ON-DEMAND LABEL PRODUCTION

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Biomedical Engineering

by
Robert Zimmerman
May 2019
COMMITTEE MEMBERSHIP

TITLE: On-demand Label Production

AUTHOR: Robert Zimmerman

DATE SUBMITTED: May 2019

COMMITTEE CHAIR: Lily Laiho, Ph.D.
Professor of Biomedical Engineering

COMMITTEE MEMBER: Robert Crockett, Ph.D.
Department Chair of General Engineering

COMMITTEE MEMBER: Richard Savage, Ph.D.
Dean of Graduate Education
ABSTRACT

On-demand Label Production

Robert Zimmerman

The production and approval process for the various labels used in clinical trials wastes significant time and resources through the need to outsource label production or rely on large reams of pre-cut label stock for each revision throughout the process. An in-house, on-demand label printing and cutting system is a potential remedy to this waste. Previous work by Cheadle et al. resulted in a functional electromechanical prototype of the label cutting aspect of this research, capable of rudimentary linear cuts. In this continued research, emphasis was placed on improved label cutting capabilities and creating PC control software for label design. Cutting operations were enhanced through the development of an algorithm for circular cuts, proportional motor control, and a prototype graphical user interface (GUI) for simple user control. The changes to cutting methods have improved linear cutting precision to an average of 0.00402-in (s = 0.00602-in, n=26) at minimum. The new method for circular cuts has an average precision of 0.04384-in (s = 0.01471-in, n=26). The target precision for cuts is 0.040-in, suggesting that linear cuts are satisfactory, but circular cuts must still be refined. The prototype user interface developed for this research is capable of driving the label cutting system through RS232 communication and exposes all functionality of the system to date. Overall, this research has enhanced the capabilities of the label cutting system significantly, but further work is required to realize a complete label production solution.
ACKNOWLEDGMENTS

Without the initial work of Lorne Stoops, Nathan Cheadle, Tony Wang, and their advisor Lee McFarland, the on-demand label cutting system would still be little more than idea. Their framework has been improved on and expanded, and the ongoing support from various contacts and advisors are to thank for much of this success.

Mr. Gerald Finken, former Chief Scientific Officer and Founder of Clinical Supplies Management, sponsored, critiqued, and guided the development of the project.

Dr. John Ridgely of California Polytechnic State University, San Luis Obispo provided invaluable support to the project with his mechatronics and embedded software development expertise.

Dr. Vladimir Prodanov of California Polytechnic State University, San Luis Obispo’s suggestions for the electrical systems helped steer the development of the limit switches, brake system, contrast sensor, and the redesign of the power system.

Ivan Real and Lucas Tintikakis, Electrical Engineering students from California Polytechnic State University, San Luis Obispo, worked in tandem with the author through part of this project, solved myriad electrical issues with the system and provided excellent input for future improvements.

Dr. Lily Laiho of California Polytechnic State University, San Luis Obispo organized the project for the interdisciplinary senior project class which allowed work to continue from where the original team left off and also facilitated interactions with Mr. Finken. Her support as an advisor for the whole of the project was invaluable to its success.
# TABLE OF CONTENTS

## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>ix</td>
</tr>
</tbody>
</table>

## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>x</td>
</tr>
</tbody>
</table>

## CHAPTER

1. **INTRODUCTION**
   
2. **BACKGROUND**

3. **DESIGN DEVELOPMENT**

   3.1 Design Requirements
   
   3.1.1 Electromechanical Design
   
   3.1.2 Label Cutter Firmware
   
   3.1.3 PC Control Software

   3.2 Electromechanical Design
   
   3.2.1 Motors
   
   3.2.2 Solenoid
   
   3.2.3 Motor Controllers
   
   3.2.4 Brake
   
   3.2.5 Photodiode
   
   3.2.6 Contrast Sensor
   
   3.2.7 Microcontroller
   
   3.2.8 Power Supply
3.3 Label Cutter Firmware ........................................................................................................22
  3.3.1 Overview of Task Structure ................................................................................24
  3.3.2 Overview of Drivers ..........................................................................................27
3.4 PC Control Software ..............................................................................................................27

4. DESCRIPTION OF THE FINAL DESIGN ........................................................................29
  4.1 Label Cutter Wiring ..........................................................................................................29
  4.2 Peripheral Board ..............................................................................................................31
    4.2.1 PCB Production ..................................................................................................31
  4.3 Version Control ..............................................................................................................34
  4.4 Label Cutter Firmware ......................................................................................................35
    4.4.1 FreeRTOS .............................................................................................................36
    4.4.2 Task Structure .......................................................................................................37
    4.4.3 Libraries and Drivers ............................................................................................39
    4.4.4 Overview of Alternate Build Environment .........................................................48
  4.5 PC Software ....................................................................................................................49
    4.5.1 Framework Discussion .........................................................................................49
    4.5.2 Libraries and Classes ............................................................................................54
  4.6 Documentation ................................................................................................................57

5. TESTING AND DESIGN VERIFICATION .......................................................................59
  5.1 Movement Calibration ....................................................................................................59
  5.2 Linear Cut Speed ............................................................................................................59
  5.3 Linear Cut Precision .......................................................................................................60
  5.4 Circular Cut Speed and Precision ..................................................................................60
  5.5 Results Analysis .............................................................................................................60
5.6 Label Edge Perpendicularity ................................................................. 61
5.7 Label Designer Verification ................................................................. 61
6. DISCUSSION ......................................................................................... 64
  6.1 Cut Speed and Accuracy Results ....................................................... 64
  6.2 CSM Label Designer ....................................................................... 66
7. FUTURE RECOMMENDATIONS ........................................................... 68
  7.1 Electromechanical Design ................................................................. 68
  7.2 Label Cutter Firmware ..................................................................... 69
  7.3 CSM Label Designer ....................................................................... 70
8. CONCLUSION ......................................................................................... 72
REFERENCES .......................................................................................... 73
APPENDICES .............................................................................................. 76
  APPENDIX A: Label Cutter Firmware Documentation ......................... 76
  APPENDIX B: Label Cutter Firmware Source Code ............................ 77
  APPENDIX C: Label Designer Software Documentation .................... 78
  APPENDIX D: Label Designer Software Source Code ....................... 79
  APPENDIX E: Experimental Data .......................................................... 80
LIST OF TABLES

Table | Page
--- | ---
TABLE I: Operational Requirements for Label Cutting System. | 10
TABLE II: Cutting Test Results | 61
TABLE III: CSM Label Designer Survey Results | 62
TABLE IV: Raw Data for X-axis Cuts | 80
TABLE V: Raw Data for Y-axis Cuts | 81
TABLE VI: Raw Data for Biaxial Cuts | 82
TABLE VII: Raw Data for Circular Cuts | 83
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIGURE 1: Final Concept for Computer-Guided Blade Developed by Cheadle et al</td>
<td>4</td>
</tr>
<tr>
<td>FIGURE 2: Final Concept for Label Cutting System, Developed by Cheadle et al</td>
<td>6</td>
</tr>
<tr>
<td>FIGURE 3: Task Diagram for the Overall Operation of the Label Cutting System</td>
<td>26</td>
</tr>
<tr>
<td>FIGURE 4: Connection Diagram for the Arduino Due and System Peripherals</td>
<td>30</td>
</tr>
<tr>
<td>FIGURE 5: Peripherals PCB produced for Real and Tintikakis</td>
<td>32</td>
</tr>
<tr>
<td>FIGURE 6: Rendering of Finalized PCB in EAGLE</td>
<td>32</td>
</tr>
<tr>
<td>FIGURE 7: Stages of PCB Production Process</td>
<td>33</td>
</tr>
<tr>
<td>FIGURE 8: Task Diagram for task_ui</td>
<td>40</td>
</tr>
<tr>
<td>FIGURE 9: Task Diagram for task_print</td>
<td>41</td>
</tr>
<tr>
<td>FIGURE 10: Task Diagram for task_xaxis</td>
<td>42</td>
</tr>
<tr>
<td>FIGURE 11: Task Diagram for task_yaxis</td>
<td>42</td>
</tr>
<tr>
<td>FIGURE 12: CSM Label Designer &quot;Designer&quot; Tab</td>
<td>50</td>
</tr>
<tr>
<td>FIGURE 13: CSM Label Designer &quot;Sheet View&quot; Tab</td>
<td>51</td>
</tr>
<tr>
<td>FIGURE 14: CSM Label Designer &quot;Options&quot; Tab, Serial Connection Menu</td>
<td>52</td>
</tr>
<tr>
<td>FIGURE 15: CSM Label Designer &quot;Options&quot; Tab, RS232 Terminal</td>
<td>53</td>
</tr>
<tr>
<td>FIGURE 16: CSM Label Designer &quot;About&quot; Tab</td>
<td>54</td>
</tr>
<tr>
<td>FIGURE 17: Documentation Classes Button</td>
<td>57</td>
</tr>
<tr>
<td>FIGURE 18: Documentation Class List</td>
<td>58</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

The current system for approving labels for prescription drugs used in clinical trials is inefficient and generates significant waste [1]. Labels for drug bottles, instrumentation, and a study’s associated labelling kits must often be approved before a study can begin, which means that these labels must also be produced before the study. Because of the strict standards surrounding labelling, however, outsourcing the printing and re-printing of adjusted labels or relying on large reams of pre-cut label stock introduces significant delays and costs into the approval process.

Given the current process, it is clear that the availability of a label production system that pharmaceutical companies could use in-house to print and cut labels for their clinical trials would be hugely beneficial in regard to time and waste, as it would eliminate the need to outsource this step or stockpile myriad sizes of pre-cut labels. Mr. Gerald Finken approached California Polytechnic State University, San Luis Obispo, with this concept in mind and mechanical engineering students Lorne Stoops, Nathan Cheadle, and Tony Wang took on the task of developing a prototype of the label-cutting portion of this system.

After the original team for the On-Demand Label Cutting System, Cheadle et al., concluded their work on the project, further refinement of the label cutting system was desired to enhance its cutting capabilities. With the guidance of Mr. Finken, it was determined that the most necessary improvements were the inclusion of circular cutting and increased overall precision for cuts.

Because the label cutting system is eventually intended to function as a solution for printing and cutting labels for pharmaceutical companies, Mr. Finken also expressed interest in selecting a printer for the labels and developing computer control software to simplify using the label cutting system. The end goal of the project was described as a
complete label production system with accompanying software, where hardware support, software support, and upgrades are the main source of long-term profit.
2. BACKGROUND

A majority of the research for the on-demand label cutting system prior to this endeavor has focused on the regulations surrounding clinical trials, a review of existing label production technology, mechanical design and analysis, and parts selection [2]. Cheadle et al. began their review of the Food and Drug Administration (FDA) guidelines for labelling in a clinical trial setting. This research concluded that the strict FDA processes and procedures introduce significant delays and costs due to the need to backstock pre-cut label sheets of various sizes and actively attend to the printing and cutting of each differently sized label, or outsource their printing entirely. Even after label production, FDA inspections require precise label composition. Sizing, markings, and durability are examined in great detail before these labels are approved and discrepancies result in reprinting.

Their review of current label cutting technologies discussed three main methods for production: stamping dies, computer-guided blades, and laser cutting. Stamping dies are expensive to produce and only allow the production of a single label size, but are very durable. Computer-guided blades are inexpensive to replace and can cut arbitrary shapes freely, but lack durability compared to other options. Laser cutters are expensive, but can cut arbitrary shapes freely and are much more durable than blades, though they can potentially require significant venting. Morphological matrices were employed to determine the attributes of each cutting method and these results were pared down after a review of feasibility, safety, and portability. This lead to the decision to design a system which used a computer-guided blade, a Cartesian coordinate scheme, and an externally applied force to drive cuts.

