporary count of the number of y pixels
moves in the +y direction
        encoderPositionY=QEIPositionGet(QEI1_BASE);    //Grab the current encoder
position
    }
    while (curPosY-desPosY>0)    //if we got the command to move in the -Y direction...
    {
        //set the stepper motor direction to reverse
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1, GPIO_PIN_1);

        for(i = 0; i < (pulsesPerPixel); i++)    //Move one full pixel width
        {
            //set the clock output high - the motor steps on rising clock
edges
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, GPIO_PIN_3);
            //wait a certain amount of time before changing the stepper
motor clock state
            if (burnDuration!=0) //if we are burning and not just moving

            SysCtlDelay((int)((SysCtlClockGet()*burnDuration)/(6000*pulsesPerPixel)));
            //pulse high for half the duration of 1/12th of the pixel width
            else

            SysCtlDelay((int)(SysCtlClockGet()/(motorStepDuration*6000))); //if we are just
jogging then go ahead and move fast
            //Set the clock output low
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, 0);
            //wait a certain amount of time before changing the stepper
motor clock state
            if (burnDuration!=0) //if we are burning and not just moving

            SysCtlDelay((int)((SysCtlClockGet()*burnDuration)/(6000*pulsesPerPixel)));
            //pulse high for half the duration of 1/12th of the pixel width
            else

            SysCtlDelay((int)(SysCtlClockGet()/(motorStepDuration*6000))); //if we are just
jogging then go ahead and move fast
        }

        positiveYPixels--;    //decrement the number of y pixels moves in the +y
direction
        curPosY--;    //decrement the temporary count of the number of y pixels
moves in the +y direction
        encoderPositionY=QEIPositionGet(QEI1_BASE);    //Grab the current encoder
position
    }
    i=0;
}
void engrave()
{
    int j=0;    //reset the index for the row engraving
    while (j < gCodeEnd) //engrave a whole row
    {
        if (gCode[j]=='G' && gCode[j+1]=='0')    //If we got the
command to jog to a given position
        {
            step(0, 0, xCommands[xCommandsIndex],
yCommands[yCommandsIndex], 0);    //move to the initial position

```

```

        correctPlacement(positiveXPixels, positiveYPixels);
//Use the encoders to make sure we jogged to the correct location
        xCommandsIndex++; //move the pointer to the next x
location
        yCommandsIndex++; //move the pointer to the next y
location
        j+=4; //point to the first pixel
        dwell(pauseValues[pauseValuesIndex]); //dwell the
amount of time indicated by the first G4
        pauseValuesIndex++; //update the pause value index
        pixelsCount=0;
    }
    else if (gCode[j]=='M' && gCode[j+1]=='2') //If we got
the command to jog to the origin (end of picture)
    {
        step(positiveXPixels, positiveYPixels, 0, 0, 0); //move
back to the origin
        SysCtlDelay(SysCtlClockGet()*20/3000);
    }
    else if (gCode[j]=='G' && gCode[j+1]=='1') //If we get a move
command...
    {
        if(gCode[j+2]=='S') //first get the intensity and burn
duration
        {
            switch(gCode[j+3]) //get the intensity level
            and turn on the laser at the specified intensity
            {
                case '0':
                    PWMPulseWidthSet(PWM0_BASE,
PWM_OUT_0,479);

                    zephyr=60;//60
                    break;
                case '1':
                    PWMPulseWidthSet(PWM0_BASE,
PWM_OUT_0,479);//410

                    zephyr=41;//48
                    break;
                case '2':
                    PWMPulseWidthSet(PWM0_BASE,
PWM_OUT_0,479);//353

                    zephyr=32;//40
                    break;
                case '3':
                    PWMPulseWidthSet(PWM0_BASE,
PWM_OUT_0,479);//300

                    zephyr=24;//42
                    break;
                case '4':
                    PWMPulseWidthSet(PWM0_BASE,
PWM_OUT_0,479);//220

                    zephyr=18;//40
                    break;
                case '5':
                    PWMPulseWidthSet(PWM0_BASE,
PWM_OUT_0,479);//163

                    zephyr=14;//42
                    break;
            }
        }
    }

```

```

        case '6':
            PWMPulseWidthSet(PWM0_BASE,
PWM_OUT_0,479); //135

            zephyr=12; //42
            break;
        case '7':
            PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,1);
            zephyr=10;
            break;
        default:
            //DO NOTHING
            break;
    } //end of switch
} //end of if
j+=6; //go find the G4 for this pixel
}
else if (gCode[j]=='G' && gCode[j+1]=='4') //when we
know the burn duration...
{
    burnDurVal=pauseValues[pauseValuesIndex]; //grab the
burn duration from pauseValues
    pauseValuesIndex++; //move the pointer for the next
pause value
    step(positiveXPixels, positiveYPixels,
xCommands[xCommandsIndex], yCommands[yCommandsIndex], zephyr); //move over the pixel
while burning it at the specified intensity
    xCommandsIndex++; //move the pointer to the next x
location
    yCommandsIndex++; //move the pointer to the next y
location
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,1); //turn the
laser off
    j+=4; //point to the start of the next gCode pixel
}
else if (gCode[j]=='R' && gCode[j+1]=='D')
{
    break;
}
} //end of while
pixelsCount=0;
pixelCorrectX=0;
pixelCorrectY=0;
for (i=0;i<1612;i++)
{
    xCommands[i]='\0';
    yCommands[i]='\0';
    pauseValues[i]='\0';
}
for (i=0;i<16012;i++)
{
    gCode[i]='\0';
}
for (i=0;i<16;i++)
{
    command[0]=UARTCharGetNonBlocking(UART1_BASE);
}
UARTRxErrorClear(UART1_BASE);
xCommandsEnd=0;

```

```

        xCommandsIndex=0;
        yCommandsEnd=0;
        yCommandsIndex=0;
        gCodeEnd=0;
        gCodeIndex=0;
        pauseValuesEnd=0;
        pauseValuesIndex=0;
        SysCtlDelay(SysCtlClockGet()*20/3000);//wait 20 milliseconds
        for (i=0;i<2;i++)
        {
            UARTCharPut(UART1_BASE,go[i]);
            SysCtlDelay(SysCtlClockGet()*20/3000);//wait 20 milliseconds
        }

        readyToGo = 0;

        return;
    }//end of engrave()
void testBenchPulse()
{
    //This test pulses the laser in succession on a point
    SysCtlDelay((int) (SysCtlClockGet()*2)/3);
    for (j=7;j>0;j--)
    {
        for (i=0;i<j;i++)
        {
            //Turn the laser on at full power
            PWMPulseWidthSet(PWM0_BASE,PWM_OUT_0,479);
            //delay for x milliseconds
            //turn the laser off to rest
            PWMPulseWidthSet(PWM0_BASE,PWM_OUT_0,1);
            //wait while we rest
            SysCtlDelay((int) (SysCtlClockGet()/timeRest)/3);
        }
        //wait 1 second while we move the wood sample to a new spot
        SysCtlDelay((int) (SysCtlClockGet())/3);
    }
}

void testBenchDuration()
{
    //THIS TEST VARIES THE DURATION OF EQUAL INTENSITY PULSES
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
    SysCtlDelay((int) (SysCtlClockGet()*2)/3);
    for (i=9;i<16;i++)
    {
        //Turn the laser on at full power
        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, GPIO_PIN_6);
        //delay for j*x milliseconds
        {
            SysCtlDelay((int) (i*(SysCtlClockGet()/1000)/3.0));
        }
        //turn the laser off to rest
        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0);
        //wait while we rest
        SysCtlDelay((int) (SysCtlClockGet()/timeRest)/3);
    }
}

```

```

    }
}

void testBenchIntensity()
{
    //THIS CODE RUNS A TEST OF VARYING LASER INTENSITY
    //delay 2 seconds before first engraving
    SysCtlDelay((SysCtlClockGet()*2)/3);
    for (i=7;i>3;i--)
    {
        //start off at the high end and engrave in decreasing fractions of
        denominator 7 out of 100% power
        j= (int) (479.0*i/7.0);
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,j);
        //engrave for 1/timeEngrave seconds
        SysCtlDelay((int) ((SysCtlClockGet()/timeEngrave) / 3));
        // turn the laser off
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, 1);
        // wait for 2 seconds while we slide wood over
        SysCtlDelay(SysCtlClockGet()*2 / (3));
    }

    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,479);
    //engrave for 1/timeEngrave seconds
    SysCtlDelay((int) ((SysCtlClockGet()/100.0) / 3));
    // turn the laser off
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, 1);
    // wait for 2 seconds while we slide wood over
    SysCtlDelay(SysCtlClockGet()*2 / (3));
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,479);
    //engrave for 1/timeEngrave seconds
    SysCtlDelay((int) ((SysCtlClockGet()/125.0) / 3));
    // turn the laser off
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, 1);
    // wait for 2 seconds while we slide wood over
    SysCtlDelay(SysCtlClockGet()*2 / (3));
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0,479);
    //engrave for 1/timeEngrave seconds
    SysCtlDelay((int) ((SysCtlClockGet()/200) / 3));
    // turn the laser off
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, 1);
    // wait for 2 seconds while we slide wood over
    SysCtlDelay(SysCtlClockGet()*2 / (3));
}

void testBenchMotor()
{
    //THIS CODE ROTATES THE MOTOR ONE FULL REVOLUTION ONE DIRECTION THE SWITCHES
    //
    // Enable the GPIO port that is used to send the clock signal to the motors and
    direction
    // PD0:PD1 are forward or reverse and PD2:PD3 are single step
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    //
    // Enable the GPIO pins
    //
    GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

```

```

    GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_1);
    //
    // Set the pins as open drain
    //
    GPIOPadConfigSet(GPIO_PORTD_BASE,GPIO_PIN_1 | GPIO_PIN_2 |
GPIO_PIN_3,GPIO_STRENGTH_8MA,GPIO_PIN_TYPE_OD);
    GPIOPadConfigSet(GPIO_PORTE_BASE,GPIO_PIN_1,GPIO_STRENGTH_8MA,GPIO_PIN_TYPE_OD);
    //
    // Start a loop to rotate the motor both directions one full revolution
    //
    for (j=0;j<2;j++)
    {
        positionX=QEIPositionGet(QEI0_BASE);
        positionY=QEIPositionGet(QEI1_BASE);
        if (j%2==0)
        {
            GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, GPIO_PIN_1);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1, 0);
        }
        else
        {
            GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, 0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1, GPIO_PIN_1);
        }
        for (i=0;i<800;i++)
        {
            //
            // Delay for 2.5 millisecond. Each SysCtlDelay is about 3 clocks.
            //
            SysCtlDelay(SysCtlClockGet() / (motorStepDuration * 3));
            //
            // Bring the clocks high.
            //
            positionX=QEIPositionGet(QEI0_BASE);
            positionY=QEIPositionGet(QEI1_BASE);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_PIN_2);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, GPIO_PIN_3);
            //
            // Delay for 2.5 millisecond. Each SysCtlDelay is about 3 clocks.
            //
            SysCtlDelay(SysCtlClockGet() / (motorStepDuration * 3));
            //
            // Bring the clocks low.
            //
            positionX=QEIPositionGet(QEI0_BASE);
            positionY=QEIPositionGet(QEI1_BASE);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_3, 0);
        }
    }
}

int main(void)
{
    Sys_Clock_Set();
    FPUEnable();
    FPUStackingEnable();
    UART1_Setup();

```

```

PWM_Setup();
QEI_Setup();
GPIO_Setup();

while(1)
{
    correctPlacement(positiveXPixels, positiveYPixels);
    if(readyToGo==1 && rowGood==1)
    {
        firstRun=0;
        engrave();
    }
    else if (rowGood==0)
    {
        pixelsCount=0;
        pixelCorrectX=0;
        pixelCorrectY=0;
        for (i=0;i<1612;i++)
        {
            xCommands[i]='\0';
            yCommands[i]='\0';
            pauseValues[i]='\0';
        }
        for (i=0;i<16012;i++)
        {
            gCode[i]='\0';
        }
        for (i=0;i<16;i++)
        {
            command[0]=UARTCharGetNonBlocking(UART1_BASE);
        }
        UARTRxErrorClear(UART1_BASE);
        xCommandsEnd=0;
        xCommandsIndex=0;
        yCommandsEnd=0;
        yCommandsIndex=0;
        gCodeEnd=0;
        gCodeIndex=0;
        pauseValuesEnd=0;
        pauseValuesIndex=0;
        SysCtlDelay(SysCtlClockGet()*20/3000); //wait 20 milliseconds
        for (i=0;i<2;i++)
        {
            UARTCharPut(UART1_BASE,rs[i]);
            SysCtlDelay(SysCtlClockGet()*20/3000); //wait 20 milliseconds
        }
        readyToGo = 0;
        rowGood=1;
    }
}
}

```

Image Ingestion

“IngestImage.cs”

```
using System;
```



```

using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using System.IO;

namespace IngestImage
{
    class Program
    {
        public void Ingest(int xImageDim, int yImageDim, int xLocation, int yLocation,
String inPath, String outputPath)
        {
            //define image and file input
            Image inputImage = Image.FromFile(inPath);
            //c is original image in bitmap
            Bitmap c = new Bitmap(inputImage);
            //d is new, modified image in bitmap
            Bitmap d = new Bitmap(inputImage, new Size(xImageDim, yImageDim));

            /*G-Code interpretations:

                G00 X--- Y--- Z--- Rapid Positioning      (units in mm, redefine?)
                G01 X--- Y--- Z--- Linear Interpolation  (units in mm, redefine?)
                G04 P--- Dwell                            (units in seconds, redefine?)
                M04 Spindle on                            (Laser on?)
                M05 Spindle off                          (Laser off?)
                M02 End of program
                S-- Set speed of spindle (Laser intensity?)

            */

            //define which x and y pixel we are at
            int row = 0;
            int column = -1;

            //define and create file for G codes
            string path = outputPath + ".gcode";
            if (!File.Exists(path))
            {
                using (StreamWriter sw = File.CreateText(path))
                {
                    //define image size
                    sw.WriteLine("size " + d.Width + "," + d.Height);

                    //write go to starting point of engraving and dwell G-Codes to file
                    sw.WriteLine("G00 X" + xLocation + " Y" + yLocation);
                    sw.WriteLine("G04 P0.500");

                    //loop pulls in pixels, converts to grayscale. i = image width
                    (rows), x = image height (columns)
                    for (int i = 0; i < d.Height; i++)
                    {
                        //increase which row (y value) we are at
                        if (row != 0 || column != -1)
                        {
                            row++;
                        }
                    }
                }
            }
        }
    }
}

```

```

    }

    //makes sure at right pixels
    if ((row % 2 == 0 && row != 0 && column != 0) || column ==
d.Width - 1)
    {
        column++;
    }
    if((row % 2 != 0 && row != 0 && column != d.Width) || column == 0)
    {
        column--;
    }

    for (int x = 0; x < d.Width; x++)
    {
        //increase/decreases which column (x value) we are at
        if (row % 2 != 0)
        {
            column--;
        }
        else
        {
            column++;
        }

        //select pixel
        Color oc = d.GetPixel(column, row);

        //define to grayscale
        int grayScale = (int)((oc.R * 0.3) + (oc.G * 0.59) + (oc.B *
0.11));

        //truncate to 8 shades
        grayScale = grayScale / 32;

        //move back to being in increments of 32 - creates better
        variance in shades
        grayScale = grayScale * 32;

        //set new pixel to grayscale
        Color nc = Color.FromArgb(oc.A, grayScale, grayScale,
grayScale);

        //write pixel to specified position
        d.SetPixel(column, row, nc);

        //G-Code to move to next pixel
        sw.WriteLine("G01 X" + (column + xLocation) + " Y" + (row +
yLocation));

        //G-Code to set intensity of laser
        sw.WriteLine("S" + grayScale / 32);

        //G-Code to turn on laser
        sw.WriteLine("M04");

        //G-Code for dwell
        sw.WriteLine("G04 P0.040");
    }
}

```

```

        //G-Code to turn off laser
        sw.WriteLine("M05");
    }
}

//End G-Code File
sw.WriteLine("M02");
sw.Close();
}
}
//save new image
d.Save(outPath + ".jpg");

//dispose of bitmap variables
c.Dispose();
d.Dispose();

//end program
return;
}
}
}

```

Graphical User Interface

“Program.cs”

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PLED_GUI
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
            Application.Exit();
        }
    }
}

```

“Form1.cs”

```

using System;

```

```

using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using IngestImage;
using System.IO;

namespace PLED_GUI
{
    public partial class Form1 : Form
    {
        //defines return values to be used for wood dimension window
        public string ReturnValue1 { get; set; }
        public string ReturnValue2 { get; set; }

        //defines strings, ints, and doubles to be used
        string xDimension, yDimension, imgPathString, filePath, outFilePath;
        int xDimPix, yDimPix, xLocPix, yLocPix, xImgSize, yImgSize;
        double xyratio, xywoodratio, xdoubletmp, ydoubletmp, tmpwoodsize, time;

        public Form1()
        {
            //Initializes application
            InitializeComponent();

            //initializes file dialog settings
            openFileDialog1.Title = "Load PLED Image";
            openFileDialog1.InitialDirectory = @"C:";
            openFileDialog1.Filter = "Image Files (*.jpg, *.jpeg, *.png, *.gif, *.bmp)|*.jpg; *.jpeg; *.png; *.gif; *.bmp";
            openFileDialog1.FilterIndex = 1;
            openFileDialog1.RestoreDirectory = true;

            //initialize save dialog settings
            saveFileDialog1.Title = "Select output File";
            saveFileDialog1.InitialDirectory = filePath;
            saveFileDialog1.Filter = "Output Filename (no extension)|";
            saveFileDialog1.FilterIndex = 1;
            saveFileDialog1.RestoreDirectory = true;

            //start image load button disabled
            imgLoad.Enabled = false;
        }

        //handler for clicking on SelectWoodDimensions button
        private void SelectWoodDimensions_Click(object sender, EventArgs e)
        {
            //creates instance of window
            using (var getWoodDimensions = new woodDimensions())
            {
                //opens window
                var result = getWoodDimensions.ShowDialog();
            }
        }
    }
}

```

```

//set invalid or closed case
if (result == DialogResult.Abort)
{
    if(getWoodDimensions.ReturnValue1 != null)
    {
        x_woodDimension.Text = getWoodDimensions.ReturnValue1;
        y_woodDimension.Text = getWoodDimensions.ReturnValue2;
    }
}

if (result == DialogResult.OK)
{
    //saves dimensions in inches
    xDimension = getWoodDimensions.ReturnValue1;
    yDimension = getWoodDimensions.ReturnValue2;

    //displays dimensions in inches
    x_woodDimension.Text = xDimension;
    y_woodDimension.Text = yDimension;

    //converts, saves and displays dimensions in pixel count
    xDimPix = Convert.ToUInt16(Convert.ToDouble(xDimension) / 0.009);
    yDimPix = Convert.ToUInt16(Convert.ToDouble(yDimension) / 0.009);
    xWoodDimPixels.Text = Convert.ToString(xDimPix);
    yWoodDimPixels.Text = Convert.ToString(yDimPix);

    //sets max slider value based on dimension in pixels
    xSlider.Maximum = xDimPix - xImgSize;
    ySlider.Maximum = yDimPix - yImgSize;

    //set wood plaque ratio and dimensions
    xywoodratio = Convert.ToDouble(xDimPix) / Convert.ToDouble(yDimPix);
    if (xDimPix > yDimPix)
    {
        PlaqueSize.Width = 330;
        tmpwoodsize = 330 * (1 / xywoodratio);
        PlaqueSize.Height = Convert.ToUInt16(tmpwoodsize);
    }
    else if (xDimPix < yDimPix)
    {
        PlaqueSize.Height = 330;
        tmpwoodsize = 330 * xywoodratio;
        PlaqueSize.Width = Convert.ToUInt16(tmpwoodsize);
    }
    else
    {
        PlaqueSize.Width = PlaqueSize.Height = 330;
    }

    if(xImgSize != 0 && yImgSize !=0)
    {
        while (xImgSize > xDimPix || yImgSize > yDimPix)
        {
            if (xImgSize > yImgSize)
            {
                //adjust image size (pixel count) according to plaque
                size if image is larger
            }
        }
    }
}

```

```

        if (xImgSize > xDimPix)
        {
            //sets y image dimension to y wood dimension
            xImgSize = xDimPix;
            //sets x dimension according to conversion above
            ydoubletmp = xImgSize * (1 / xyratio);
            yImgSize = Convert.ToInt16(ydoubletmp);
        }
    }

    else if (yImgSize > xImgSize)
    {
        //adjust image size (pixel count) according to plaque
size if image is larger
        if (yImgSize > yDimPix)
        {
            //sets x image dimension to x wood dimension
            yImgSize = yDimPix;
            //sets y dimension according to conversion above
            xdoubletmp = yImgSize * xyratio;
            xImgSize = Convert.ToInt16(xdoubletmp);
        }
    }

    }

    //set max slider location positions
    xSlider.Maximum = xDimPix - xImgSize;
    ySlider.Maximum = yDimPix - yImgSize;

    //set max slider sizes
    ximgslider.Maximum = xImgSize;
    yimgslider.Maximum = yImgSize;

    //set default position and size values
    xSlider.Value = 0;
    ySlider.Value = 0;
    ximgslider.Value = xImgSize;
    yimgslider.Value = yImgSize;
    ximgsizebox.Text = Convert.ToString(xImgSize);
    yimgsizebox.Text = Convert.ToString(yImgSize);
    xLocBox.Text = "0";
    yLocBox.Text = "0";

    //set image size
    xdoubletmp = (xImgSize * PlaqueSize.Width) / xDimPix;
    ydoubletmp = xdoubletmp * (1 / xyratio);
    imgBox.Width = Convert.ToInt16(xdoubletmp);
    imgBox.Height = Convert.ToInt16(ydoubletmp);
}
//disable button
SelectWoodDimensions.Enabled = false;
imgLoad.Enabled = true;
}
}
}

private void openFileDialog1_FileOk(object sender, CancelEventArgs e)

```

```

{
}

//handler for imgLoad button
private void imgLoad_Click(object sender, EventArgs e)
{
    //opens openFileDialog1 form
    DialogResult result = openFileDialog1.ShowDialog();

    if (result == DialogResult.OK)
    {
        //obtains file path
        imgPathString = openFileDialog1.FileName;
        filePath = Path.GetDirectoryName(imgPathString);
        imgPath.Text = imgPathString;
        //loads image
        Image inputImage = Image.FromFile(imgPathString);
        //c is the image in bitmap
        Bitmap c = new Bitmap(inputImage);

        //sets image size according to image
        xImgSize = c.Width;
        yImgSize = c.Height;

        //sets xyratio
        xyratio = Convert.ToDouble(xImgSize) / Convert.ToDouble(yImgSize);

        while (xImgSize > xDimPix || yImgSize > yDimPix)
        {
            if (xImgSize == yImgSize)
            {
                //adjust image size (pixel count) to fit plaque
                xImgSize = xDimPix;
                yImgSize = yDimPix;
            }

            else if (xImgSize > yImgSize)
            {
                //adjust image size (pixel count) according to plaque size if
image is larger
                if (xImgSize > xDimPix)
                {
                    //sets y image dimension to y wood dimension
                    xImgSize = xDimPix;
                    //sets x dimension according to conversion above
                    ydoubletmp = xImgSize * (1 / xyratio);
                    yImgSize = Convert.ToInt16(ydoubletmp);
                }
            }

            else if (yImgSize > xImgSize)
            {
                //adjust image size (pixel count) according to plaque size if
image is larger
                if (yImgSize > yDimPix)
                {
                    //sets x image dimension to x wood dimension

```

```

        yImgSize = yDimPix;
        //sets y dimension according to conversion above
        xdoubletmp = yImgSize * xyratio;
        xImgSize = Convert.ToInt16(xdoubletmp);
    }
}

//set max slider location positions
xSlider.Maximum = xDimPix - xImgSize;
ySlider.Maximum = yDimPix - yImgSize;

//set max slider sizes
ximgslider.Maximum = xImgSize;
yimgslider.Maximum = yImgSize;

//set default position and size values
xSlider.Value = 0;
ySlider.Value = 0;
ximgslider.Value = xImgSize;
yimgslider.Value = yImgSize;
ximgsizebox.Text = Convert.ToString(xImgSize);
yimgsizebox.Text = Convert.ToString(yImgSize);
xLocBox.Text = "0";
yLocBox.Text = "0";

//set image size
xdoubletmp = (xImgSize * PlaqueSize.Width) / xDimPix;
ydoubletmp = xdoubletmp * (1 / xyratio);
imgBox.Width = Convert.ToInt16(xdoubletmp);
imgBox.Height = Convert.ToInt16(ydoubletmp);
}
imgBox.ImageLocation = imgPathString;
imgLoad.Enabled = false;

//Estimates engrave time for image
time = (((Convert.ToDouble(xImgSize) * Convert.ToDouble(yImgSize) * (0.0155))
+ (0.0005 * (Convert.ToDouble(xLocPix) + Convert.ToDouble(yLocPix)))) / 3600);
EngTime.Text = Convert.ToString(time);
}

//x slider handler
private void xSlider_Scroll(object sender, EventArgs e)
{
    //allows user to adjust location to place image on x axis
    xLocPix = xSlider.Value;
    //displays pixel location
    xLocBox.Text = Convert.ToString(xLocPix);

    //moves image block representation on x axis
    xdoubletmp = ((Convert.ToDouble(xLocPix) * (PlaqueSize.Width -
Convert.ToDouble(imgBox.Width))) / Convert.ToDouble(xSlider.Maximum));
    imgBox.Location = new Point((440 + Convert.ToInt16(xdoubletmp)),
imgBox.Location.Y);

    //Estimates engrave time for image