Further review of possible parts for this cutting method led to the design shown in FIGURE 1. It employs a glass cutting wheel driven by a solenoid to allow cuts through various label stock thicknesses. The glass cutting wheel is allowed to rotate freely along
the cutting plane and fixed in its vertical axis. This allows the head to radius label corners simply by changing the cutting direction.

FIGURE 1: Final Concept for Computer-Guided Blade Developed by Cheadle et al. Produced in Solidworks [2]. It utilizes a glass cutting wheel which is fixed vertically and allowed to pivot freely in the cutting plane. Force is applied by a solenoid connected to the wheel, allowing the dynamic force application and, thus, depth control for cuts.

The design and selection of the remaining system components by Cheadle et al. focused on ease of production and repair. Parts were required to be easily obtainable (pre-manufactured) or simply produced without computer numerical control (CNC) or welding. This lead to the creation of a system with a large, replaceable base plate for cutting, rollers to guide and tension the label sheet, a brake to stall the rollers during cutting, motorized rail and pulley assemblies to drive the cutting head, and the cutting
assembly itself. An acrylic case for the cutting area was also developed for increased safety and fans were included to provide ventilation for the solenoid if necessary. A computer model of this finalized design is shown in FIGURE 2.

The author worked in tandem with Cheadle et al. during the parts selection for electromechanical components and development of basic control firmware for the label cutting system, referred to as "mechatronics" in their research. This lead to the evaluation of a number of electronic components, discussed in the Design Development section, and the eventual selection of motors, motor controllers, a photodiode, the brake, a development board, and a power supply for the system. The firmware developed in this stage of the project allowed the label cutting system to perform a series of proscribed, linear cuts on startup.

The research undertaken by Cheadle et al. concluded in June 2013 and a patent on the system by Gerald Finken was published on August 28, 2014 [1], [2]. This work successfully established the need for and desired function of the cutting system, specified requirements for precision cutting, and produced a prototype of the system. The author continued research on the system to improve cutting precision and program circular cuts. It became apparent that increasing the precision of cuts could be achieved through several means: faster recognition cutting process' termination in software, implementing more advanced motor controls in software, switching to a motor type with inherently more precise control, or redesigning the motor circuitry to incorporate a larger resistance to absorb back-electromotive force (EMF) during braking. Switching the motor type was immediately ruled out, as the DC motors had not been fully tested. Redesigning the motor control circuitry was decided against, as the selected motor controller already incorporated a method for draining back-EMF. Faster recognition of cuts in software could be realized as a simple change in encoder interrupt service routines (ISRs), however, and was chosen as for implementation.
FIGURE 2: Final Concept for Label Cutting System, Developed by Cheadle et al.

Produced in Solidworks [2]. The base for the system is shown in yellow and the replaceable cutting plate is shown in red. The label sheet tensioning and guide rollers are located on both the intake and exit for label sheets. The cutting assembly and its associated rails and pulleys are depicted to the left side of the image. The acrylic case is the clear box surrounding the cutting area.
More advanced motor controls could be introduced in several ways: ramping speed downward near the end of cuts to prevent overshoot; or implementing proportional, integral, or derivative control. Both methods were determined to be feasible and pursued.

Implementing circular cuts involved first determining a method to calculate cutting coordinates, then a method for moving the cutting head with respect to those coordinates. Through software, circle coordinate calculation is possible through a number of means: lookup tables for various angles, direct calculation using squares and roots, or direct calculation using sinusoidal functions. Using a lookup table would potentially be the fastest solution, but could impact accuracy if values needed to be interpolated, so it was eliminated as an option. Between the remaining methods, calculation of coordinates with sinusoidal functions was chosen because it required less function calls. Because both methods rely on the use of Taylor series approximations in their implementations [3], simply using one call to sine or cosine was determined to be faster than the equivalent two squares and a square root.

Movement of the cutting head for circular cuts could be achieved through synchronous cuts in both axes, broken down into individual steps for each degree of change. This method was chosen because cuts should seldom be less than 90-degrees (as when a label's corner is radiused), and because development of a stepped cutting routine was necessary to cut diagonally already.

Based on research, development of dedicated PC control software for the label cutter was determined to be a huge undertaking. At a high level, it required developing a robust interface between a PC and the label cutting system, a method for communicating with the label printer that will eventually be chosen, a solid graphical framework for the user interface, and an aesthetically pleasing and capable user interface.
After some analysis of the task, it became obvious that a proof-of-concept application for interfacing with the label cutter could be developed using the RS232 communications protocol, which was natively supported by the development board selected for the project. Developing a GUI to overlay this protocol and provide basic user interaction would be possible through the use of any number of graphics and window libraries. It was decided that detailed aesthetics for the GUI would be out of the question, as functionality was not yet proven. Similarly, because printers compatible with the label paper and cutting system had yet to be researched, selecting and implementing a protocol for communication with a printer was not pursued.
3. DESIGN DEVELOPMENT

The scope of the on-demand label cutter project is large mainly due to the level of sophistication required to reach its desired specifications. Broadly speaking, the system can be broken down into three subprojects: the electromechanical design, the label cutter firmware, and the PC control software. Without any one of these three parts, it is apparent that the label cutter would either not function or prove inviable for use as an end-user product.

A majority of the electromechanical work was completed by Cheadle et al. in their pioneering work on the system. They mainly focused on the mechanical design of the label cutter (frame, rollers, pulleys, and the cutting head) and preparation for the mounting of electronic components, as well as their required wiring [2]. As such, starting point of the electromechanical design for this stage of the project was the selection of the core electronic components for the system (motors and motor controllers, brake, photodiode, microcontroller). The second step was to source the additional electronics required to interface each of the core components and to connect them such that their control was easily orchestrated.

The remaining subprojects, the label cutter firmware and the PC control software, have no previous work and their requirements needed to be finalized before their development began. The label cutter firmware must be designed to facilitate precise control over the label cutter, user-friendly interaction, and easily maintainable and expandable code.

Similarly, the PC control software must be designed for effortless control over the label cutter while abstracting the technical aspects of its operation; expandability for the entirety of the cutting system and new features (which will include printing at a later stage); as well as easily maintainable code.
3.1 Design Requirements

As discussed above, the work required for the on-demand label cutting system was divided into three subprojects to facilitate the development of meaningful requirements for each step. These requirements are outlined and discussed in their respective sections below.

3.1.1 Electromechanical Design

The original operational requirements for the on-demand label cutting system were determined by Cheadle et al. [2]. Their figures, shown below in TABLE I, were based on a review of similarly purposed products and a relative ranking of importance of each specification. Cutting speed was ranked highest at 100% importance, cutting tolerance at 97%, feed speed at 76%.

TABLE I: Operational Requirements for Label Cutting System. This table summarizes the requirements developed by Cheadle et al. and Mr. Gerald Finken, using the three metrics of highest importance: cutting speed, cutting tolerances, and feed speed [2].

<table>
<thead>
<tr>
<th>Specification</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Speed (minimum)</td>
<td>17 in/min</td>
</tr>
<tr>
<td>Cutting Tolerances</td>
<td></td>
</tr>
<tr>
<td>Label Edge Length</td>
<td>+/- 0.031 in</td>
</tr>
<tr>
<td>Label Edge Perpendicularity</td>
<td>+/- 0.040 in</td>
</tr>
<tr>
<td>Feed Speed (minimum)</td>
<td>2.25 in/min</td>
</tr>
</tbody>
</table>
In conjunction with Cheadle et al, additional requirements for the system’s components were established based on the goals listed above in TABLE I. These requirements were as follows:

- Use high speed, high power motors
- Solenoid must provide consistent and adequate cutting force
- Select robust motor controllers to complement high power requirements
- Photodiode must be accurate at high speeds
- System must include an emergency stop mechanism

Because the main purpose of the system is cutting labels, the motors and solenoid were a large concern in the initial design requirements, as they would dictate the maximum cutting speed and affect the tolerances of cuts, which are very close in importance to speed. As such, the motors and their controllers needed to be prioritized during the component selection process to ensure the basic functionality of the label cutter. Additionally, the solenoid used to apply force to the cutting head required special consideration as well, as it would dictate the depth and quality of cuts.

In order to satisfy the project’s initial goal of beginning cuts at specified locations on the label sheets, a photodiode that could react and function at high speeds was also essential. The addition of this sensor would allow separation of labels on the sheet based on a user’s preference, as it would allow detection of pre-printed positioning markings on the label sheet. As the label cutter is meant to provide in-house labelling options for pharmaceutical companies, this feature will save on paper used and afford customization of printing, cutting order, and label placement.

Finally, the emergency shut off mechanism should be included because, while the system should not be dangerous to operate given the acrylic cover that attaches to the base, unforeseen malfunctions may require the system to stop immediately and
disconnecting the power cable is not always an option. An emergency shut off option is also beneficial as the system is not intended for use by specially trained personnel and can still be classified as heavy machinery, so increasing the overall safety of the system is a distinct benefit.

3.1.2 Label Cutter Firmware

Because Cheadle et al. focused primarily on the mechanical design of the label cutting system, there was no previous work related to the firmware or the control system for each component. As such, the task of assessing requirements for the firmware and its development became the main focus for this research.

Developing the requirements for the system began with prioritizing the goals for its functionality: precise control, an intuitive user interface, and easily maintainable code. The highest priority for the firmware is ensuring precise cutting operations. This is because precision and speed are both expected by users and required to meet FDA specifications for pharmaceutical labels. Second, it must be simple to control all cutting operations via a user interface. The user interface for the label cutter is imperative to operating the system, so it must be designed in a way to expose all core functions and debugging features. A feature-rich interface with a solid backbone also aids developers in testing and diagnosing errors. Third, the code for the firmware must be easily readable, editable, and simple to upload to the system. As the label cutter is a prototype, its firmware will need to change frequently and eventually be passed to another team. High quality code and documentation, as well as a streamlined development and upload process will aid the current and future teams working on the project.

Using the above guidelines, the requirements for firmware development, listed below, were finalized:
● Facilitate interaction between electrical and mechanical components
● Provide stable operation under normal conditions and intelligently handle errors
● Allow simple interaction for complex control over the system
● Follow common coding standards for ease of maintenance and expansion
● Use version control to ensure consistent backups and promote collaboration
● Leverage common protocols and libraries where possible

These requirements encapsulate and expand on the system’s core needs, particularly for the programming aspect of the project. Using version control will facilitate tracking changes to the code, keeping all team members up-to-date with those changes, and allow for reverting potentially negative changes as well. Additionally, by incorporating standardized libraries and conforming to accepted protocols, especially in the realm of communications, the project development can focus more on original feature development than backend functionality.

3.1.3 PC Control Software

While discussing the future of the project with Cheadle et al. as well as Mr. Finken, the user experience with the label cutting system was discussed extensively. It was determined that a clean user interface that simplified printing and cutting labels was the far-field goal for PC software. The system will need highly orchestrated communication between the end-user, printer, and label cutter upon completion, otherwise the end-user could become frustrated while attempting to control the system.