```



```

        time = (((Convert.ToDouble(xImgSize) * Convert.ToDouble(yImgSize) * (0.0155))
+ (0.0005 * (Convert.ToDouble(xLocPix) + Convert.ToDouble(yLocPix)))) / 3600);
        EngTime.Text = Convert.ToString(time);
    }

    //y slider handler
    private void ySlider_Scroll(object sender, EventArgs e)
    {
        //allows user to adjust location to place image on y axis
        yLocPix = ySlider.Value;
        //displays pixel location
        yLocBox.Text = Convert.ToString(yLocPix);

        //moves image block representation on y axis
        ydoubletmp = ((Convert.ToDouble(yLocPix) * (PlaqueSize.Height -
Convert.ToDouble(imgBox.Height))) / Convert.ToDouble(ySlider.Maximum));
        imgBox.Location = new Point(imgBox.Location.X, (13 +
Convert.ToUInt16(ydoubletmp)));

        //Estimates engrave time for image
        time = (((Convert.ToDouble(xImgSize) * Convert.ToDouble(yImgSize) * (0.0155))
+ (0.0005 * (Convert.ToDouble(xLocPix) + Convert.ToDouble(yLocPix)))) / 3600);
        EngTime.Text = Convert.ToString(time);
    }

    //centers image
    private void cent_Click(object sender, EventArgs e)
    {
        //centers on actual image
        xLocPix = (xDimPix - xImgSize) / 2;
        yLocPix = (yDimPix - yImgSize) / 2;
        xSlider.Value = xLocPix;
        ySlider.Value = yLocPix;
        xLocBox.Text = Convert.ToString(xLocPix);
        yLocBox.Text = Convert.ToString(yLocPix);

        //centers on diagram
        if(xDimPix != xImgSize)
        {
            xdoubletmp = ((Convert.ToDouble(xLocPix) * (PlaqueSize.Width -
Convert.ToDouble(imgBox.Width))) / Convert.ToDouble(xSlider.Maximum));
            imgBox.Location = new Point((440 + Convert.ToUInt16(xdoubletmp)),
imgBox.Location.Y);
        }

        if(yDimPix != yImgSize)
        {
            ydoubletmp = ((Convert.ToDouble(yLocPix) * (PlaqueSize.Height -
Convert.ToDouble(imgBox.Height))) / Convert.ToDouble(ySlider.Maximum));
            imgBox.Location = new Point(imgBox.Location.X, (13 +
Convert.ToUInt16(ydoubletmp)));
        }

        //Estimates engrave time for image
        time = (((Convert.ToDouble(xImgSize) * Convert.ToDouble(yImgSize) * (0.0155))
+ (0.0005 * (Convert.ToDouble(xLocPix) + Convert.ToDouble(yLocPix)))) / 3600);
        EngTime.Text = Convert.ToString(time);
    }
}

```

```

//vertically centers image
private void vertCent_Click(object sender, EventArgs e)
{
    //vertically centers actual image
    yLocPix = (yDimPix - yImgSize) / 2;
    ySlider.Value = yLocPix;
    yLocBox.Text = Convert.ToString(yLocPix);

    //vertically centers diagram
    if (yDimPix != yImgSize)
    {
        ydoubletmp = ((Convert.ToDouble(yLocPix) * (PlaqueSize.Height -
Convert.ToDouble(imgBox.Height))) / Convert.ToDouble(ySlider.Maximum));
        imgBox.Location = new Point(imgBox.Location.X, (13 +
Convert.ToInt16(ydoubletmp)));
    }

    //Estimates engrave time for image
    time = (((Convert.ToDouble(xImgSize) * Convert.ToDouble(yImgSize) * (0.0155))
+ (0.0005 * (Convert.ToDouble(xLocPix) + Convert.ToDouble(yLocPix)))) / 3600);
    EngTime.Text = Convert.ToString(time);
}

//horizontally centers image
private void horzCent_Click(object sender, EventArgs e)
{
    //horizontally centers actual image
    xLocPix = (xDimPix - xImgSize) / 2;
    xSlider.Value = xLocPix;
    xLocBox.Text = Convert.ToString(xLocPix);

    //horizontally centers diagram
    if(xDimPix != xImgSize)
    {
        xdoubletmp = ((Convert.ToDouble(xLocPix) * (PlaqueSize.Width -
Convert.ToDouble(imgBox.Width))) / Convert.ToDouble(xSlider.Maximum));
        imgBox.Location = new Point((440 + Convert.ToInt16(xdoubletmp)),
imgBox.Location.Y);
    }

    //Estimates engrave time for image
    time = (((Convert.ToDouble(xImgSize) * Convert.ToDouble(yImgSize) * (0.0155))
+ (0.0005 * (Convert.ToDouble(xLocPix) + Convert.ToDouble(yLocPix)))) / 3600);
    EngTime.Text = Convert.ToString(time);
}

//handler for ximgslider slider
private void ximgslider_Scroll(object sender, EventArgs e)
{
    //allows user to adjust image size on x axis
    xImgSize = ximgslider.Value;
    //displays pixel count
    ximgsizebox.Text = Convert.ToString(xImgSize);

    //sets y size according to conversion above and updates slider
    ydoubletmp = xImgSize * (1 / xyratio);
    yImgSize = Convert.ToInt16(ydoubletmp);
}

```

```

yimgsizebox.Text = Convert.ToString(yImgSize);
yimgslider.Value = yImgSize;

//set max slider location positions
xSlider.Maximum = xDimPix - xImgSize;
ySlider.Maximum = yDimPix - yImgSize;

//set default slider location
xSlider.Value = 0;
ySlider.Value = 0;
xLocPix = 0;
yLocPix = 0;
xLocBox.Text = "0";
yLocBox.Text = "0";
imgBox.Location = new Point(440, 13);

//set image size
xdoubletmp = (xImgSize * PlaqueSize.Width) / xDimPix;
ydoubletmp = xdoubletmp * (1 / xyratio);
imgBox.Width = Convert.ToUInt16(xdoubletmp);
imgBox.Height = Convert.ToUInt16(ydoubletmp);

//Estimates engrave time for image
time = (((Convert.ToDouble(xImgSize) * Convert.ToDouble(yImgSize) * (0.0155))
+ (0.0005 * (Convert.ToDouble(xLocPix) + Convert.ToDouble(yLocPix)))) / 3600);
EngTime.Text = Convert.ToString(time);
}

//handler for yimgslider slider
private void yimgslider_Scroll(object sender, EventArgs e)
{
    //allows user to adjust image size on x axis
    yImgSize = yimgslider.Value;
    //displays pixel count
    yimgsizebox.Text = Convert.ToString(yImgSize);

    //sets y size according to conversion above and updates slider
    xdoubletmp = yImgSize * xyratio;
    xImgSize = Convert.ToUInt16(xdoubletmp);
    ximgsizebox.Text = Convert.ToString(xImgSize);
    ximgslider.Value = xImgSize;

    //set max slider location positions
    xSlider.Maximum = xDimPix - xImgSize;
    ySlider.Maximum = yDimPix - yImgSize;

    //set default slider location
    xSlider.Value = 0;
    ySlider.Value = 0;
    xLocPix = 0;
    yLocPix = 0;
    xLocBox.Text = "0";
    yLocBox.Text = "0";
    imgBox.Location = new Point(440, 13);

    //set image size
    xdoubletmp = (xImgSize * PlaqueSize.Width) / xDimPix;
    ydoubletmp = xdoubletmp * (1 / xyratio);

```

```

        imgBox.Width = Convert.ToUInt16(xdoubletmp);
        imgBox.Height = Convert.ToUInt16(ydoubletmp);

        //Estimates engrave time for image
        time = (((Convert.ToDouble(xImgSize) * Convert.ToDouble(yImgSize) * (0.0155))
+ (0.0005 * (Convert.ToDouble(xLocPix) + Convert.ToDouble(yLocPix)))) / 3600);
        EngTime.Text = Convert.ToString(time);
    }

    private void submitPLED_Click(object sender, EventArgs e)
    {
        DialogResult result = saveFileDialog1.ShowDialog();
        if(result == DialogResult.OK)
        {
            //select save path
            outFilePath = saveFileDialog1.FileName;

            //pass everything to image ingestion backend
            IngestImage.Program imageIngest = new IngestImage.Program();
            imageIngest.Ingest(xImgSize, yImgSize, xLocPix, yLocPix, imgPathString,
outFilePath);

            //create outpath file
            StreamWriter sw = File.CreateText(@"C:\PLED\PLEDpath.txt");
            sw.Write(outFilePath + ".gcode");
            sw.Close();

            //tell user job ready to begin begins the engraving process
            var jobComplete = new complete();
            jobComplete.ShowDialog();

            var eng = new engraving();

            Application.Exit();
        }
    }

    //handler for endPLED button
    private void endPLED_Click_1(object sender, EventArgs e)
    {
        //closes application
        Application.Exit();
    }
}

```

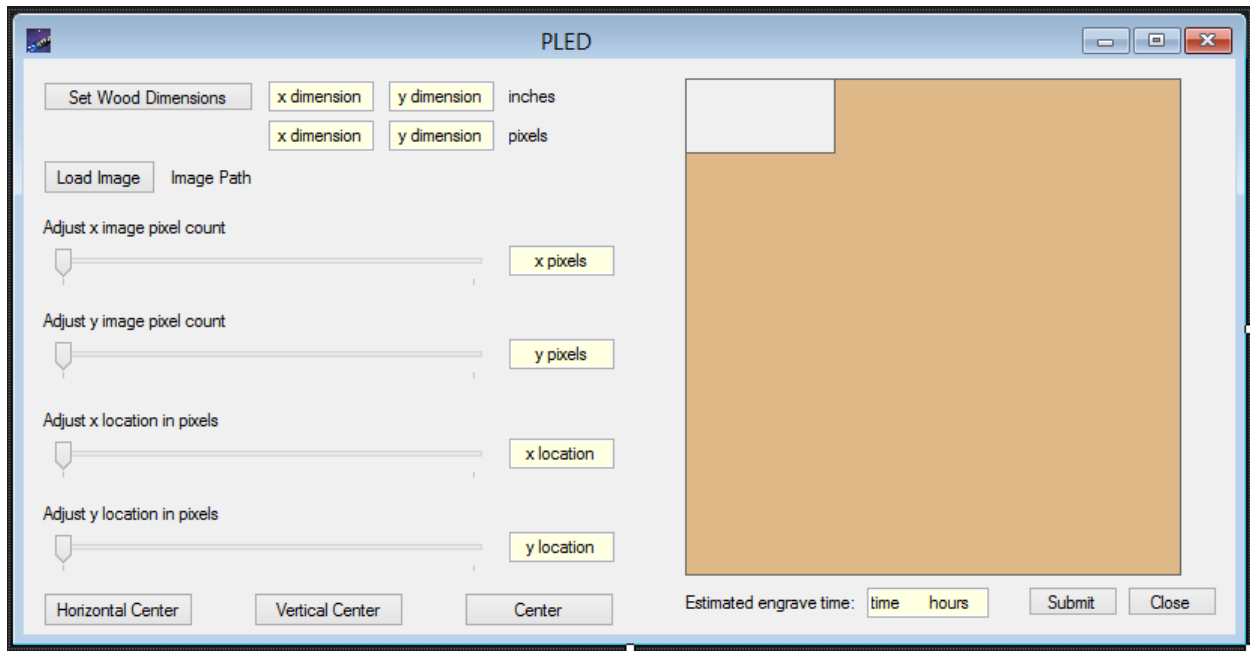


Figure 15. Form1.Designer.cs

“complete.cs”

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PLED_GUI
{
    public partial class complete : Form
    {
        public complete()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.OK;
            this.Close();
        }
    }
}

```

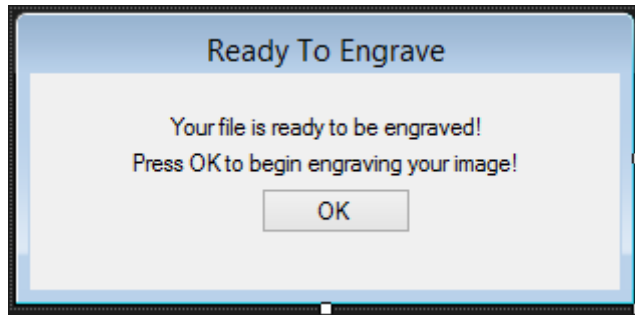


Figure 16. complete.Designer.cs

“woodDimensions.cs”

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PLED_GUI
{
    public partial class woodDimensions : Form
    {
        public string ReturnValue1 { get; set; }
        public string ReturnValue2 { get; set; }

        public woodDimensions()
        {
            InitializeComponent();
        }

        private void setDimension_Click(object sender, EventArgs e)
        {
            if (x_woodDimension.Text == "x dimension (inches)" || y_woodDimension.Text ==
"x dimension (inches)"
                || Convert.ToDouble(x_woodDimension.Text) < 0 ||
Convert.ToDouble(x_woodDimension.Text) > 14.4
                || Convert.ToDouble(y_woodDimension.Text) < 0 ||
Convert.ToDouble(y_woodDimension.Text) > 14.4)
            {
                this.ReturnValue1 = "Invalid Input";
                this.ReturnValue2 = "Invalid Input";
                this.DialogResult = DialogResult.Abort;
                this.Close();
            }

            else
            {
                this.ReturnValue1 = x_woodDimension.Text;
                this.ReturnValue2 = y_woodDimension.Text;
                this.DialogResult = DialogResult.OK;
            }
        }
    }
}

```

```

        this.Close();
    }
}

private void closeDim_Click_1(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.Abort;
    this.Close();
}
}
}

```

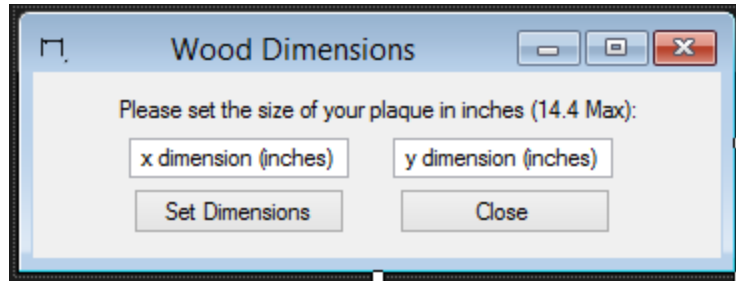


Figure 17. woodDimensions.Designer.cs

“engraving.cs”

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PLED_GUI
{
    public partial class engraving : Form
    {
        public engraving()
        {
            InitializeComponent();
            this.Show();
            System.Diagnostics.Process.Start("C:/PLED/src/SerialToPC.exe");
        }
    }
}

```



Figure 18. engraving.Designer.cs

9.6 Appendix 6 – Mathematical Model

The mathematical model for the Plaque Laser Engraver Device (PLED) is below. It details the torque requirements of the system, as well as its maximum load and velocity.

$$T = -T_{max} \sin\left(\frac{2\pi\theta}{\rho\Delta\theta}\right) \quad T = -0.45 \sin\left(\frac{2\pi\left(-\frac{1.8}{2}\right)}{2(1.8)}\right) = 0.45 \text{ Nm}$$

$T = \text{torque source}, T_{max} = \text{holding torque}, \theta = -\Delta\theta = -\text{step angle}$

$\rho = \text{number of phases}$

Using the example on page 720 (Mechatronics: An Integrated Approach), we can apply Newton's second law to the torques in the system, resulting in the following equation of motion:

$\text{torque supplied} - \text{torque load} - \text{viscous damping} = \text{total torque} = J\ddot{\theta}$

$$T - T_L - T_B(\theta, \dot{\theta}) = J\ddot{\theta}$$

We believe that our motor is a PM (permanent magnet) stepper motor. According to page 722,

$$T = -k_m i_p \sin(n_r \theta)$$

$T = \text{torque}, i_p = \text{phase current}, n_r = \text{number of rotor teeth}$

$k_m = \text{torque constant}, \theta = \text{rotor position}$

Using the torque of the motors and the radius of the sprocket to be used in our system, we can approximate the force that can therefore be produced, and the mass that can be moved.

$$\text{Force} = \frac{\text{torque}}{\text{sprocket radius}}$$

$$F = \frac{T}{r} = \frac{0.45}{0.01} = 45 \text{ N} \cong 10 \text{ lbs}$$

Using specific information relative to our system, including our motor driver, microcontroller, and the specs of our components, we can determine the unloaded maximum linear velocity of our system.

$$\omega = \text{angular velocity } \left(\frac{m}{s}\right), f = \text{clock speed (Hz)}, T = \frac{\text{counts}}{\text{step}},$$

$$r = \text{radius (m)}$$

$$SPR = \frac{\text{steps}}{\text{revolution}}, rps = \text{revolutions per second}, v = \text{linear velocity}$$

$$f = 16 \times 10^6, T = 3813, SPR = 400, r = 8.75 \times 10^{-3}$$

$$rps = f \cdot \frac{1}{T} \cdot \frac{1}{SPR} = 16 \times 10^6 \cdot \frac{1}{3813} \cdot \frac{1}{400} = 10.5 \text{ rev/sec}$$

$$\omega = 2\pi(rps) = 2\pi(10.5) = 65.9 \text{ rads/sec}$$

$$v = \omega r = 65.9 \cdot 8.75 \times 10^{-3} = 0.58 \text{ m/s}$$

v represents our maximum theoretical unloaded motor velocity of our system. Our loaded motors will move more slowly, however even if the speed is reduced by 50%, we will be moving very quickly. Additionally, by testing, we know that moving a single step takes 250 μ s