It was decided that determining rough requirements for this software’s functionality and design, as well as a mock-up of the graphical user interface (GUI) would be a help to future workers on the project. While programming this software was initially considered beyond the scope of this project, it was noted that the foundations of the PC software could greatly influence design decisions within the label cutter firmware.
This requirement forces the team to consider development paths that are universal or adaptable enough to interface with other software.

Because of the similarity of the nature and goals of the label cutter firmware and PC software, many of their requirements are similar, but are slightly different based on the intended audience for the software. While the label cutter firmware is more geared toward assisting developers and potentially technicians, the PC software is intended for true end-users and companies who expect to purchase a system that simply works and is easy to use.

After considering the required functionality for the label cutter, as well as the necessity of maintainability and readability of the code, it was decided that several key design principles should be followed:

- Expose all label cutter features
- Able to accept and format all information required for FDA-compliant labels
- Simple for the consumer to set up, configure, and operate
- Provide a clean, responsive, and intuitive GUI
- Interact with user through runtime and diagnostic information
- Use common protocols and libraries where possible
- Use version control to ensure consistent backups and promote collaboration
- Maintain high level of readability and expandability
- Easy to update and maintain subscription to service

The main requirement for the GUI is to provide full-featured access to all components of the system that are necessary to create and inspect labels, as well as to monitor their progress through the process. In terms of design, this requires a high degree of polish and modularity given that the functions of three interdependent systems must be represented fully, consistently, and in an aesthetically pleasing fashion throughout the program. To support this development, emphasis must be placed on the
use of vetted libraries and frameworks to ensure a polished user experience. Version control and consistent, well-documented code will also promote collaboration and assist with future expansion and changes to the software.

3.2 Electromechanical Design

Selecting the proper electronics to interface with the label cutter’s mechanical system was the chief concern, as Cheadle et al. recruited the author to the project with the mechanical system in a mostly complete state. As such, the continuation of the design process required evaluating a wide range of components to determine the best fit for the functionality, longevity, and ease of repair of the label cutting system.

3.2.1 Motors

The selection process began with a review of motors to determine what type would be most beneficial to the system. The two logical choices, stepper motors and DC motors both have distinct advantages and drawbacks. Stepper motors can be highly precise and their exact position can be easily calculated based on the number of step pulses sent to the motor, and tend to have high torque and low speed. DC motors, on the other hand, do afford similarly high precision, as their encoders allow for counting pulses to determine their exact position and distance travelled, and they also afford much higher speed than stepper motors and offer high torque. Stepper motor control tends to be simpler, as a single pulse will advance the motor by a specified distance, where DC motors require a finer control method, as inertia in the rotor will cause it to overshoot the desired final position if it is not taken into account [4].

After evaluating the positives and negatives of each type of motor, it was decided that DC motors would provide more benefit to the system overall, as cutting speed is the requirement with the highest priority. Torque also is unlikely to be a concern, as torque
and speed are directly related for most DC motors and high torque is only required when the cutting head is in motion.

The Pololu-1447 DC gear motor was chosen for all three of the system's motors (x-axis, y-axis, and label feed motor) [5]. These motors run at 80RPM, provide roughly 15.6lb*in of torque, and operate at 12V with a maximum current draw of 5A. With the addition of the Mitsumi XL timing belts and timing pulleys chosen for each motor by Cheadle et al. [2], the maximum linear speed for each motor translates to roughly 150 in/min, which is nearly an order of magnitude larger than the required 17 in/min for axial motors and more than an order of magnitude greater than the 2.25 in/min requirement for the label feed motor.

3.2.2 Solenoid

Driving the cutting head with enough force to cut the label paper cleanly, but not the backing paper was a particular concern, as it is necessary for labels to meet FDA guidelines and still be easily peeled. After reviewing solenoids and testing the cutting head with controlled loads, Cheadle et al. determined that the ideal cutting force range was between 3.5lbs to 4 lbs [2].

Because of the incredible importance of the solenoid to the accuracy and usability of the system, Cheadle et al. chose the Magnetic Sensor Systems S-22-200-H solenoid for its ease of use under a wide range of stroke lengths while providing a cutting force of 3.5lbs to 4lbs [6]. This allowed Cheadle et al. to design the cutting head apparatus for a viable stroke length and left the tasks of determining the optimal duty cycle at the final designed stroke length and implementing the proper controls to produce adequate force for cuts.
3.2.3 Motor Controllers

Given the high power requirements for both the DC motors and solenoid, as well as their similar modes of operation and power requirements, it was sensible to explore the possibility of using motor controllers for both components to simplify both obtaining replacement parts and their control via microcontroller. To determine the feasibility of using motor controllers, the maximum voltage and current requirements for a wide range of DC motors and solenoids were explored.

Using the data collected by Cheadle et al. regarding solenoids and the ideal cutting force range of 3.5 to 4 lbs, it was determined that the maximum expected voltage for researched solenoids that fit these requirements was 12V and the maximum expected current was 2A. For the reviewed DC motors, which were included if their max speed was greater than 60RPM and if their torque was greater than 8.5 lb*in, the maximum expected voltage was 16V and the maximum expected current was 6A. This suggested that any motor controller designed to control at least 16V and 6A would be sufficient for both motor and solenoid control.

The Pololu MD01B, a carrier for the STMicroelectronics VNH2SP30 single axis motor driver, was chosen for control of the axis and feed motors, as it was capable of controlling motors with operating voltages between 5.5V and 16V and had a maximum output current of 14A [7]. This choice fit the operating requirements of both the motors and solenoid and allowed for simple control via pulse width modulation.

3.2.4 Brake

Ensuring consistent cutting quality was a chief concern with the cutting system and, because of the large forces required to penetrate the label paper, designing a brake onto the label feed input roller was a high priority. It was noted by Cheadle et al. that during their testing phases, the feed rollers took several iterations of materials to find a
non-marking surface that evenly fed the label paper [2], but after introducing the cutting
head and manually testing cuts, it was discovered that the input roller needed to be
locked to prevent slippage of the label paper as well.

The main factor in choosing the correct brake was the torque it could counteract.
Because the chosen motors for the system, Pololu-1447 DC gear motors, have a
maximum theoretical torque of 15.6 lb*in, it was determined that a brake with similar or
higher torque would be required to effectively stall the paper while the cutting head
was in motion.

After reviewing several industrial brakes and looking into coupling with the feed
motor shaft, the Anaheim Automation BRK-18H was chosen as the brake for the cutting
system. It requires 24V and 0.42A at full load, providing 15 lb*in of torque [8]. It was
decided that although the torque provided by the BRK-18H was less than that of the
Pololu-1447 motors, that the motors could be run at a lower speed to lower their torque
below the brake’s maximum and to potentially extend their useful lives without
compromising the system’s cutting speed significantly.

3.2.5 Photodiode

Detecting the proper location on the label paper to begin cutting labels was a
necessity to the project because it allows a simple method to position the cutting head
for multiple sizes of label stock without requiring the user to specify its exact position on
the cutting plate, only its size. Software would then calculate the effective cutting area
and position labels on the sheet accordingly. To facilitate alignment, it was originally
determined that a photodiode and IR diode pair would suit this need.

The IR diode and photodiode would be mounted on the cutting head and the IR
diode would illuminate the area below the photodiode. Over unmarked label paper, the
photodiode would remain open or highly conductive. When the photodiode encounters a
dark mark, which would be printed prior to the cutting operation, the IR light would not be reflected and thus the photodiode would close or become less conductive. Using this change in conductivity, it would be possible to determine where the label paper is marked and to properly cutting head based on the location of that mark.

Quickly detecting and responding to these markings was a high priority, as well as determining how large a mark would be necessary to reliably trigger a reaction in the photodiode. After researching photodiodes, it became apparent that the frequency response tends to be on the order of milliseconds, so nearly all photodiodes would be sufficient for this purpose [9].

As such, the Pacific Silicon PS0.25-7-TO52 was initially chosen as the photodiode for the system. Its typical response time of 1.5 ns, high angle of half intensity at 110-degrees, and high response to light between the wavelengths of 750 to 950nm make it ideal for detecting markings. It also allows the use of a wide variety of inexpensive IR diodes for lighting [10]. The IR diode chosen to illuminate the label paper was the Vishay TSHG6400, which outputs light at the PS0.25-7-TO52’s peak detection wavelength of 850nm and has a relatively small angle of half intensity, 22-degrees [11], making it ideal for focused light delivery, reducing the chance of early or erroneous mark detection.

3.2.6 Contrast Sensor

After a review of the photodiode and its potential utility in identifying marks on the label sheet, however, Real and Tintikakis determined that an industrial optical sensor would be fast enough for the project’s speed requirements, more accurate than a photodiode-based method, and easier to both code for and maintain [12].

To that end, the Banner Engineering D11E2N6FP contrast sensor and corresponding PBT46U fiber optic cable were chosen to replace the photodiode on the
on-demand label cutting system [13]. This resulted in a small change, the addition of a 0.125-mm hole to the center of the mounting bracket designed for the photodiode. The power requirements for the optical sensor peak at 0.015A from between 10V and 30V. Designing the circuit with the 24V rail for the brake, this brought the estimated power usage to 0.36W which fell well-within specification of the power supply. Additionally, the sensor’s output was restricted to the 0 to 3V range in order to interface with the Arduino Due - this was accomplished through a simple voltage divider circuit with an added bypass capacitor.

3.2.7 Microcontroller

Accommodating the diverse functionalities needed in the label cutting system required a microcontroller unit (MCU) capable of interfacing with inputs from a large number of peripherals, coordinating system functions based on sensor and user inputs, and outputting the corresponding actions to the peripherals. In order to facilitate this process, it was decided early on that a standardized prototyping board would serve this purpose while designing the label cutter, as each pin of the microcontroller is usually connected to a header on a prototyping board and many come with a simple interface for uploading firmware and interacting with PCs.

After reviewing the previously selected components, it was determined that at least 26 I/O pins would be required to adequately sense and control the label cutting system. It was decided that timing requirements for reliable operation, based on the need for accurate motor control and the photodiode, would be on the order of milliseconds. For ease of programming, a simplistic user interface for both editing code and uploading it to the MCU would also be preferable.

The Arduino Due was chosen for the project after evaluating of a number of prototyping boards [14]. It is based around an ATSAM3X8E ARM microcontroller,
clocked at 84MHz, and provides 12 analog and 54 digital inputs, as well as 2 analog and 12 digital outputs, and is programmable via a USB interface. Additionally, because the board runs on any input voltage between 7V and 12V, it would tie in easily with the power requirements of the other main system components. The high clock speed of 84MHz also allows 84,000 MCU cycles per millisecond, which equates to between 42,000 instructions per millisecond - this suggested that timing would not be an issue, even with the multiple sensors and peripheral that must effectively act simultaneously. Finally, because Arduino and its community have a vast collection of official and user-developed libraries, the project can leverage these libraries to drastically increase efficiency and reduce turnover time for new features.

3.2.8 Power Supply

It was necessary to choose a power supply for the label cutting system that would support the multiple voltages required for its various components to operate, as well as the overall wattage for each component. From the chosen parts, it was clear that the power supply would require 3.3V, 5V, 12V, and 24V outputs and a rated wattage of at least 215W.

Additionally, it was decided to add a safety factor of 20% to allow for additions to the system and to prevent the power supply from running at full capacity constantly, and to account for the expected aging of X7R capacitors in the power supply over a period of 10000 active hours (about 1.1 years), which would diminish their efficiency by roughly 10% [15]. This suggested that the ideal safety factor for power supply wattage was roughly 30%, thus the selected supply should be able to output at least 280W.