Now we will consider the loaded velocity using a simplified model:

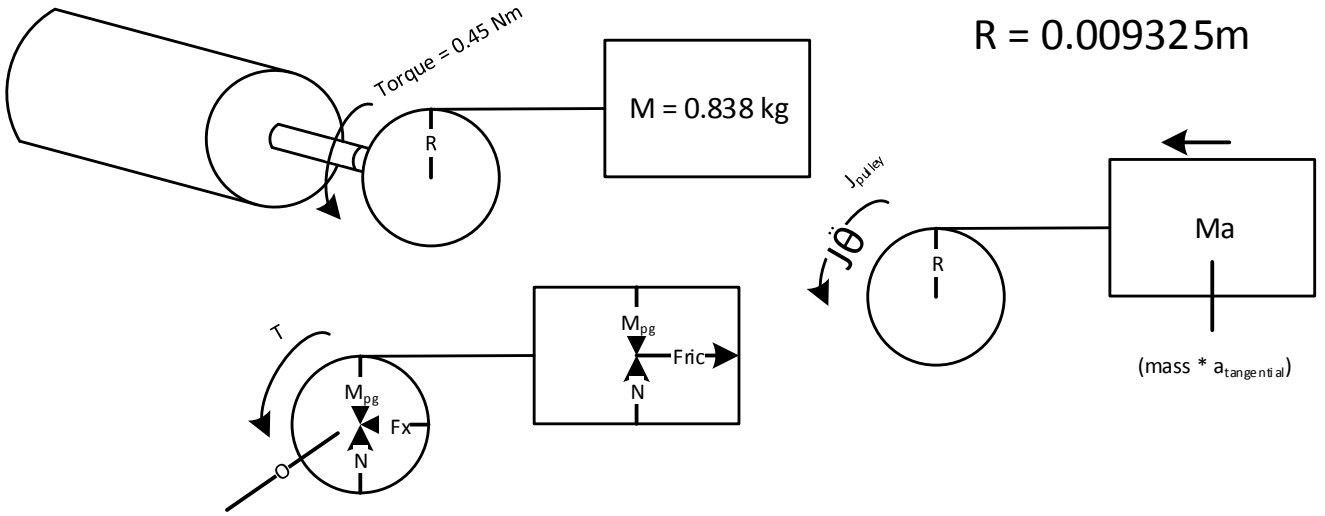


Figure 19. Diagram of mathematical model of motors

$$\Sigma M_0 = Torque - Friction(R) = J_{pulley}\ddot{\theta} + (mass \cdot a_t)R, \quad a_t = \ddot{\theta}R$$

$$(J_{pulley} + Mass \cdot R^2)\ddot{\theta} = Torque - Friction(R)$$

From solid works model:

$$J_{pulley} = 2.337 \times 10^{-6} \text{ kgm}^2, \quad J_{mass} = Mass \cdot R^2 = 7.287 \times 10^{-5} \text{ kgm}^2,$$

$$J_{total} = 7.521 \times 10^{-5} \text{ kgm}^2$$

Assuming sliding AKA bad fiction of mass we will use (Teflon on steel) value (0.04)

$$J_{total}\ddot{\theta} = 0.45 \text{ Nm} - ((0.04)(9.81)(8.38)(0.009325)) \rightarrow \text{Negligible} \cong 0.0031 \text{ Nm}$$

$$J_{total}\ddot{\theta} = 0.45 \text{ Nm}$$

$$\frac{\delta \dot{\theta}}{\delta t} = \ddot{\theta} = 5983.56 \text{ rad/s}^2 \rightarrow \dot{\theta} = 5.983.6(t)$$

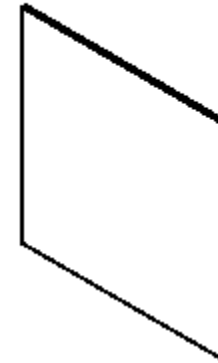
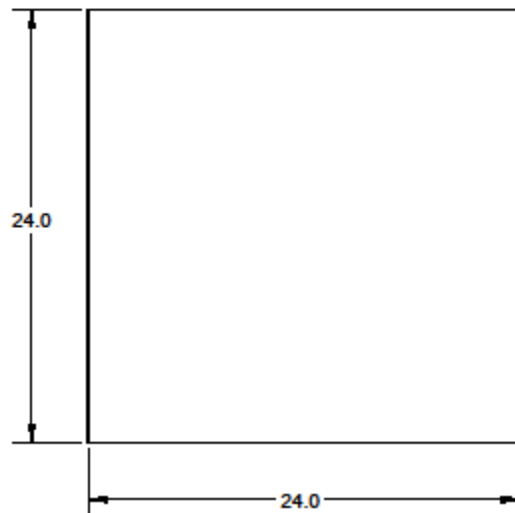
Given a short time step of 0.25 seconds to move, we get $\dot{\theta} \cong 1496$

$$\dot{\theta}R = \text{velocity of gantry after 0.25 seconds} = 13.9 \text{ or } \cong 14 \text{ m/s}$$

This seems fast. We will not work at this speed. We will go a lot slower.

9.7 Appendix 7 – Machining

On the following pages, the machine drawings for the Plaque Laser Engraver Device can be found. In Appendix 8, the assembly instructions for the PLED are also included.



ITEM NO.	PART NUMBER	QTY.
1	BASE	1



Mechanical & Aerospace
ENGINEERING
Utah State University

PROJECT:
PLED

DRAFTED BY: T. SHORTHILL

CHECKED BY: Z. GARRARD

APPROVED BY: C. WOOD

DATE APPROVED: 4/28/2018

SHEET SCALE: 1:3

SHEET NUMBER: 1 of 1

SPECIFICATIONS AND TOLERANCES

UNLESS OTHERWISE NOTED
DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009

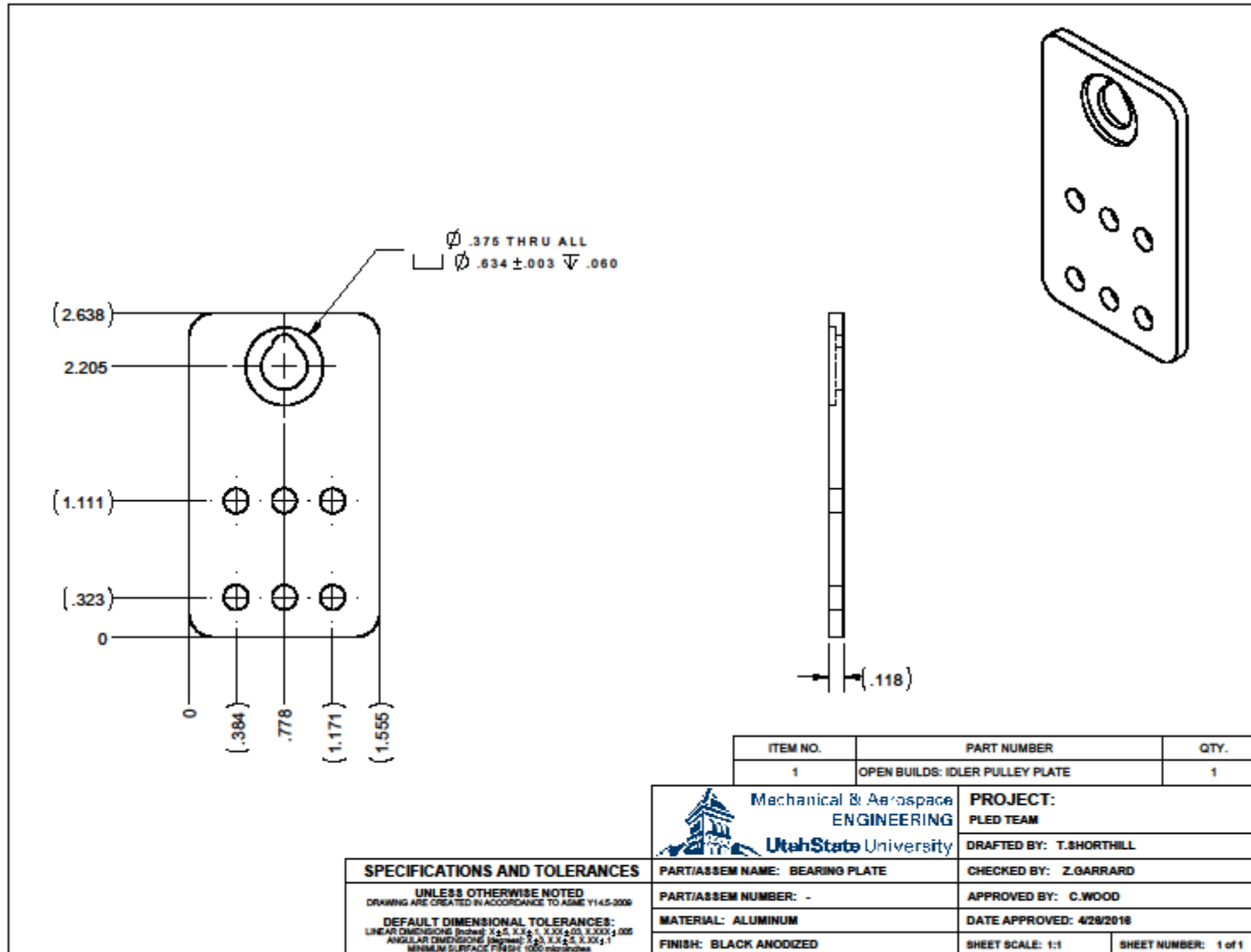
DEFAULT DIMENSIONAL TOLERANCES:
LINEAR DIMENSIONS (Inches): X.X, X.XX, X.XXX ±.005, X.XXX ±.002
ANGULAR DIMENSIONS (Degrees): X.X, X.XX ±.1
MINIMUM SURFACE FINISH: 1000 microinches

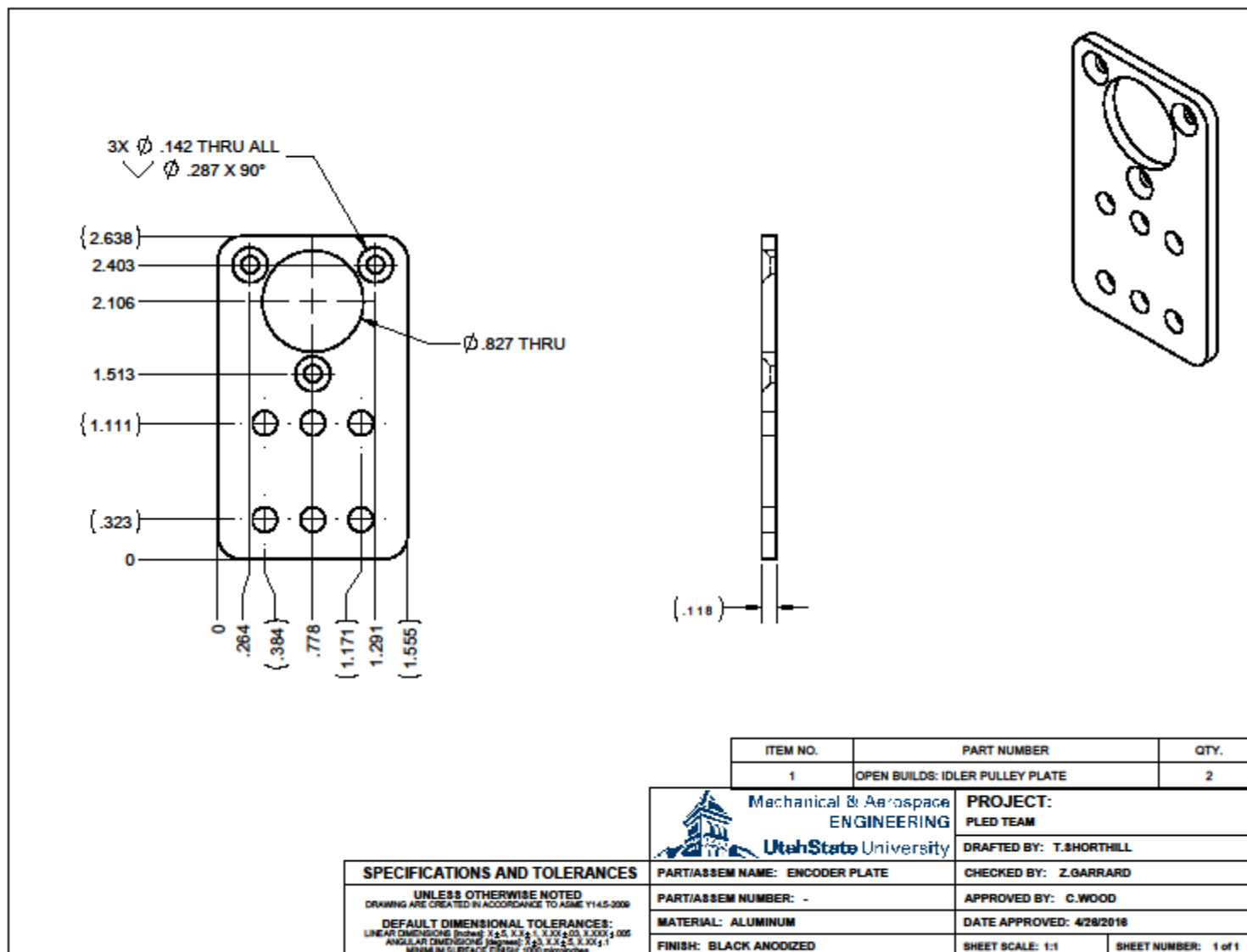
PART/ASSEM NAME: BASE

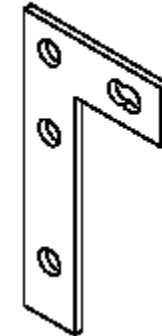
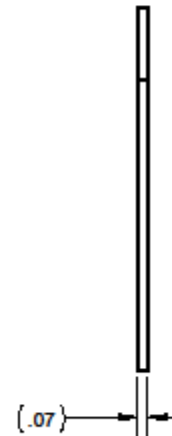
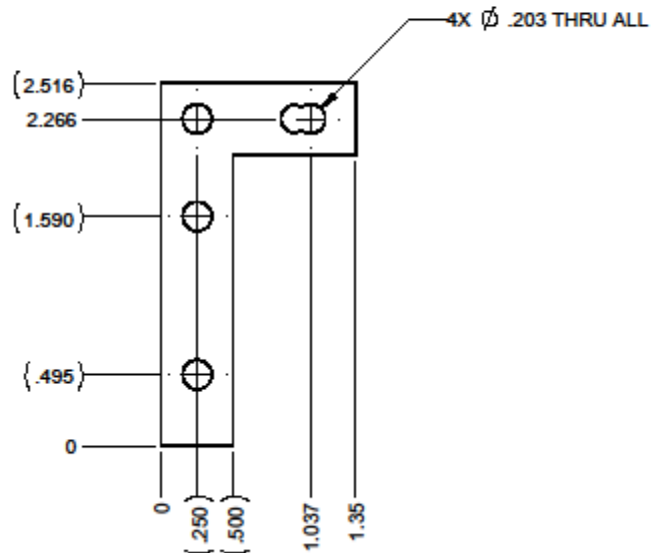
PART/ASSEM NUMBER: -

MATERIAL: OSB WOOD BOARD

FINISH: BLK SPRAY PAINT







NOTE: THIS IS A PART MODIFICATION

ITEM NO.	PART NUMBER	QTY.
1	ZINC PLATED STEEL 2.5 INCH L- BRACKET	4

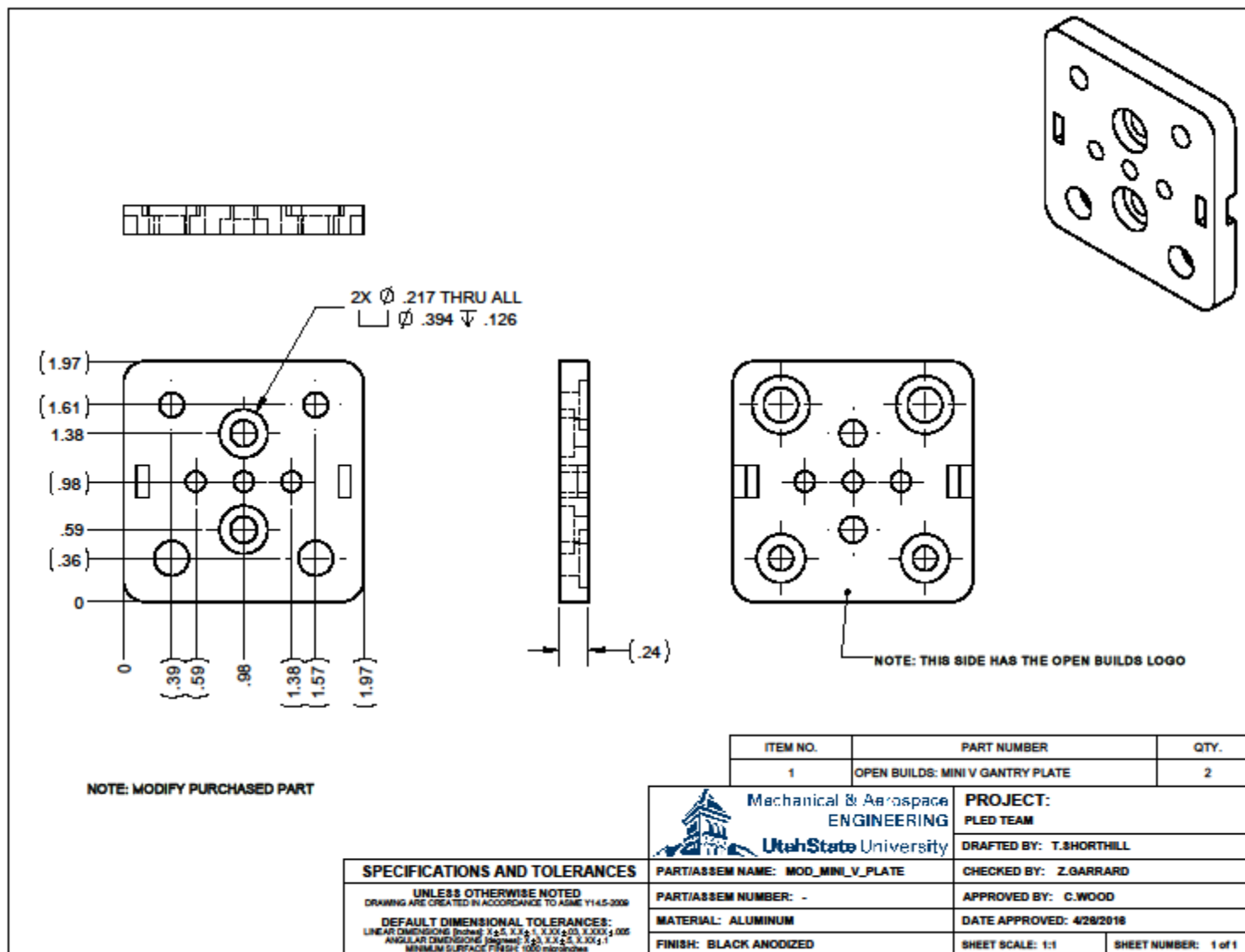


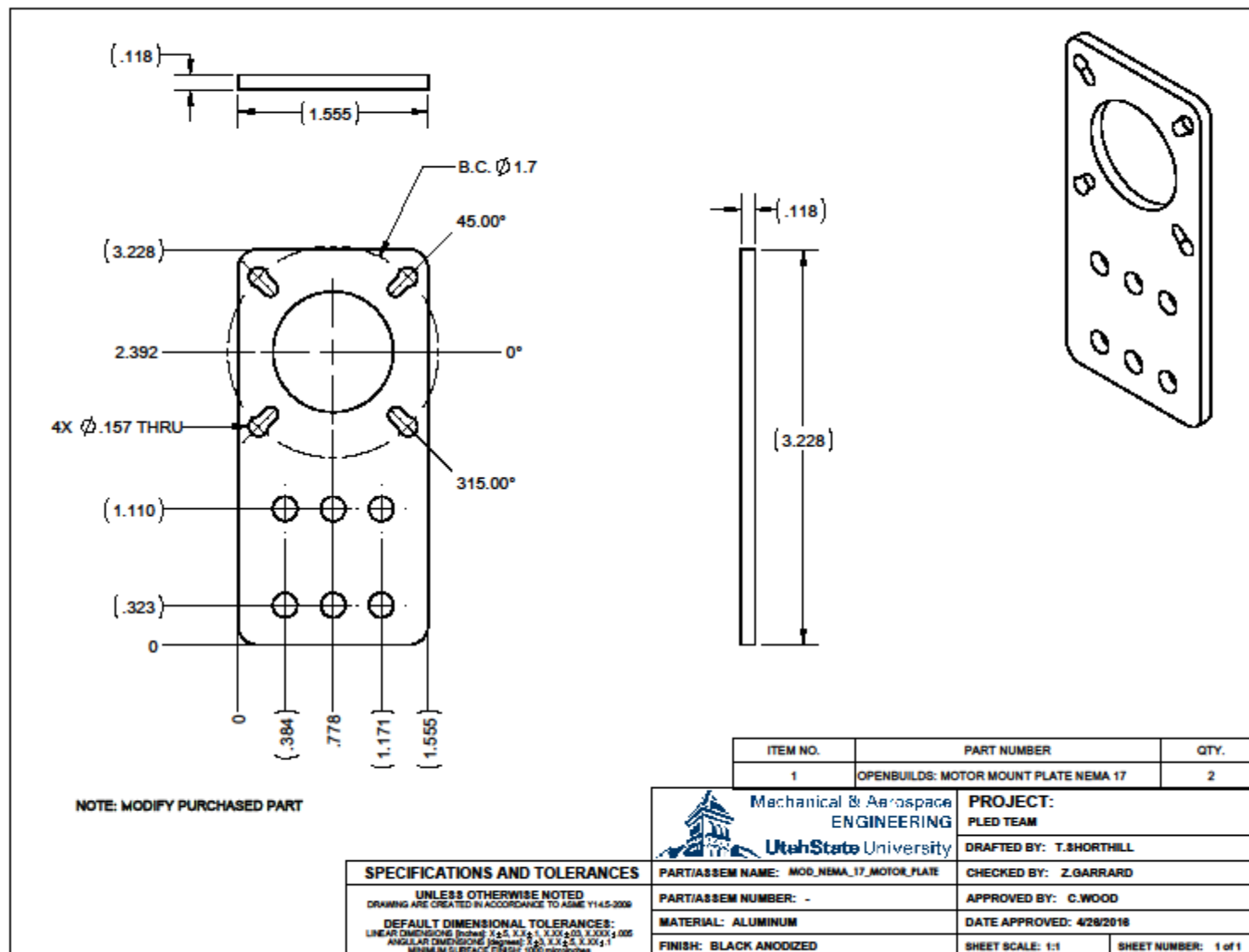
PROJECT:
PLED TEAM
DRAFTED BY: T.SHORTHILL

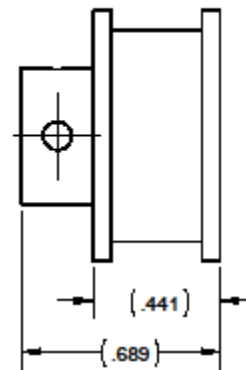
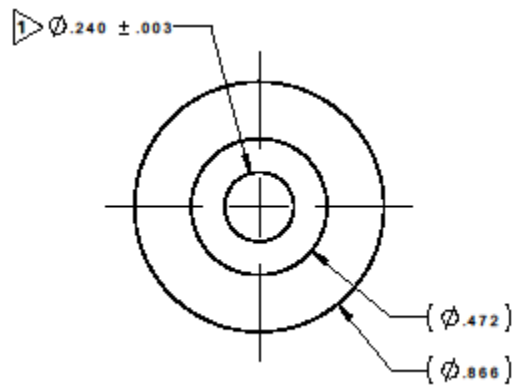
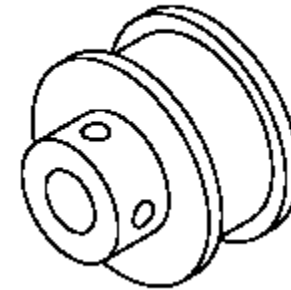
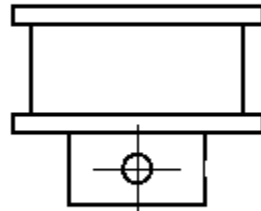
SPECIFICATIONS AND TOLERANCES

UNLESS OTHERWISE NOTED
DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009
DEFAULT DIMENSIONAL TOLERANCES:
LINEAR DIMENSIONS (Inches): X.X, X.XX, X.XXX ±.005
ANGULAR DIMENSIONS (Degrees): X.X, X.XX ±.1
MINIMUM SURFACE FINISH: 1000 microinches

PART/ASSEM NAME: L_BRACKET	CHECKED BY: Z.GARRARD
PART/ASSEM NUMBER: -	APPROVED BY: C.WOOD
MATERIAL: ZINC PLATED STEEL	DATE APPROVED: 4/28/2018
FINISH: -	SHEET SCALE: 1:1
	SHEET NUMBER: 1 of 1







NOTES:
1. MODIFY THE THRU HOLE DIAMETER

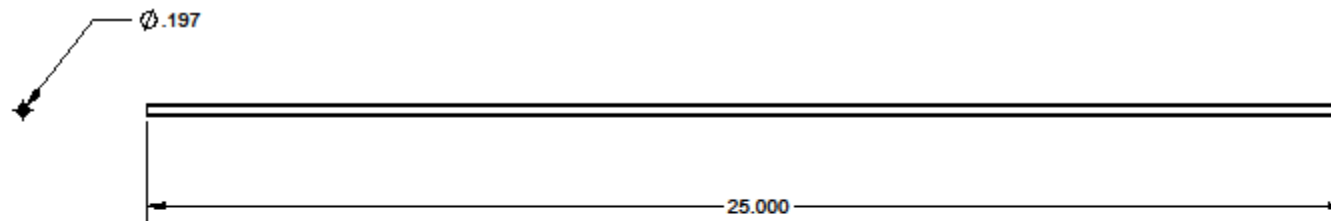
ITEM NO.	PART NUMBER	QTY.
1	OPEN BUILDS: GT2 TIMING PULLEY 30 TOOTH	2



PROJECT:
PLED TEAM
DRAFTED BY: T.SHORTHILL

SPECIFICATIONS AND TOLERANCES
UNLESS OTHERWISE NOTED
DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009
DEFAULT DIMENSIONAL TOLERANCES:
LINEAR DIMENSIONS (Inches): .125, .125, .125, .125, .125, .125, .125, .125
ANGULAR DIMENSIONS (Degrees): .125, .125, .125, .125, .125, .125, .125, .125
MINIMUM SURFACE FINISH: 1000 microinches

PART/ASSEM NAME: PULLEY_8MM	CHECKED BY: Z.GARRARD
PART/ASSEM NUMBER: -	APPROVED BY: C.WOOD
MATERIAL: ALUMINUM	DATE APPROVED: 4/28/2018
FINISH: -	SHEET SCALE: 2:1
	SHEET NUMBER: 1 of 1



NOTE: 6MM DIAMETER SHAFT

SPECIFICATIONS AND TOLERANCES

UNLESS OTHERWISE NOTED
DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009

DEFAULT DIMENSIONAL TOLERANCES:
LINEAR DIMENSIONS (Inches): X+.5, X+.5, X+.5, X+.5, X+.5, X+.5, X+.5, X+.5
ANGULAR DIMENSIONS (Degrees): X+.5, X+.5, X+.5, X+.5, X+.5, X+.5, X+.5, X+.5
MINIMUM SURFACE FINISH: 1000 microinches

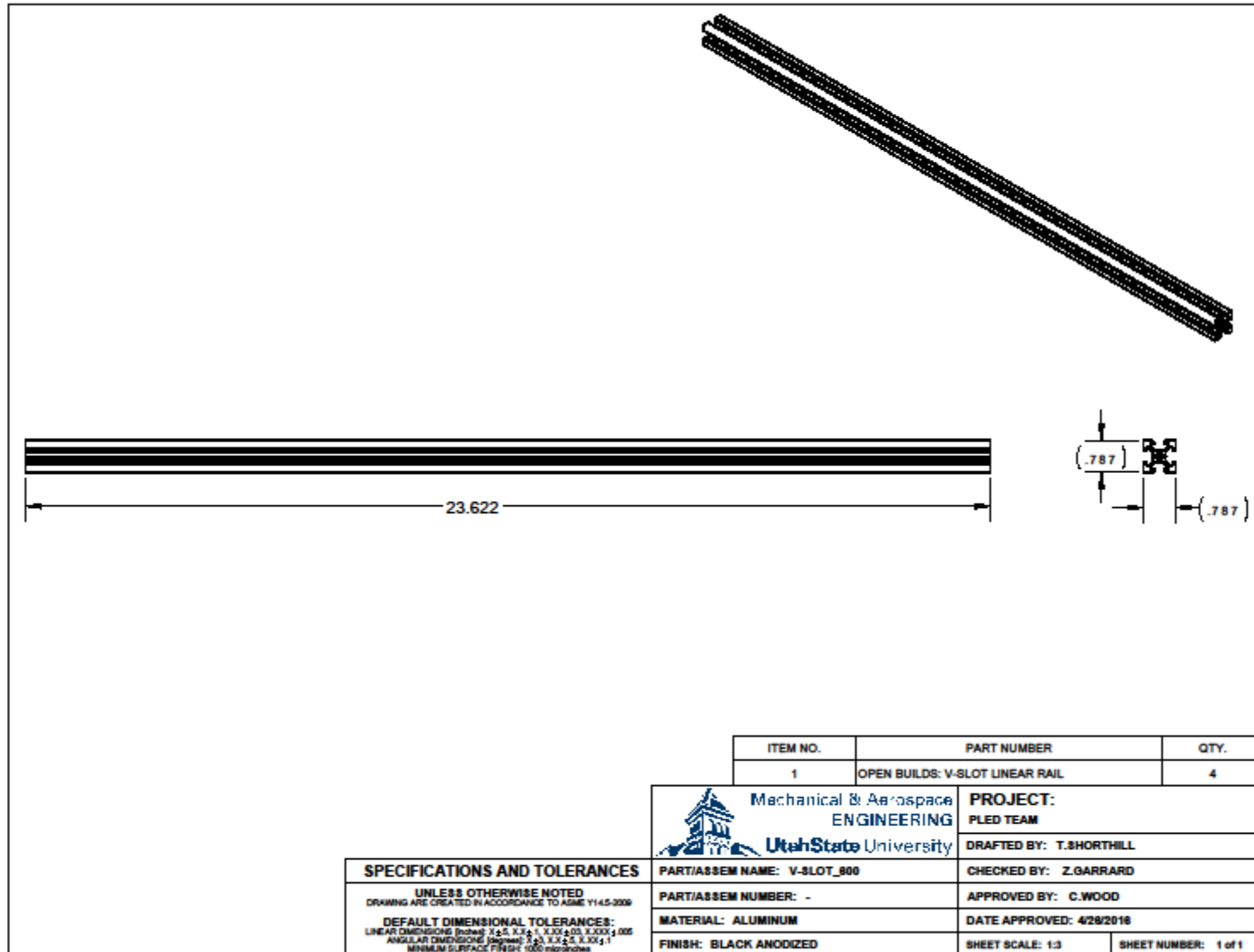


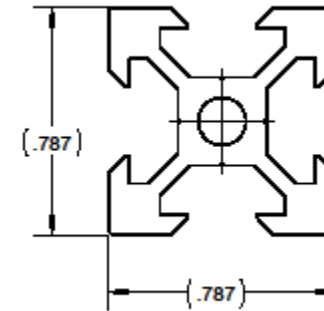
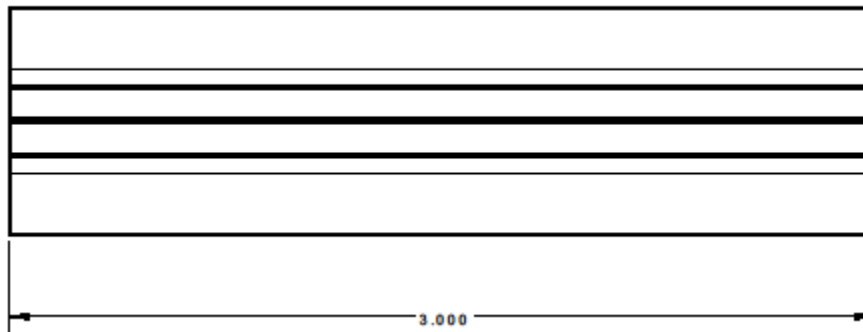
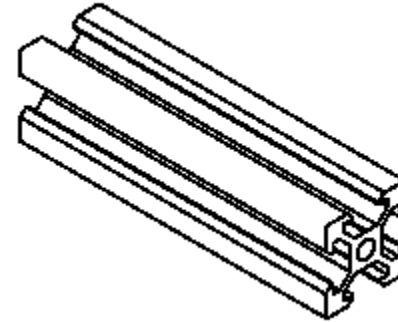
Mechanical & Aerospace
ENGINEERING
Utah State University

PART/ASSEM NAME: SHAFT_6mm
PART/ASSEM NUMBER: -
MATERIAL: 316 STAINLESS (OR EQUIVALENT)
FINISH: -


PROJECT: PLED TEAM
DRAFTED BY: T.SHORTHILL
CHECKED BY: Z.GARRARD
APPROVED BY: C.WOOD
DATE APPROVED: 4/28/2018
SHEET SCALE: 1:3
SHEET NUMBER: 1 of 1

ITEM NO.	PART NUMBER	QTY.
1	SHAFT_6mm	1





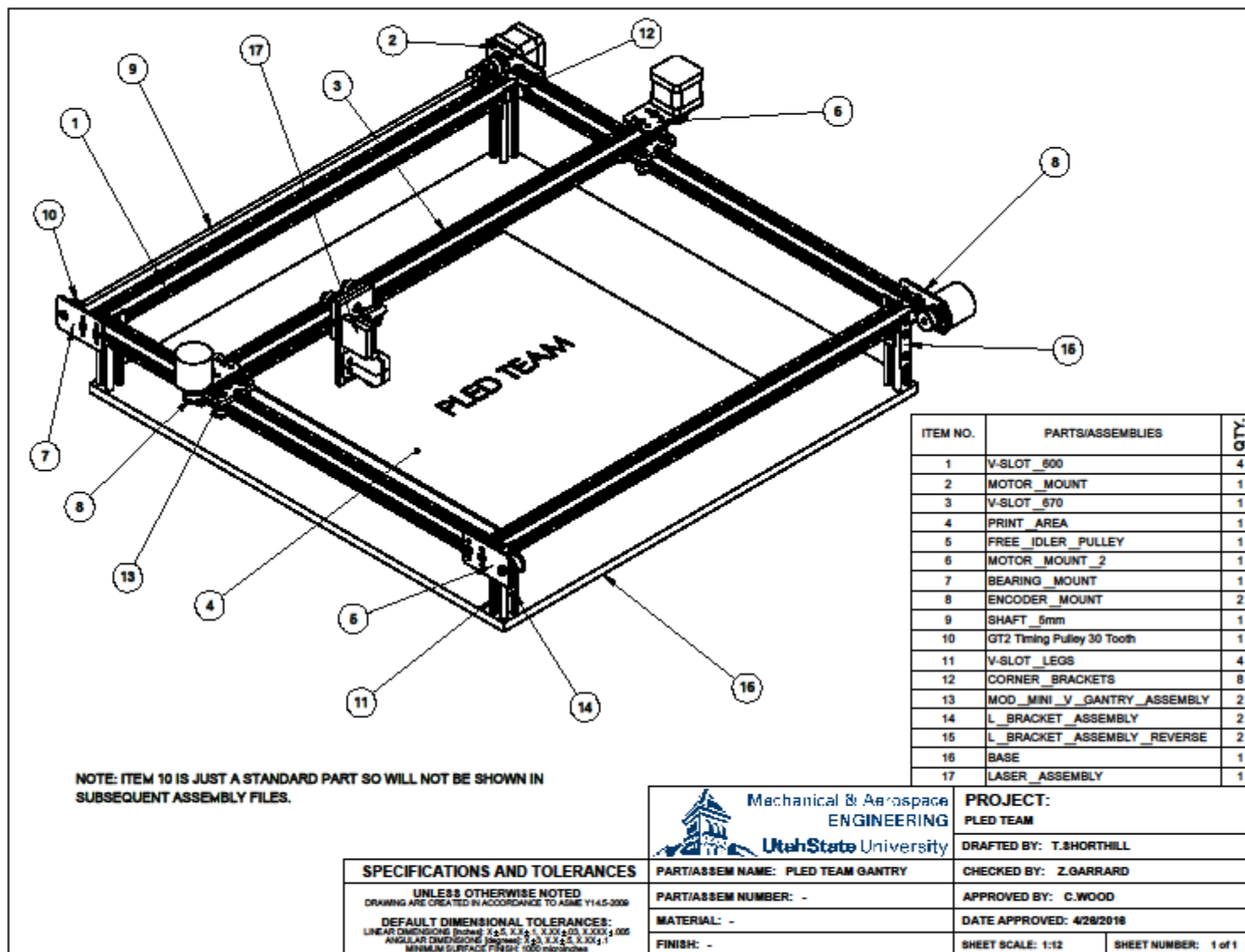
ITEM NO.	PART NUMBER	QTY.
1	OPEN BUILDS: V-SLOT LINEAR RAIL	4

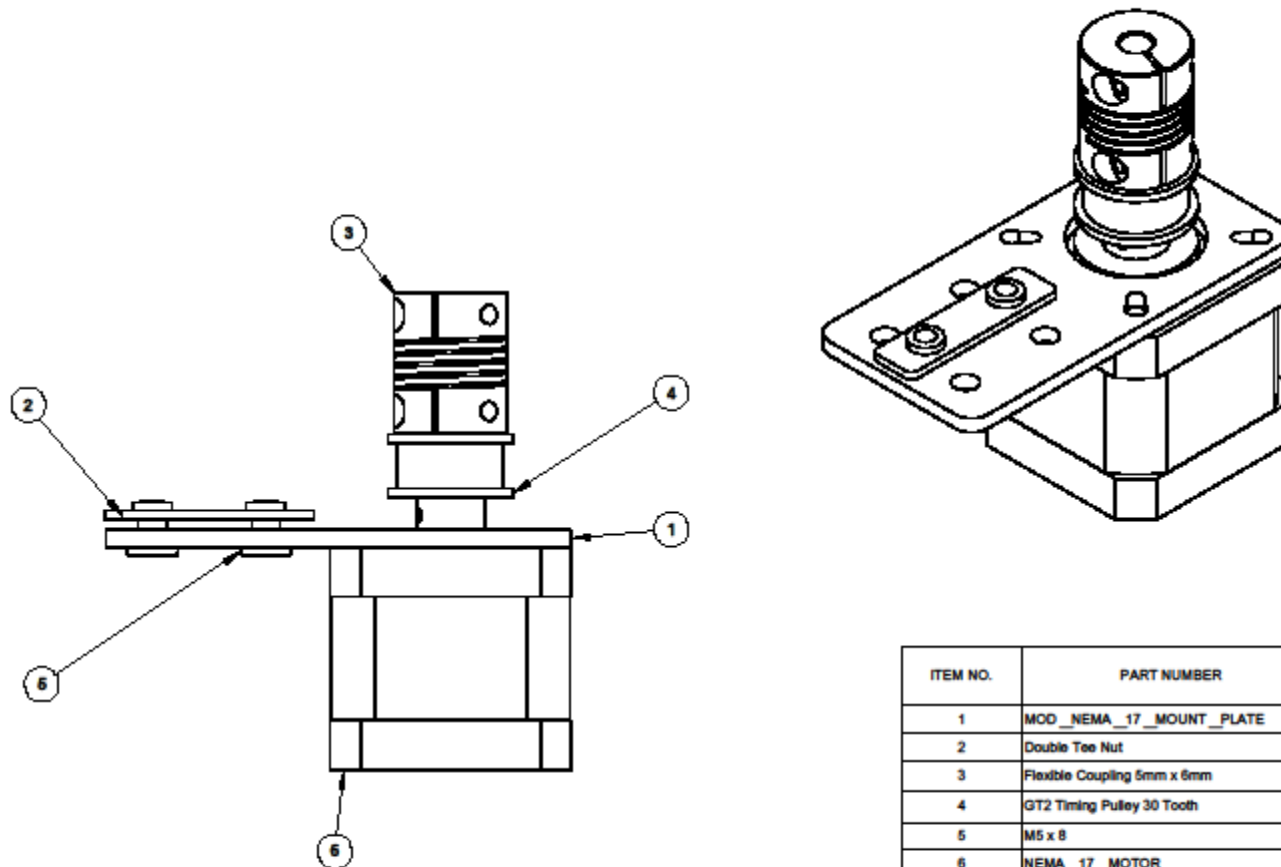
 Mechanical & Aerospace ENGINEERING Utah State University	PROJECT:	
	PLED TEAM	
	DRAFTED BY: T.SHORTHILL	
	CHECKED BY: Z.GARRARD	
PART/ASSEM NAME: V-SLOT_LEGS		APPROVED BY: C.WOOD
PART/ASSEM NUMBER: -		DATE APPROVED: 4/28/2018
MATERIAL: ALUMINUM		SHEET SCALE: 2:1
FINISH: BLACK ANODIZED		SHEET NUMBER: 1 of 1

SPECIFICATIONS AND TOLERANCES
UNLESS OTHERWISE NOTED DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009 DEFAULT DIMENSIONAL TOLERANCES: LINEAR DIMENSIONS (Inches): X.X, X.XX, X.XXX ±.005, X.XXX ±.002 ANGULAR DIMENSIONS (Degrees): X.X, X.XX ±.1 MINIMUM SURFACE FINISH: 1000 microinches

9.8 Appendix 8 – Assembly

On the following pages, the assembly instructions for the Plaque Laser Engraver Device can be found.





ITEM NO.	PART NUMBER	QTY.
1	MOD_NEMA_17_MOUNT_PLATE	1
2	Double Tee Nut	1
3	Flexible Coupling 5mm x 6mm	1
4	GT2 Timing Pulley 30 Tooth	1
5	M5 x 8	2
6	NEMA_17_MOTOR	1



PROJECT:

PLED TEAM

DRAFTED BY: T.SHORTHILL

CHECKED BY: Z.GARRARD

APPROVED BY: C.WOOD

DATE APPROVED: 4/28/2018

SHEET SCALE: 1:2

SHEET NUMBER: 1 of 1

SPECIFICATIONS AND TOLERANCES

UNLESS OTHERWISE NOTED
DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009