An Apevia ITX-AP250W (250W output) computer power supply was initially chosen to test the system, as it met all requirements for output voltages (+12V, +5V, +3.3V, and -12V) and was provided a quick and inexpensive solution for prototyping.
Additionally, computer power supplies have a dedicated line for powering on and off, so testing the emergency shut off switch would be a simple task.

After verification of the system’s functionality, a dedicated power supply and voltage regulator solution was suggested and implemented by Real and Tintikakis to cater better to the system’s needs [12]. As detailed in their report, they selected the Mean Well SP-200-12 and RD-35B for their respective +12V (16.7A) and dual +5V (4A) and +24V (1.3A) outputs. In addition to these power supplies, the author added a Texas Instruments LM1086ISX-3.3V, a +3.3V (1.5A) regulator, to eliminate the need for voltage dividers for the system’s limit switch debouncers and to accommodate future sensors.

3.3 Label Cutter Firmware

Because the label cutter’s firmware is necessary for controlling the system’s individual components, such as its motors, brake, cutting head, and photodiode, developing or leveraging a solid framework for control over each device and overall system timing was necessary. Having chosen the Arduino Due as the prototyping board for the label cutter, a wide variety of libraries and frameworks were available to the project, as Arduino boards have a high level of first and third party support [16], [17]. The strict timing requirements for fast operation while producing precise cuts in multiple axes suggested that simulated multithreading or a state machine architecture would be required. This need is amplified with the number of components which require simultaneous action.

A review of the system’s intended operation suggested that the maximum number of components required to function simultaneously would be the five required during a standard cutting operation: the x-axis motor, the y-axis motor, the brake, the solenoid, and the communications interface.
To briefly explain the concepts of multithreading and finite state machines, they are both methods of coordinating control of a system that has multiple elements competing for resources at the same time. Multithreading takes the approach of assigning priority to tasks and switching between them based on the values of internal flags or signals, known as semaphores. Finite state machines compartmentalize various functions into states and cycle through each state in a loop, using flags to determine the required actions within each state before moving to the next. As such, both methods, can be used to simulate concurrent operation of multiple functions and assign priorities to those functions.

In order to accelerate the development process, it was decided that leveraging a well-supported multi-tasking framework would benefit the project greatly. After reviewing a number of potential frameworks packaged for the Arduino Due, FreeRTOS was chosen for its combination of multithreading and finite state machine elements, its liberal commercial license, and the author’s previous experience with it [18]. Using FreeRTOS as a backbone for the project allowed the author to focus on determining which operations should be grouped into tasks and their required timings, then directly move to programming those tasks rather than building and debugging an original framework. In addition to the increase in productivity, using an established library simplifies code maintenance by allowing drop-in updates if they are ever necessary.

Having decided the general framework and structure for the project’s code, it was necessary to determine the most effective version control solution for the project. Version control can be achieved through regularly archiving project files, but a more robust solution was deemed appropriate, as a potentially large number of people may need to simultaneously work on the project in the future. File synchronization services, such as the ubiquitously known Dropbox, were quickly ruled out because they are convenient for smaller teams, but do not allow for tracking specific changes or have
good systems in place for concurrent file editing, which poses a problem when two team members are working on separate parts of the same task. After a review of several version control systems, Subversion was chosen for the project, as it provides a simple platform for uploading code and comments on changes, informing all users of changes, coordinating simultaneous edits through a simple merging system, and allowing for simple reversions when unintended changes are made [19]. Also, because Subversion repositories are not proprietary, they can be hosted in house for increased security and regularly backed up to ensure valuable comments and contributions are not lost due to hosting system failures or migrations.

With the decision of a project framework and version control system, the design of individual tasks could be addressed based on FreeRTOS’ functionality and limitations. This involved grouping similar functions into logical tasks, deciding their required timings and priorities, and determining when standard libraries should be used in place of original code.

3.3.1 Overview of Task Structure

Before proceeding with designing individual functions for the label cutter, it was necessary to determine the overall task structure. This involved deciding which operations each task would control, their required timing, and how they would interact with each other to produce cuts as desired and interact with the user. After some deliberation with Cheadle et al., a task diagram, shown in FIGURE 3, was produced.

The task diagram shows the absolute minimum coordination required for each task to communicate. As such, it gives a clear vision while designing each task, which helps to ensure that development is goal-focused and will result in a functional product. Additionally, task diagrams abstract the code out of the system function, which eases the process of explaining overall system functionality to those unfamiliar with the project.
To briefly explain, semaphores are a data type often used as a variable to convey data between multiple simultaneous processes and control access to that data. In FreeRTOS, there are multiple types of semaphores which can function as event flags and counters [20]. Very generally, the access of data stored in semaphores is limited by “take” and “give” functions, which lock and unlock the data within the current task, respectively. When locked, a semaphore must be unlocked before another task or process can access the resource. In the label cutter code, semaphores are used mostly as flags to signal when events have taken place, such as cut completion, queued text for display, etcetera.

Efficient task timing is essential to ensure the label cutting system operates as intended. This can be achieved by prioritizing tasks that provide core functionality (such as positioning and cutting) and slowing down less important tasks (such as the user interface). Deciding the specific interval for timing for a particular task has three main factors: the response times of the sensors involved, the response times of tasks that must work in tandem with the task in question, and the microcontroller clock speed. Because of the Arduino Due MCU’s high clock speed, however, process bottlenecking was not observed or programmed around for this project. This allowed task timing to be split very intuitively between the cutting tasks for each axis and the UI tasks. Cutting tasks were both given the same timing, 10 ms, based on the decision to implement cuts in small step increments and the need for both and x and y cuts to keep a consistent ratio and quick pace. UI tasks were given the same timing as well, 50 ms, to ensure a relatively speedy user interface for input and output without taking significant processing time away from cutting operations.
FIGURE 3: Task Diagram for the Overall Operation of the Label Cutting System.

It describes each task with its function, what system elements it controls, and its required timing for reliable operation. Signals that coordinate actions between tasks are shown as arrows leading from the originating task and ending at the affected task. All signals are shown with their data type and number of elements, if relevant. Semaphores are denoted with dashed lines, where all other data types are denoted with solid lines.

Each task is subdivided into individual states to simplify and separate their operation. A task’s state is determined based on user input, process completion, or sensor values. This provides a cascading set of rules for each task to follow, which provides three distinct benefits: first, the code is easy to compartmentalize and develop.
in a modular fashion when functions are distinct from one another; second, debugging is much easier when states and substates can be printed for diagnostic purposes, which speeds development; third, code maintenance and later additions are simplified considerably, as the code is clearly separated by function.

3.3.2 Overview of Drivers

In order to better separate the label cutter’s framework from its specific functions, libraries to govern those functions were created. These libraries, called drivers because they drive the action of individual components or processes, are often classes designed to expose the full functionality of sensors that interface system and classes which implement new, specific functionalities purely in code.

A large number of libraries were sourced from external sources to speed development. The initial choice of an Arduino platform for the label cutting system brought a wealth of well-tested libraries for core system functionality (math, interrupt handling, serial communications, and advanced data types, among others) to the table. And, as Arduino products are rather ubiquitous in the do-it-yourself and “maker” communities, third-party support and libraries are also abundant.

Drivers created specifically for this project fell into two main categories: user interface (UI) and component control. The UI drivers focus on improving serial communications and buffering the text for display. The component drivers are designed to control the various sensors and actuators required for the label cutting system to operate as intended.

3.4 PC Control Software

While the electromechanical design and firmware development for the on-demand label cutting system were the main goals of this project, the business plan
involved selling the label cutting system and licensing the control software. As mentioned in the Design Requirements, developing a proof-of-concept control application was a goal which its sponsor, Gerald Finken, intended for a future iteration of the project. It became apparent during planning the label cutter’s firmware that considering and designing for a future connection with this software was necessary, however, which lead to an in-depth examination of its most important requirements and the decision to create a proof-of-concept program.

It makes a great deal of sense to lower the technical barrier of entry to use of the label cutting system to ensure a high adoption rate in the target audience, so an intuitive user interface is a must. Developing the core functionality of the program, however, is the highest priority. As such, interfacing with the label cutting system is the first goal, followed by allowing access to its various features.

Visual appeal and ease of use are more important for an end product than a prototype, though they should not be entirely neglected while developing the application. Rather than devoting significant time to presentation, the GUI should be coded in a way that integrates as much functionality as possible while organizing these functions and their associated UI elements for ease of future editing.

Documentation is crucial for this portion of the project, as it will serve not only to educate the development team, but it is likely to contribute a significant amount to the user documentation and experience as well. For example, configuration steps for the compiled program can be directly borrowed from the source documentation. Known bugs and quirks with the program, if well-tagged, can be used as guidance for future patches.
4. DESCRIPTION OF THE FINAL DESIGN

The sections below detail the design of the prototype label cutter at the project’s completion. This includes descriptions of physical layouts and code organization, rationality for design decisions, and requirements for setup and use of the project.

4.1 Label Cutter Wiring

Because the label cutting system has a number of peripherals and connections to various power sources, several steps were taken to ensure proper connections are made should the system require disassembly.

First, each wire that was not soldered into place was twisted into a group with wires from the same device and flagged at its termination. For example, the Feed Motor Encoder’s output A was twisted with all Feed Motor wires and labeled with the flag “Feed Enc A” to denote the signal it relays to the Arduino Due. This simplifies the identification for all wires, which is a tedious process when dealing with so many components.

Second, references to each wire and its pin were listed in the main project file, CSMLabelCutter.ino (see Appendix B, p. 2-3). Each software reference was named similarly to the physical labels for each wire connection and documented with a brief summary of the signal captured by the pin.

Third, a pinout diagram for the Arduino Due, originally created by Robert Gray, was altered to visually show the correlation between each signal wire and Arduino pin [21]. The finalized pinout diagram with all connections between the label cutter and its peripherals is included on the following page in FIGURE 4.
FIGURE 4: Connection Diagram for the Arduino Due and System Peripherals.

Connections between the label cutting system’s peripherals and the Arduino Due listed in blue. Arduino-assigned pin numbers are listed next to the header representations; special MCU functionalities are listed in white bubbles along the pinned-out line; CSM label cutter connections are shown in rectangular blue highlights at the end of relevant pinned-out lines. Diagram sourced from Robert Gray [21] and edited to fit the needs of this project.
4.2 Peripheral Board

The circuitry developed by Real and Tintikakis in [12] was originally tested on a personal breadboard, which did not translate well to permanent inclusion in the project. They later produced a perfboard version of the circuits as well, which allowed the board to stay with the label cutter, but it was decided that mounting a prototyping board was not ideal.

To that end, CadSoft’s EAGLE was used to design a PCB specifically for the contrast sensor, limit switches, 3.3V voltage regulator, and brake control circuitry. This reduced the footprint for the board, allowed the use of surface mount components (which reduced the cost of several parts), and reduced the complexity of maintenance through the inclusion of on-board text labelling parts, their values, and connection points.

A circuit board was produced in-house using 1.2mm double-sided FR-4 board through dry-film photo etching and the application of a dry-film solder mask. Vias and mounting holes were drilled with a press. The final etched and drilled board is shown in FIGURE 5. Additionally, the EAGLE rendering of the front of the PCB is shown in FIGURE 6.

4.2.1 PCB Production

The PCB production process can be summarized as follows. Using the solder mask, text, and pad layers for each side of the board from EAGLE, light masks were printed and produced. A piece of the FR-4 board slightly larger than the final circuit board was cut, and a dry-film negative photo resist was applied to both sides of the PCB with a laminator. The top and bottom layer pad light mask pieces were aligned then taped together to prevent movement, and the cut FR-4 board was placed between the sheets. The light-masked board was placed in a UV exposure box with a glass plate flattening both sides for the time required to properly expose the photo resist.
FIGURE 5: Peripherals PCB produced for Real and Tintikakis. Materials used were 1.2mm FR-4 board, dry-film photo resist and solder mask.

For this particular photo resist and UV box, 8 minutes was sufficient exposure to harden the resist. The board was removed from the UV box and scrubbed in a soda ash solution to remove unexposed resist and reveal the copper for etching. A photo of the board in this state is shown in FIGURE 7A. The board was then placed in a ferric acid solution until the traces and pads were properly etched (for the copper thickness on these boards, at room temperature, 30 to 45 minutes was sufficient). After etching, the board was rinsed, then the remaining photo resist was removed by further soaking and scrubbing in a soda ash solution. A photo of the board in this state is shown in FIGURE 7B.
Next, the dry-film solder mask was applied to both sides of the board with the laminator. As with the pad light masks, the top and bottom solder mask and text layer light masks were aligned and taped together to prevent movement. The circuit board was then placed between them. To prevent movement, the circuit board was first aligned to the masks (aligning to vias made this simple), and the board was subsequently taped to the mask sheets. Once again, both sides of the masked board were flattened with glass plates and placed in a UV box for the time required to harden the exposed mask (for the mask used, roughly 10 minutes was sufficient). The board was removed from the UV box and allowed to sit in the soda ash for an additional ten minutes to soften the unexposed solder mask, scrub the board, and rinse it. This left the vias exposed and text for parts identification and placement clearly visible on the board. A photo of the board in this state is shown in FIGURE 7C.

**FIGURE 7: Stages of PCB Production Process.** A) Photo resist applied, exposed, excess resist removed. B) PCB etched and photo resist removed. C) Solder mask applied, exposed, excess mask removed.
A drill press was used to drill all through-holes and vias. Pads and vias were pre-tinned with MG Chemicals 421 Liquid Tin and vias were then connected by soldering strands 0.5-mm unshielded copper wire in their through-holes.

### 4.3 Version Control

As noted in the Label Cutter Firmware section of Design Development, Apache Subversion was chosen as the version control software for the project. This allowed each member of the project access to up-to-date versions of all projects files through a central repository, the ability to work on the same files semi-simultaneously, as well as the ability to examine and revert to previous file versions as necessary.

Version control in any form is an important aspect of this project, as the firmware and software for the label cutting system have the potential to become exceedingly convoluted. As this project evolves into a commercial product, Subversion will allow for branching the code base to develop features outside of the production software while still maintaining synchronization between all team members associated with the project. It will also greatly facilitate bug hunting when changes break functionality, and the backup process for project files. Version control is not a backup in itself, just a history of changes, but the repository can be condensed to a single file for backup purposes.

Deveo, now absorbed by Perforce and rebranded as Helix TeamHub, was chosen as the host for the Subversion repository for the wide breadth of features available to free, private repositories. Beyond repository hosting, the platform also allows for issue tracking, wiki integration, and milestone marking [22]. While these features were not used with such a small team, they will likely become increasingly relevant as the scale of the project and team size increase.

In order to download the project source, a Subversion client must be installed (such as TortoiseSVN on Windows, or kdesvn on Linux) and a directory to contain the
source must be created. Clients can be either GUI or command-line based, but GUI-based environments are far more convenient for newer users.

To continue the checkout process, open the Subversion client, select the "Checkout" option, and enter the repository URL as well as proper login credentials. Additionally, for simplicity, disabling the "append source URL name to subfolder" option during checkouts is advised, as it reduces the directory complexity substantially. This is often an option on the Checkout window of a GUI or by specifying the folder path explicitly during a command line checkout.

After the Checkout operation completes, a local copy of the most up-to-date project source is available in the folder designated during checkout.

4.4 Label Cutter Firmware

Firmware describes the programming that controls the hardware for a specific device. In the case of the on-demand label cutting system, this includes a task-based operating system and various extensions for interacting with the peripherals. The label cutter’s firmware is written in C++ and, for simplicity in interfacing with the Arduino Due, the team chose to use the official Arduino IDE [14]. This not only simplified coding, but also allowed for rapid testing of new code, as the IDE incorporated an Upload function for compiling and flashing binaries to Arduino-based boards.

The firmware code, tailored to the Arduino IDE, is located in the project’s CSMLabelCutter directory. The main project file, CSMLabelCutter.ino, can be opened in the Arduino IDE (version 1.5 or greater) and uploaded after transferring the project’s libraries (located in the CSMLabelCutter/libraries subfolder) to the Arduino libraries folder. The correct Port and Board (Arduino Due) must also be selected within the Tools menu in the IDE, otherwise errors will occur during the upload.
For further information regarding the code and its usage, see the documentation Appendix A, as well the source in Appendix B.

4.4.1 FreeRTOS

FreeRTOS is an MIT-Licensed real time operating system kernel developed by Real Time Engineers Ltd. and now maintained by Amazon Web Services [18]. It offers a robust, well-documented kernel for microcontroller-based devices through software-timed task scheduling and the availability of mutexes and semaphores for atomic data access.

Using FreeRTOS within the Arduino environment has two important ramifications on the structure of the code: first, FreeRTOS uses the Arduino setup() function as the main() function would be used in standard C and C++ programs. This means that any code that must be run before FreeRTOS starts its task scheduler (such as creating task objects, giving or taking semaphores, or initializing variables) must be done within setup() and above the call to vTaskStartScheduler().

Second, FreeRTOS uses the Arduino loop() function as the FreeRTOS idle task loop. In FreeRTOS, the idle task generally runs when no other task is running - meaning all tasks are blocked or delayed. Given that it is occasionally called, it is imperative that the idle task does not tie up the RTOS with critical code sections or MCU-cycle-intensive operations. As such, it is left as a while(1){} loop in the current version of the code, though it can be used for garbage collection operations and the like if they become necessary in the future.
4.4.2 Task Structure

As described in FIGURE 3, the label cutter’s code is separated into tasks that govern main system functions - these tasks communicate with each other, relaying important information through shared data and C++ objects.

In their most basic form, tasks in FreeRTOS must be given a name, they must have a stack set aside for themselves, they must have a priority, and they must be tied to a function that works similarly to a standard C or C++ main() function. For the purposes of the label cutter, a C++ wrapper for FreeRTOS tasks was created to afford the option of dynamically creating and deleting tasks with the standard C++ new() and delete() methods, as well as creating pointers to specific task objects when necessary.

The constructor to the task wrapper, named TaskClass, takes in a character string as a name, a priority, and a stack size. Each child class of TaskClass is also required to override its pure virtual run() method, which will serve as the task’s pseudo-main() function. It is worth noting that if the child class’ run() function returns, the task is scheduled for deletion within FreeRTOS via TaskWrap’s (TaskClass’s parent class) destructor.

With this class hierarchy, task objects follow the general paradigm of being instantiated with relevant data or pointers, saving that data in their constructors, then waiting for the FreeRTOS task scheduler to start their run() methods.

Fairly precise timing of the tasks can be achieved with the delayms() method built in to TaskClass. It suggests to the RTOS that the task which calls it should be delayed for the interval passed to the function or longer. If more precision is required, the delay_from_for() method can be called, which takes in the current number of ticks registered by the RTOS and gives it a strong suggestion to wake up the task in the number of milliseconds specified by the interval. This tends to be more precise than
delayms(), but tasks of higher priority will still block execution if they request microcontroller time before the task that calls delay_from_for().

4.4.2.1 Task_ui and Task_print

The user interface for the label cutter is displayed and manipulated via terminal over a standard RS232 serial connection emulated over USB. The two tasks that work in tandem to create this interface are task_ui (the logic behind the menu and system interactions) and task_print (a wrapper for all text output via the USB-RS232 interface).

task_ui, visualized in FIGURE 8, uses a state machine architecture to divide its various functionalities and simplify the navigation experience for the user. After connecting to the label cutter via the USB-RS232 interface, the main menu presents options for Diagnostic Tests, Motor Control, and Cutting Tests. The Diagnostic Tests menu allows the user to print out system uptime as well as the task stack/heap information. The stack usage is particularly useful in determining if a task is operating in an undefined manner because it has run out of memory. Second, the Motor Control menu allows manual adjustment of all motor-related systems. This allows the user to arbitrarily re-zero the position counts for the encoders without changing the motor positions, print out current encoder counts, and stop the motors from running. Third, the Cutting Test Menu allows the user to test all cuts that the label cutting system can perform, as well as request feedback from the system regarding deviation from expected positions. Possible actions include enabling and disabling diagnostic information on cut precision and timing, calibrating the cutting head position and re-zeroing it based on a printed marking on the label paper, performing straight and circular cuts, feeding the label paper manually, zeroing accumulated cutting position error, enabling and disabling the cutting brake, and testing motor speed. The final option in the main menu simply re-
prints the menu to the terminal. After any individual action is completed within the UI, the current state (submenu or main) is retained, but the task delays itself for at least 50ms.

`task_print` uses a simple conditional check to determine its action. If the serial text queue (TextQueue) has stored any text for output to the terminal, then it is taken out, character by character, and transmitted to the terminal. When the text queue is empty, the task will delay itself for 50ms before resuming and checking the queue again. This process is shown in FIGURE 9.

4.4.2.2 Task_xaxis and Task_yaxis

`task_xaxis` and `task_yaxis`, shown pictorially in FIGURE 10 and FIGURE 11 respectively, control the behavior of the label cutter in their respective axes and are nearly identical. Upon their construction, they instruct the motors for the axes which they control to automatically zero to the system's limit switches. This ensures a consistent start point and usable sheet area for label cutting. After the axes are zeroed, the tasks enter their main state, which simply calls the CuttingDriver's `checkXCut()` or `checkYCut()` method to check the encoder position and ramp motor speed as necessary.

4.4.3 Libraries and Drivers

The functionality of the label cutter hinges on a number of external libraries and component- and operation-specific drivers. This separates code for the framework, components, and particular functionalities from the task logic, allowing for simpler debugging and code maintenance. Each library and project driver is discussed briefly in the following sections.
FIGURE 8: Task diagram for task_ui. It describes functions and options for each of the task's four states. The user input variable, input, is used to transition between states. Directionality of transitions is indicated by arrows. STATE_MAIN is a hub state which connects to the other states. STATE_DIAG displays diagnostic information about the system. STATE_MOT allows control over the motor-related aspects of the system. STATE_CUT affords control over cutting-related operations.
FIGURE 9: Task diagram for task_print. The printing task is composed of a printing and a waiting state. TextQueue's checkWaiting() method signals the transition from printing to waiting, and the transition from waiting to printing is made automatically after 50ms.
FIGURE 10: Task diagram for task_xaxis. The task has an initialization state and a main loop. After initialization is completed, signaled by xAxisComplete, the task enters its main state. This state checks the status of pending cuts in the x-axis, waits 10ms, then loops.