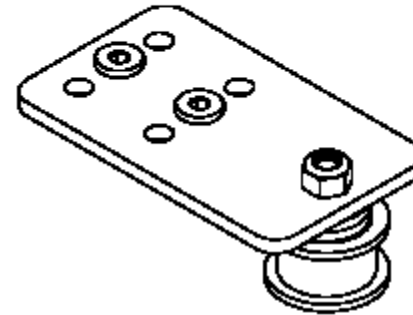
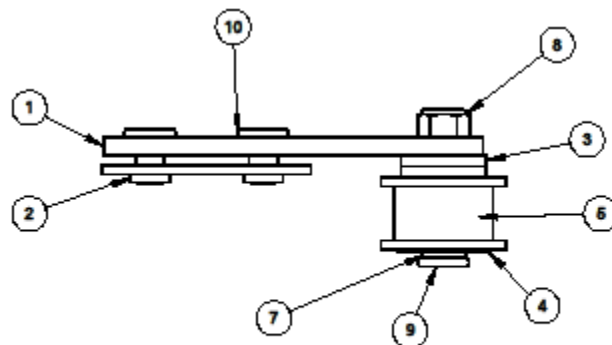
DEFAULT DIMENSIONAL TOLERANCES:
LINEAR DIMENSIONS (Inches): X.X, X.XX, X.XXX ±0.005, X.XXX ±0.002
ANGULAR DIMENSIONS (Degrees): X.X, X.XX ±1
MINIMUM SURFACE FINISH: 1000 microinches

PART/ASSEM NAME: MOTOR MOUNT

PART/ASSEM NUMBER: -

MATERIAL: -

FINISH: -



ITEM NO.	PART NUMBER	QTY.
1	Idler Pulley Plate	1
2	Double Tee Nut	1
3	Slot Washer 15 x 5 x 2	2
4	Ball Bearing 5 x 16 x 5	2
5	Smooth Idler Pulley Wheel	1
6	Nylon Spacer 0.125in	1
7	Precision Shim 8 x 5 x 1	1
8	Nylon Insert Lock Nut M5	1
9	M5 x 25	1
10	M5 x 8	2



Mechanical & Aerospace
ENGINEERING
Utah State University

PROJECT:

PLED TEAM

DRAFTED BY: T.SHORTHILL

CHECKED BY: Z.GARRARD

APPROVED BY: C.WOOD

DATE APPROVED: 4/28/2018

SHEET SCALE: 1:1

SHEET NUMBER: 1 of 1

SPECIFICATIONS AND TOLERANCES

UNLESS OTHERWISE NOTED
DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009

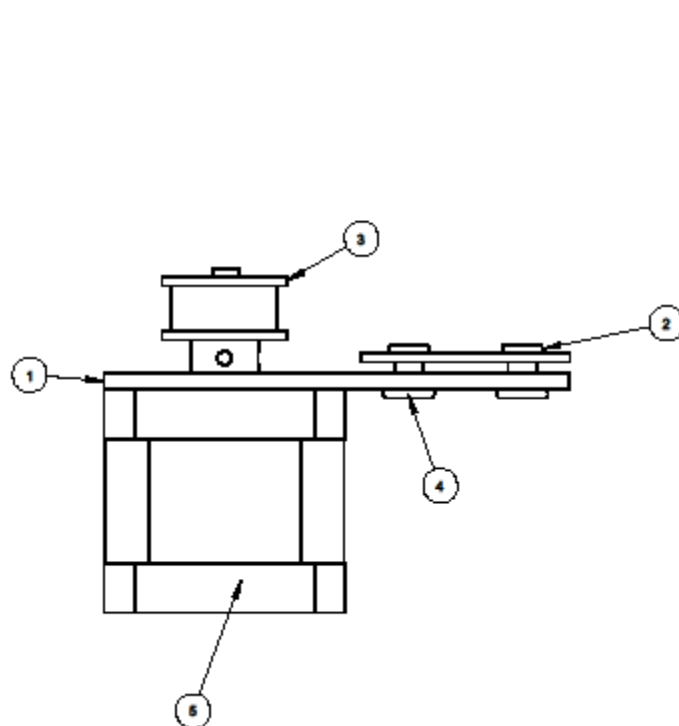
DEFAULT DIMENSIONAL TOLERANCES:
LINEAR DIMENSIONS (Inches): .125, .125, .125, .125, .125, .125
ANGULAR DIMENSIONS (Degrees): .125, .125, .125, .125
MINIMUM SURFACE FINISH: 1000 microinches

PART/ASSEM NAME: FREE IDLER PULLEY

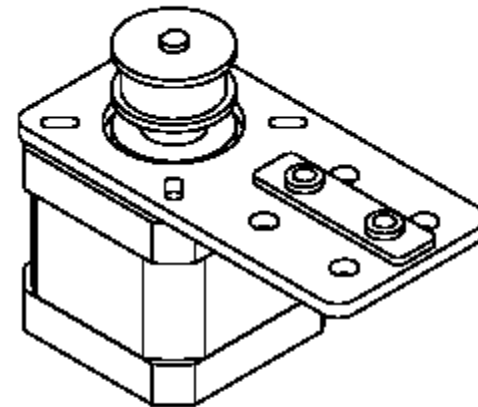
PART/ASSEM NUMBER: -

MATERIAL: -

FINISH: -



NOTE: M3 SOCKET HEAD CAP SCREWS REQUIRED FOR MOUNTING NEMA 17 MOTOR TO PLATE.



ITEM NO.	PART NUMBER	QTY.
1	Motor Mount Plate Nema 17	1
2	Double Tee Nut	1
3	GT2 Timing Pulley 30 Tooth	1
4	M5 x 8	2
5	NEMA_17_MOTOR	1



Mechanical & Aerospace
ENGINEERING
Utah State University

PROJECT:
PLED TEAM

DRAFTED BY: T.SHORTHILL

CHECKED BY: Z.GARRARD

APPROVED BY: C.WOOD

DATE APPROVED: 4/28/2018

SHEET SCALE: 1:2

SHEET NUMBER: 1 of 1

SPECIFICATIONS AND TOLERANCES

UNLESS OTHERWISE NOTED
DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009

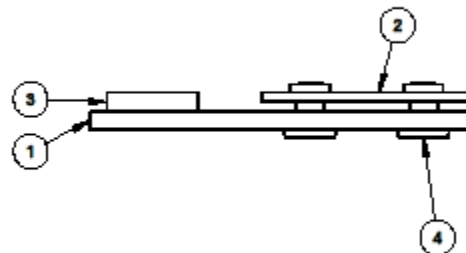
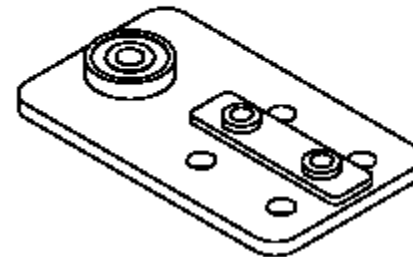
DEFAULT DIMENSIONAL TOLERANCES:
LINEAR DIMENSIONS (Inches): X.X, X.XX, X.XXX ±0.005, X.XXXX ±0.002
ANGULAR DIMENSIONS (Degrees): X.X, X.XX ±1, X.XXX ±1
MINIMUM SURFACE FINISH: 1000 microinches

PART/ASSEM NAME: MOTOR MOUNT 2

PART/ASSEM NUMBER: -

MATERIAL: -

FINISH: -



ITEM NO.	PART NUMBER	QTY.
1	BEARING_PLATE	1
2	Double Tee Nut	1
3	Ball Bearing 5 x 16 x 5	1
4	M5 x 8	2



Mechanical & Aerospace
ENGINEERING
Utah State University

PROJECT:
PLED TEAM

DRAFTED BY: T.SHORTHILL

CHECKED BY: Z.GARRARD

APPROVED BY: C.WOOD

DATE APPROVED: 4/28/2018

SHEET SCALE: 1:1

SHEET NUMBER: 1 of 1

SPECIFICATIONS AND TOLERANCES

UNLESS OTHERWISE NOTED
DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009

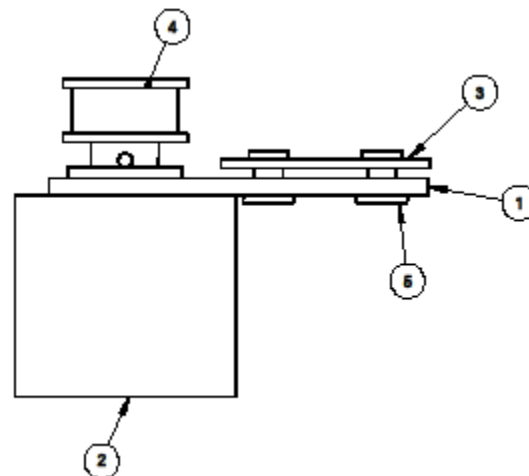
DEFAULT DIMENSIONAL TOLERANCES:
LINEAR DIMENSIONS (Inches): X.X, X.XX, X.XXX ±0.005, X.XXX ±0.0025
ANGULAR DIMENSIONS (Degrees): X.X, X.XX ±1, X.XX ±0.5
MINIMUM SURFACE FINISH: 1000 microinches

PART/ASSEM NAME: BEARING MOUNT

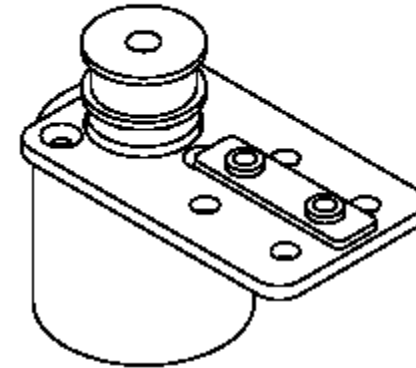
PART/ASSEM NUMBER: -

MATERIAL: -

FINISH: -



NOTE: 3M FLAT HEAD SCREWS ARE NEEDED TO MOUNT
ENCODER TO THE PLATE.



ITEM NO.	PART NUMBER	QTY.