FIGURE 11: Task diagram for task_yaxis. The task has an initialization state and a main loop. After initialization is completed, signaled by yAxisComplete, the task enters its main state. This state checks the status of pending cuts in the y-axis, waits 10ms, then loops.
4.4.3.1 Arduino Standard Libraries

The Arduino standard libraries are a set of libraries for simplifying the control of Arduino prototyping boards. They are developed by Arduino AG and licensed under LGPL 2.1. Common operations for embedded systems (port configuration, communications, timing, interrupt control, etcetera) are vastly simplified through the use of these libraries and, as such, they can potentially reduce the time required to prototype a project substantially by abstracting microprocessor-specific details from the coder [17].

Use of these libraries in the code for the on-demand label cutting system allowed much more emphasis to be placed on function-specific code than the framework and also reduced the complexity of code that could potentially need replacing should the microprocessor for the label cutter change later in development.

4.4.3.2 Library: FreeRTOS_ARM

FreeRTOS_ARM is an MIT-Licensed Arduino Due port of FreeRTOS, created by Bill Greiman [23]. It wraps and configures FreeRTOS for drop-in use with projects built with the Arduino IDE. Leveraging this port rather than creating one specific to the Arduino Due and capable of running within the Arduino environment saved time for feature development when, as mentioned above, the MCU may change with future iterations of the project.

4.4.3.3 Class: DirectDigitalIO

The DirectDigitalIO library implements the direct method for changing the input and output status of pins on the Arduino Due, which is much faster than the standard libraries allow. Because the standard libraries are meant to support a wide variety of platforms, several checks are performed before each bit flip, which slows the process down considerably. This library simply bypasses the checks in the standard library by
wrapping the Atmel Software Framework (ASF) method that the Arduino framework leverages to change port values instead.

It is important to speed up input and output operations for digital pins because they directly impact the speed with which peripherals such as the motors respond to user requests. As such, they directly impact cutting precision through increased speed and are less likely to be interrupted by a task shift in FreeRTOS.

4.4.3.4 Driver: MotorDriver

Motor and solenoid actuation are controlled with the MotorDriver class, which is designed to allow interaction between STMicroelectronic’s VNH2SP30 H-bridge motor controller and a generic microcontroller. Upon creating a MotorDriver object, the user must specify a numeric identifier for the motor, as well as the Diagnostic (DIAG), Pulse-Width Modulation (PWM), and input pins (INA and INB) from the microcontroller. These definitions are used in methods which control a motor’s spin direction and speed.

The Diagnostic pin is used to either enable output to the motor or the diagnostic mode for the VNH2SP30. It should be noted that there are two DIAG pins (DIAGA and DIAGB) on the VNH2SP30, but they are tied together for the purposes of this software, as all output and diagnostic options are available when both pins are simultaneously pulled high or low.

The Input pins (INA and INB) allow the selection of various modes of output for the motor controller. There are four possible controls: spin clockwise, spin counterclockwise, “soft” brake, and “hard” brake. The spin modes enable spinning in the selected direction at a rate dictated by the input on the controller’s PWM pin. The “soft” brake mode essentially connects the motor output to ground, allowing it to spin down without resistance. The “hard” brake mode connects the motor output to VCC, which forcibly prevents spinning.
The Pulse-Width Modulation (PWM) pin dictates the speed at which the motor spins when clockwise or counterclockwise spinning is enabled. The PWM pin provides an analog voltage input to the VNH2SP30. This voltage, in combination with the input pins, instructs the motor controller to provide the connected motor with an output voltage proportional to the PWM input in the desired direction.

4.4.3.5 Driver: EncoderDriver

Each motor selected for the label cutting system also includes a quadrature encoder, which is a combination of two signals which pulse as a motor spins. These pulses are identical, but 90-degrees out of phase, which allows for four different states. By counting these pulses, both the direction of spinning and the motor's position can be tracked.

An EncoderDriver object for a particular motor is created with two input pins (pinA and pinB) and a pointer to its corresponding MotorDriver object. It handles both keeping track of the motor's current and desired positions, as well as the interrupt service routine (ISR) required to determine the motor's position as it spins.

The main functionality of the EncoderDriver class rests in the encoderInterrupt() method, which functions as the ISR for both input pins. Once this method is tied to an interrupt and instructed to fire on a pin state change, it runs automatically, each time the motor has moved far enough to trigger a new pulse. The logic in the ISR method simply checks the new and old states of each pin to determine the current direction of spin, increments the position counter (or error counter if a change was missed), and stops the motor if it has reached the desired position.
4.4.3.6 Driver: LimitSwitchDriver

The system’s limit switches are controlled by the LimitSwitchDriver class. Upon its construction, a LimitSwitchDriver object takes an input pin as well as reference to a particular motor’s MotorDriver and its EncoderDriver. The LimitSwitchDriver’s intended function is to stop the motor and zero the encoder position whenever a limit switch is triggered.

This functionality is achieved by attaching an interrupt to the limit switch’s input pin and triggering it on a rising edge. Because the limit switches are normally low, they only read high values when they are activated. When a high is read on the pin, the MotorDriver instructs the appropriate motor to “hard” brake and the EncoderDriver zeros its position and error count.

4.4.3.7 Driver: CuttingDriver

Nearly all of the logic required for making cuts with the label cutting system is contained in the CuttingDriver class. A pointer to each of the system’s MotorDrivers (including the MotorDriver used for the solenoid), each EncoderDriver, and the BrakeDriver are required to create a new object. This class allows control over cuts through various status checks, calibration routines, performance tests with CSV-formatted output, speed ramping, and logic for single-axis linear, multiple-axis linear, and circular cuts of arbitrary length and degree.

To briefly explain the cutting logic, the user’s input for a desired cut length and angle, where applicable, is stored. For straight cuts along a single axis, the length is translated into a number of encoder counts and the cut proceeds. For diagonal cuts, the length is also converted to encoder counts, then the slope of the line is determined and used to instruct the cutter to make a series of small, stepped cuts which add up to the desired length. For circular cuts, each degree of change is calculated as a stepped cut.
(they are cut as a series of diagonal lines) based on the user’s input radius and desired number of degrees and direction. During each cut the label feed brake is applied and it is subsequently released as a cut ends.

4.4.3.8 Driver: BrakeDriver

The BrakeDriver class interfaces with the circuitry designed by Real and Tintikakis [12] to allow control of the label feed brake which is used to provide tension during cuts. It provides a simple on-off signal to the brake through an opto-isolator, a device used for optically transferring signals between two isolated circuits.

4.4.3.9 UI Control

Handling the large amounts of text from different, simultaneously-running tasks and forming them into a cohesive user experience required the creation of a specialized text output class. Simply using the putchar() and puts() methods available from the Arduino Standard Libraries would have provided output, but that output is not thread safe and, thus, can be interrupted as tasks switch. If two or more tasks attempted to send terminal messages at the same time, the likely result is a mess of characters rather than the intended output of each task.

To prevent simultaneous writes to the terminal, FreeRTOS’ mutexes, semaphores, and queues were leveraged. A FreeRTOS queue sized to the char data type is created to store outgoing strings from all classes. A FreeRTOS mutex is used to lock access to the queue when a task is accessing it, then subsequently unlock access. This forces tasks to stall or “wait” until the lock is released, allowing other tasks to run while they wait to write to the queue.

In addition to text queuing operations, the TextQueue class also overloads the << operator, allowing its objects to function similarly to the C++ standard Stream library.
This was a choice inspired by Dr. John Ridgely and his prior work with FreeRTOS, as it was noted that it significantly improves the readability of the task code and reduces its overall complexity by allowing the compiler to choose the proper method for sending data to the terminal.

4.4.4 Overview of Alternate Build Environment

The Arduino Due used in this project is a prototyping platform. This suggests that in the future two main changes must be made to the on-demand label cutter before commercial production is viable: first, a project-specific circuit board should be produced (preferably one that incorporates all features of the Arduino board, the motor drivers, the switch debouncers, and the brake-control circuitry), and, second, much of the project’s source code will need to be rewritten, especially if the chosen microcontroller’s architecture varies significantly from the Arduino Due’s SAM3X8E.

In order to ease the transition away from the Arduino platform, an effort was made to create a build environment for the On Demand Label Cutter’s firmware which was independent of the Arduino IDE. The manifold advantages of building the source in a different environment begin by inverting the negative aspects of retaining Arduino compatibility: the code can be more complex and better organized. In addition, it reduces the compiled source’s size considerably by removing unused Arduino libraries and functions and allows a more streamlined build and uploading process that uses Makefiles to compile and upload code in a single command.

A majority of the work required to set up an alternate build environment for the Arduino Due, and thus other boards of similar architecture, has been completed by the author and is available to download at GitHub through the use of Subversion or Git [24]. The project is also fully ported to the Altrino Due environment, which is included in the altrino-due folder of the project files. If, in the future, the project continues to rely on the
SAM3X8E or any similar Atmel ARM microcontroller, using this environment will considerably reduce the time required to port the current code. Details on its setup and use are located in the folder’s README.md file.

### 4.5 PC Software

The computer software developed as a proof-of-concept for a label design and cutting interface was named “CSM Label Designer.” It satisfies the basic requirements of exposing all current label cutter functionality, providing a graphical user interface for cutting operations, and storing and manipulating required label information.

As with the label cutter firmware, Apache Subversion was used for version control for the label designer code. Pre-compiled versions of the Windows and Linux software and their required libraries are included in the attached project files. For further information regarding the code and its usage, see the documentation Appendix C, as well the source in Appendix D.

#### 4.5.1 Framework Discussion

The proof-of-concept CSM Label Designer program is functional on Windows and Linux systems (and was specifically tested on Windows XP, 7, 8, and 10 as well as Ubuntu LTS releases 14.04 through 16.04). It allows the user to add labels to a queue for cutting, shows previews of the labels on a scaled representation of the cutting plane, allows editing of already made labels, controls cutting of user-defined labels, delivers progress messages, and also contains a fully functioning RS232 terminal for complete control of the label cutting system.

The starting page for the label designer software is the "Designer" screen, shown in FIGURE 12. It contains entries for relevant information to prescription drug labels, such as patient and prescription names, instructions, and label sizes. A preview of the
label size and orientation is also provided. When the entries are complete, the user can click “Add Label to Sheet” to add the current label to the list on the right, as well as the label sheet preview. Editing labels is possible through clicking the “Edit” button on an entry in the list. Removing entries is possible through the “X” button on a particular entry.

**FIGURE 12: CSM Label Designer "Designer" Tab.** This is the main tab for label design. The user is able to enter a name for the current label, its size, information regarding the prescription, and add the label to the cut queue. Previously completed labels are listed on the right side of the screen and editable, if necessary.
Previewing the labels queued to cut is possible on the second tab, Sheet View, shown in FIGURE 13. This screen provides a scaled preview of all labels that the user has added to the queue as well as a button, “Send to Cutter”, which checks the connection with the label cutter and begins the cutting process. While the cutting occurs, a progress bar appears to alert the user of the number of remaining labels to cut.

![CSM Label Designer "Sheet View" Tab](image)

**FIGURE 13: CSM Label Designer "Sheet View" Tab.** It displays a preview of the label system’s full cutting area and organizes all user-generated labels on the cutting area. From this tab, the user can initiate cutting of all displayed labels.

The Options tab, shown in FIGURE 14 and FIGURE 15, allows the user to configure the RS232 serial connection if the label cutter is disconnected before starting the program or during its use. It simply scans a list of known port names upon pressing
the “Refresh” button and selects a port and baud rate upon pressing the “Set” button. This tab also contains the RS232 terminal, which is accessible through the keyboard shortcut Ctrl+T.

**FIGURE 14**: CSM Label Designer “Options” Tab, Serial Connection Menu. This menu allows the user to select the communication (USB) port to which the label cutter is connected and the baud rate for the connection.
FIGURE 15: CSM Label Designer "Options" Tab, RS232 Terminal. This is a fully functional terminal which can be used to access all menu-accessible features of the label cutting system.

Last, the About tab, shown in FIGURE 16, simply displays the Clinical Supplies Management logo, program information, and the license for the program.

These features satisfy the main requirement of providing a functional user interface which enables the user to control the label cutter. While, in its current state, the program is little more than a proof-of-concept, it proves the capability of reliably interfacing with the label cutting system via RS232 and cutting labels of arbitrary size and shape based on user input.
FIGURE 16: CSM Label Designer "About" Tab. This tab simply displays the CSM logo, as well as version and copyright information.

4.5.2 Libraries and Classes

The development of the GUI and its underlying connectivity to the label cutter were greatly simplified through the use of several mature libraries. Along with the creation of a few project-specific files designed to control data, the project's GUI and control aspects are isolated from the main code. Below, each of these libraries and project files are described in detail.
4.5.2.1 Library: SFML

The Simple and Fast Multimedia Library, SFML, was chosen as the graphical framework for the label designer program. It is a multi-platform (Windows, Linux, and Mac OS), zlib-licensed library aimed at simplifying access to various computer components and displaying graphics [25].

4.5.2.2 Library: SFGUI

The Simple and Fast Graphical User Interface library, SFGUI, was chosen to build the user interface for the label designer on top of SFML. It is a zlib-licensed C++ library that handles window creation and provides a number of predefined widgets such as buttons, toggles, text fields, and scroll bars [26]. Window layout is also simplified by automatic element sizing and positioning, and the appearance of most elements can be tweaked through a few lines of code or theme files.

4.5.2.3 Library: TinyThread++

In order to handle multiple threads in the program, TinyThread++ (TinyThreadpp) was leveraged as a simple, cross-platform alternative to the common Boost library. It is a zlib-licensed C++ library that implements threads, mutexes, and condition variables without requiring external dependencies [27].

4.5.2.4 Library: Serialib

For implementing the RS232 communications protocol, Serialib was determined to be the simplest solution that supported multiple platforms. It is a license-free C++ library which facilitates RS232 serial port access on Windows and Linux at a number of standard baud rates [28].
4.5.2.5 Class: shared_data

Because the label designer uses multiple threads and shared variables can change value in multiple threads, it is important to make sure that simultaneous access of variables is not possible. The shared_data class contains a template for shared use of any input data type. It leverages mutexes from the TinyThread++ library to implement locking and waiting on variable access for multiple threads.

4.5.2.6 Structure: Label

The Label structure is intended to store all relevant information about a label for sizing and printing. It can currently store the size of a label (in inches), as well as prescription, patient, and doctor information. Additional information is simple to add and access within the structure as necessary.

This structure is also type-defined as the “label” type where its header is included, allowing it to be declared as a variable in the code for readability and ease of use.

4.5.2.7 Class: dynamic_label

The dynamic_label class uses SFML, SFGUI, and the label structure to completely define a label as an object within the software. It contains all of the user-input information about a label within the label structure and creates SFML and SFGUI objects for their proper graphical display.

SFML properties describe the polygon rendered to represent the label on the preview sheet. SFGUI properties include the code for click events and rendering position. Other label properties are stored as standard C++ strings, as dictated by the label structure.
4.6 Documentation

For a project of this magnitude, a standardized approach to documenting source files is integral to long term maintenance and success. As such, Doxygen, a GPL-licensed tool [29], was chosen for generating all project documentation for the label cutter firmware as well as CSM Label Designer. This ensured all in-source comments followed a common structure and allowed for the generation of easily-navigable HTML documentation.

In order to access the documentation, it must first be generated, if it has not been done previously. This requires that the current system have Doxygen as well as a Make utility installed. Once Doxygen has been installed, simply open a Terminal or Command Prompt window, navigate to the project root directory, and type "make doc". This generates the full software documentation at (project root)/doc/html, but its main page (index.html) is embedded in the HTML file located at (project root)/doc/Documentation.html for convenience. Opening Documentation.html will load the documentation’s front page, which briefly describes the project and its various components.

In order to access the documentation for specific aspects of the project from the main documentation page, it is easiest to click on the “Classes” button in the top bar. This button is shown boxed in red to the right in FIGURE 17.

In FIGURE 17: Documentation Classes Button. Boxed in red, takes the user to the list of documented items in the project.
Clicking on the Classes button takes the user to the class list, shown below in FIGURE 18. The documentation for each class varies in depth, but methods that are more widely used or prone to confusing the programmer are described in greater detail.

![Class List](image)

**FIGURE 18: Documentation Class List.** The links to each class provide the user with descriptive comments about the parameters and usage of each method.

Additionally, if the programmer is unsure about a feature of the software, the documentation has a search box, located in the upper-right corner of each page, that allows for searching the documentation for specific variables, classes, definitions, and various other parameters.

A copy of the documentation main page and each class’ documentation page for the label cutter firmware and CSM Label Designer are included in Appendix A and Appendix C, respectively.
5. TESTING AND DESIGN VERIFICATION

Throughout development, several qualitative and quantitative tests were performed to ensure the project remained on track. A final evaluation of all design requirements was performed following the completion of this stage of the project. The results of these tests are detailed below and their ramifications are explored in the Discussion section.

5.1 Movement Calibration

In order to measure distances properly, it was necessary to experimentally determine the conversion between quadrature encoder counts and inches. This was done by replacing the cutting head with a fine-tipped pen and manually moving the x- and y-axis motors for 2-inches across an engineering scale. Each line was measured retroactively to ensure that a 2-inch distance had been covered. As expected, this resulted in an identical measurement for the conversion factors for each motor: 2600 encoder-counts/inch.

5.2 Linear Cut Speed

Linear speed tests, which equate to feed motor speed tests as well, were carried out with a series of 26 single axis cuts of 4-inches, forward and backward. This test is appropriate as a majority of the cuts on a label are straight and perpendicular. Because radiused corners can be achieved in two ways (allowing the cutting head to turn on its own, or programming in a specific circular cut), and because they are small, short cuts, they were not included in this measurement. Timings were reported based on the FreeRTOS system clock, taking measurements just before the start of the cut and just after the end of the cut.
5.3 Linear Cut Precision

Linear precision tests were carried out in a similar fashion to cutting tests for single axis cuts (26 cuts, 4-inch intervals, forward and backward), but instead of measuring cutting time, the total difference in encoder counts before and after each cut was measured. Each value was converted to inches using the conversion factor discussed above. Data for single axis cuts is shown in TABLE IV and TABLE V and data for cuts in both axes is shown in TABLE VI.

5.4 Circular Cut Speed and Precision

For circular cuts, 26 cuts of a 2-inch radius circle were performed in single degree increments. Algorithmically, this was equivalent to making 360 individual cuts in each axis sequentially. Speed and precision tests were carried out at the same time using this method. After each cutting operation completed, the difference between encoder counts and system clock time before and after the cuts was calculated. The encoder counts were converted to inches as discussed above. Data for circular cuts is shown in TABLE VII.

5.5 Results Analysis

The resulting data from testing is displayed below in TABLE II. In general, both the cutting speed (142.62 in/min, s=0.19334 in/min, n=26) and label edge length error (max: 0.00402 in, s=0.00686 in, n=26) are well-within the requirements outlined in TABLE I. Radial cut error (0.04384 in, s=0.01471 in, n=26), however, exceeded the project’s requirements at roughly 110% the target value with standard deviation roughly 37% of the target value.
**TABLE II: Cutting Test Results.** All tests were performed 26 times. Timing was measured by the Arduino Due clock. Distances were measured by differences in encoder counts, then converted to inches with the conversion factor of 2600 counts/inch.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Avg. Measurement</th>
<th>Std. Deviation (n=26)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Speed (average)</td>
<td>142.62 in/min</td>
<td>0.19334 in/min</td>
</tr>
<tr>
<td>Cutting Error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label Edge Length, X</td>
<td>0.00083 in</td>
<td>0.00554 in</td>
</tr>
<tr>
<td>Label Edge Length, Y</td>
<td>0.00402 in</td>
<td>0.00686 in</td>
</tr>
<tr>
<td>Label Radius</td>
<td>0.04384 in</td>
<td>0.01471 in</td>
</tr>
</tbody>
</table>

### 5.6 Label Edge Perpendicularity

Throughout testing, labels edges were checked for perpendicularity. Because both motors were on tracks, only circular cuts produced edges with measurably different perpendicularity. While label corners occasionally did not meet perfectly, the angle between corners for linear cuts was consistently between 89.8-degrees and 90.1-degrees. The difference in inches was negligible for the lengths measured. This process involved extending the edge for cut labels with a fine-point ball pen and measuring the corner angles with a protractor.

### 5.7 Label Designer Verification

A series of qualitative tests were performed to determine the efficacy of the CSM Label Designer software. These tests were performed first by the project’s team members, then six volunteer subjects (n=6) and involved judging the success of the design on a scale of 1 to 5 (low to high) based on three criteria: functionality, usability,
and presentation. Functionality described how well the program performed the actions that the user requested. Usability described how intuitive or easy actions were to perform within the program. Presentation described how pleasing the user interface was to the user. Results from these surveys are shown in TABLE III.

**TABLE III: CSM Label Designer Survey Results.** Six participants were allowed to test the program to cut a sheet of labels and then assess the label designer software based on its functionality, usability, and presentation. 1 is the lowest possible score, 5 the highest.

<table>
<thead>
<tr>
<th>Test</th>
<th>Avg. Response</th>
<th>Std. Deviation (n=6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>4.83</td>
<td>0.408</td>
</tr>
<tr>
<td>Usability</td>
<td>3.83</td>
<td>0.753</td>
</tr>
<tr>
<td>Presentation</td>
<td>2.17</td>
<td>0.753</td>
</tr>
</tbody>
</table>

This data suggested that the main goal of the software was accomplished, as users felt that the program was able to control the label cutting system well (4.83, s=0.4.08). Usability scored lower (3.83, s=0.753), however, and follow-up questions revealed that the choice of a tabbed interface confused several users, as the “Send to Cutter” button was not included on the tab where label data was entered. Two users suggested including a “Continue to Preview” button on the main page to navigate to the Sheet View tab. Finally, Presentation was scored relatively low universally (2.17, s=0.753). User comments suggested that while functions worked, they were not drawn in by any aesthetic and felt that the experience was lacking. Every surveyed user said that the interface was plain or bland and three users questioned why the Sheet View tab was
not incorporated into the Designer tab as well as why the Designer, Options, and About tabs were so spread out (see FIGURE 13, FIGURE 14, FIGURE 15).
6. DISCUSSION

The results obtained from the tests described above were generally positive given the project requirements, with the notable exception of circular cutting precision. An evaluation of the ramifications and validity of this data is conducted below.

6.1 Cut Speed and Accuracy Results

Based on the requirements in TABLE I and the results in TABLE II, cutting speed greatly exceeds the design requirement for cutting, averaging 8.4 times the desired minimum speed. This speed also applies to the feed motor speed, which is 63.4 times the desired minimum speed for label feeding.

Linear cuts should nearly always fall within specification: based on test data, 99% of cuts should fall within 0.04-in of their expected end location (0.01745-in for the x-axis and 0.02460-in for the y-axis). This accuracy was achieved through a combination of ramping speeds downward towards the end of cutting operations and using the changes in positioning error as proportional feedback to correct line-segment computations.