1	Encoder_Plate	1
2	ENCODER	1
3	Double Tee Nut	1
4	PULLEY_6MM	1
5	M5 x 8	2



Mechanical & Aerospace
ENGINEERING
Utah State University

PROJECT:

PLED TEAM

DRAFTED BY: T.SHORTHILL

CHECKED BY: Z.GARRARD

APPROVED BY: C.WOOD

DATE APPROVED: 4/28/2018

SHEET SCALE: 1:2

SHEET NUMBER: 1 of 1

SPECIFICATIONS AND TOLERANCES

UNLESS OTHERWISE NOTED
DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009

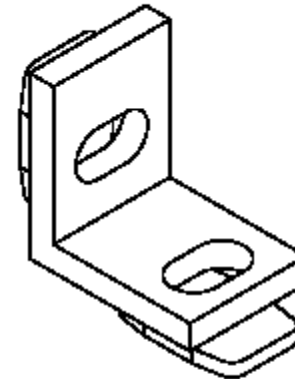
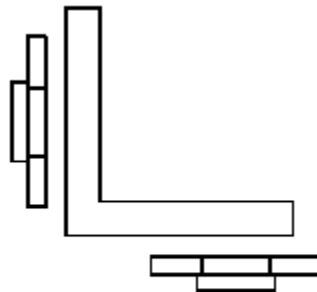
DEFAULT DIMENSIONAL TOLERANCES:
LINEAR DIMENSIONS (Inches): X.X, X.XX, X.XXX ±0.005, X.XXX ±0.002
ANGULAR DIMENSIONS (Degrees): X.X, X.XX ±1
MINIMUM SURFACE FINISH: 1000 microinches

PART/ASSEM NAME: ENCODER MOUNT

PART/ASSEM NUMBER: -

MATERIAL: -

FINISH: -



NOTE: THESE ARE ASSEMBLED WITH M6 SCREWS 8-10 mm LENGTH

ITEM NO.	PART NUMBER	QTY.
1	L Bracket Single	1
2	Two Nut	2



Mechanical & Aerospace
ENGINEERING
Utah State University

PROJECT:
PLED TEAM

DRAFTED BY: T.SHORTHILL

CHECKED BY: Z.GARRARD

APPROVED BY: C.WOOD

DATE APPROVED: 4/28/2018

SHEET SCALE: 2:1

SHEET NUMBER: 1 of 1

SPECIFICATIONS AND TOLERANCES

UNLESS OTHERWISE NOTED
DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009

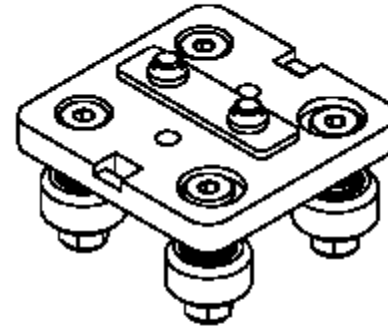
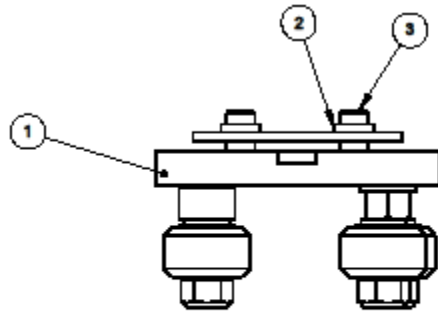
DEFAULT DIMENSIONAL TOLERANCES:
LINEAR DIMENSIONS (Inches): X.X, X.XX, X.XXX ±0.005, X.XXX ±0.002
ANGULAR DIMENSIONS (Degrees): X.X, X.XX ±1, X.XX ±0.5
MINIMUM SURFACE FINISH: 1000 microinches

PART/ASSEM NAME: CORNER BRACKET

PART/ASSEM NUMBER: -

MATERIAL: -

FINISH: -



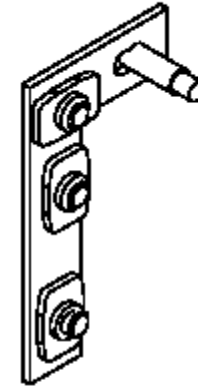
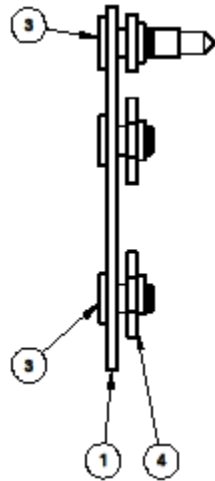
ITEM NO.	PART NUMBER	QTY.
1	MOD_MINI_V_GANTRY	1
2	Double Tee Nut	1
3	M5 x 10	2



Mechanical & Aerospace
ENGINEERING
Utah State University

PROJECT:
PLED TEAM
DRAFTED BY: T.SHORTHILL

SPECIFICATIONS AND TOLERANCES	PART/ASSEM NAME: MOD_MINI_V_GANTRY_ASSEMBLY	CHECKED BY: Z.GARRARD
UNLESS OTHERWISE NOTED DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009	PART/ASSEM NUMBER: -	APPROVED BY: C.WOOD
DEFAULT DIMENSIONAL TOLERANCES: LINEAR DIMENSIONS (Inches): X.X ± .01, X.XX ± .005, X.XXX ± .001 ANGULAR DIMENSIONS (Degrees): X ± .5, X.X ± .1, X.XX ± .05	MATERIAL: -	DATE APPROVED: 4/28/2018
MINIMUM SURFACE FINISH: 1000 microinches	FINISH: -	SHEET SCALE: 1:1
		SHEET NUMBER: 1 of 1



ITEM NO.	PART NUMBER	QTY.
1	ZINC PLATED STEEL 2.5 INCH L- BRACKET	1
2	Self Tapping Screw	1
3	M5 x 8	3
4	Tee Nut	3



Mechanical & Aerospace
ENGINEERING
Utah State University

PROJECT:
PLED TEAM

DRAFTED BY: T.SHORTHILL

CHECKED BY: Z.GARRARD

APPROVED BY: C.WOOD

DATE APPROVED: 4/28/2018

SHEET SCALE: 1:1

SHEET NUMBER: 1 of 1

SPECIFICATIONS AND TOLERANCES

UNLESS OTHERWISE NOTED
DRAWING ARE CREATED IN ACCORDANCE TO ASME Y14.5-2009

DEFAULT DIMENSIONAL TOLERANCES:
LINEAR DIMENSIONS (Inches): X.125, X.125, X.125, X.125, X.125, X.125, X.125, X.125
ANGULAR DIMENSIONS (Degrees): X.125, X.125, X.125, X.125, X.125, X.125, X.125, X.125
MINIMUM SURFACE FINISH: 1000 microinches

PART/ASSEM NAME: L BRACKET ASSEMBLY

PART/ASSEM NUMBER: -

MATERIAL: -

FINISH: -