As noted in the Testing and Design Verification section, however, circular cuts are, on average, out of the desired range of precision. While reducing cutting speed and using positioning error as feedback in calculations did positively correlate to a reduction in cutting error for circular cuts, the accumulated error over a full circle still exceeded the desired precision by 10%. The experimental data suggests that 99% of cuts should fall within 0.08797-in of the desired end position, which is problematic. Given the strict requirements for FDA labelling of prescription drugs as well as the potential penalties associated with not meeting those requirements, a much smaller margin of error is absolutely necessary.

The validity of the test data was accepted after an examination of the cutting processing method and its timings. It can be assumed that the error distribution in the
system’s cutting time is normal and centered roughly 3 standard deviations from the expected cutting end time for several reasons. The chief factor is the system’s reactionary nature - timing errors that would result in cuts ending prematurely are highly unlikely because FreeRTOS is signaled to switch between tasks based on the value of semaphores that signal cut completion [20]. Because these semaphores end tasks based on the expected encoder position at the end of each cut, tasks will never end prematurely given proper coding. This suggests that any random error introduced by FreeRTOS will be delay, as the microcontroller cannot instantaneously respond to the interrupts issued by encoder or clock pulses and it will not anticipate future pulses, thus this interrupt handling is also reactionary and inconsistent.

Error in cutting positioning can be explained partly through the logic above. Cuts are stopped in the encoder interrupts once the encoder counter reads a desired value. This suggests, once again, that cuts will never end prematurely and that the time it takes for an interrupt to fire, as well as the time it takes for the interrupt code to run will cause the cutting error to run consistently high. Another source of error lies in the time it takes the rotor to stop spinning due to inertia after the brake() method is applied. This also serves to inflate the cutting error, as inertia will never force the motor in the opposite direction of its previous travel.

The significant increase in error for circular cuts can be explained through the accumulation of error through the many individual cuts used to form a circle. Cutting error manifests as overshoot so the error should never “average out” and, indeed, only overshoot was observed. As such, it stands to reason that the observed error in cutting measurements was introduced by the system’s operation rather than testing procedures and that proportional control and speed ramping were insufficient to reduce the repeated error introduced in the circle-cutting process.
Examining the importance of circular cuts is also of merit. While prescription drug bottle labels do have rounded corners, they are never entirely circular. As specified in the requirements developed by Cheadle et al., only label corners are expected to be rounded. And indeed, quarter-circular cuts tend to fall within the desired error tolerance (around 0.022-in at maximum).

Reducing error in the system, however, is still beneficial. Even small increases in accuracy are likely to improve the user experience in pulling labels off of a sheet, as the label will have edges which align more closely. And, because deviation from the expected corner radius has the potential to introduce perpendicularity discrepancies through error accumulation in each axis, increased accuracy will also help to keep edge perpendicularity precision within specification.

Overall, combining the accuracy of linear cuts and the cutting head's tendency to rotate as the cutting direction changes produces labels of acceptable quality. Corners on labels produced in this fashion are somewhat sharp, so more accurate circular cuts would create more aesthetically pleasing labels.

6.2 CSM Label Designer

The label designer software achieved the project goal of simplifying the user experience of cutting labels, though testers noted that there were some questionable design choices. First, every individual who participated in the survey suggested that the program's visuals were subpar. Second, in the same vein, testers noted that some layout choices were unintuitive to them, such as the separation of the Sheet View tab from the Designer tab and the "emptiness" of the Sheet View, Options, and About tabs. These comments suggest that future iterations of the label design software will require a distinct focus on its appearance, as Mr. Finken initially anticipated. Because these aesthetic
issues also affect the usability of the program, addressing them is likely to improve both presentation and usability for users and thus increase both scores.
7. FUTURE RECOMMENDATIONS

Should work on the on-demand label cutting system continue, there are a number of avenues for improvement within the electromechanical system, the label cutter firmware, and the label designer software. These topics are discussed in detail in their respective sections below.

7.1 Electromechanical Design

The label cutter’s mechanical design can likely be improved with two significant changes: switching out the cutting head with a laser, and switching out the DC motors with stepper motors. Originally, Cheadle et al. suggested that lasers would be a poor choice for cutting labels, as they are were determined to be more expensive and would require room-wide ventilation, which reduced the portability of the system [2]. Upon further investigation, it was determined that the concern of reduced portability was unfounded, as many commercial label cutting systems, such as those produced by Spartanics and D.P.R. Srl, successfully incorporate lasers without altering room-wide ventilation [30], [31].

An added benefit in using a laser for cutting is finer control over cutting depth due to the relative linearity of a laser’s power output [32] compared to the solenoid, which makes precision depth easier to achieve for various label sheet thicknesses in software. A laser cutting head would also eliminate the occasional issue of labels catching on the cutting head and bunching during cuts.

Replacing the DC motors with stepper motors is recommended to increase the accuracy of cuts. The original decision to use DC motors for the label cutting system was made in favor of their speed and for the perceived need to provide tension on the label sheet during cutting. The cutting speed also vastly exceeded the requirements for the system. Stepper motors, however, provide a much higher holding torque than DC motors
[4] and thus are more suitable to maintaining tension on the label sheet. While their speed is lower, 17-in/min is not a hard requirement to meet with the large number of commercially available stepper motors and the label cutter’s pulley system.

The biggest benefit of using stepper motors would be the increase in positioning precision. Because stepper motors move a definite amount for every pulse sent from their controllers, keeping track of their position and keeping motion in multiple axes in sync is vastly simplified compared to DC motors. This change in the approach to positioning would allow very granular control over all movement in the system and has the potential to reduce cutting error substantially, as overshoot would never occur as long as the control lines for the motors are kept noise-free.

7.2 Label Cutter Firmware

Improvements to the label cutter firmware should mainly focus on circular cuts. If the current DC motor system is kept, the introduction of integral control for the motors is likely the best step to take to reduce cutting error. Integral control corrects for the error accumulated over time in a system. Introducing integral control would, therefore, help the system adjust to its target position over time. This could produce some ringing (or travel above and below the target position), but it is likely to improve cutting accuracy substantially.

Should stepper motors be introduced, the MotorDriver will need to be rewritten to send pulses to the stepper controllers and the EncoderDriver could simply be repurposed to keep track of positions based on the number of pulses sent to the motors. After making these changes, CuttingDriver would remain largely unchanged, though the motor speed ramping could be removed.
If the cutting head is replaced with a laser diode, a new driver should be written to control it. Currently, the solenoid operates using the MotorDriver, as operation of the motors and solenoid is largely consistent.

7.3 CSM Label Designer

The CSM Label Designer software has many potential improvements, as it was a proof-of-concept. First, the GUI should probably be rewritten to use a more standard framework or library. If C++ is the desired language for consistency between the label cutter firmware and label designer code, switching to Qt would be an excellent choice (though it comes with licensing fees for commercial projects [33]) as it is well-supported and maintained, has a graphical editor to ease the design and layout process, and supports all major platforms, including mobile. Should the program be rebuilt from the ground up, coding in Java will allow multi-platform compilation, native multi-threading, and GUI development through the official Swing or Netbeans widget toolkits.

Either choice will likely benefit the project more than continuing with SFML, which was chosen based on the assumption that a graphics-focused library would simplify rendering the label previews for the user interface. In the end, a polished user interface is much more important than simplifying the code for rendering graphics. Removing the dependency on SFML will also remove a restriction on usable platforms: SFML introduces a dependency on OpenGL 1.2 or higher, which created an issue during testing on a system that was not capable of supporting OpenGL 1.2.

Another recommendation for future work on the GUI is to integrate it with a relational database of prescription drug information. This would involve research on what types of databases pharmaceutical companies already have in place for storing drug information which may be available to pull data from. If there is not a common, viable system available to link with, there are two distinct options for storing prescription
information: an online database created and maintained for CSM customers or a local database filled by the company’s staff.

The online database could be built on standard relational database software, such as MySQL, Oracle, or Mongo DB. A local database could be achieved through the use of an embedded database library, such as Berkeley DB or H2. No matter the solution, it is important to simplify storage of basic prescription drug information for the end-user, as it is needed for every label for a particular drug. Storing details for created labels, including size and patient-specific directions, is a potential benefit to including a database as well.

Presets for common label sizes and text would also be a great improvement for the software. These could be researched and added into the software, possibly as a drop-down menu, to further simplify the label creation process for users. Providing these presets could also simplify label rendering somewhat, as specific rendering cases could be written rather than a generic method.

Last, communication protocols for common industrial printers should be researched and the inclusion of communication between both the printer and PC as well as the printer and the label cutter should be discussed. This topic is still somewhat nebulous, as little research has gone into it so far. A general plan of action, however, could use the following steps: 1) research suitable printers, 2) determine method for sending label printing data to printer, 3) implement communications standard to send printer in label designer software.
8. CONCLUSION

The purpose of this research was to improve upon the on-demand label cutting system previously developed by Cheadle et al. An emphasis was placed on further development of cutting operations through increasing precision and adding circular cuts, as well as the creation of dedicated PC control software to simplify the user experience. The introduction of proportional control and motor speed ramping significantly increased cutting precision for linear cuts, and circular cuts were introduced, although testing revealed that they do not fall within the necessary precision metric for label perpendicularity. Control software for the label cutter was developed for both Windows and Linux platforms and it is capable of fully manipulating the label cutting system, though the software is in its infancy and will need significant work before it is viable as an end-user product. Overall, the research conducted had a positive impact on the label cutting system and made clear the avenues which should be pursued for further refinement and future work.
REFERENCES


APPENDICES

APPENDIX A: Label Cutter Firmware Documentation

See supplementary files: Zimmerman_appendix_a
APPENDIX B: Label Cutter Firmware Source Code

See supplementary files: Zimmerman_appendix_b
APPENDIX C: Label Designer Software Documentation

See supplementary files: Zimmerman_appendix_c
APPENDIX D: Label Designer Software Source Code

See supplementary files: Zimmerman_appendix_d
## APPENDIX E: Experimental Data

### TABLE IV: Raw Data for X-axis Cuts

<table>
<thead>
<tr>
<th></th>
<th>X, Expected (in)</th>
<th>X, Actual (in)</th>
<th>X, Difference (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2.006923</td>
<td>0.006923</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1.995</td>
<td>-0.005</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2.007692</td>
<td>0.007692</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1.995769</td>
<td>-0.004231</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2.007692</td>
<td>0.007692</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1.995</td>
<td>-0.005</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>2.008076</td>
<td>0.008076</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1.99423</td>
<td>-0.00577</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2.008076</td>
<td>0.008076</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>1.995384</td>
<td>-0.004616</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>2.000769</td>
<td>0.000769</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1.996538</td>
<td>-0.003462</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>2.008461</td>
<td>0.008461</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>1.995759</td>
<td>-0.004241</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2.007692</td>
<td>0.007692</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>1.996538</td>
<td>-0.003462</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>2.001923</td>
<td>0.001923</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>1.994615</td>
<td>-0.005385</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>2.006538</td>
<td>0.006538</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>1.996923</td>
<td>-0.003077</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>2.002692</td>
<td>0.002692</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>1.996923</td>
<td>-0.003077</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
<td>2.007307</td>
<td>0.007307</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>1.995769</td>
<td>-0.004231</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>2.002692</td>
<td>0.002692</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>1.996538</td>
<td>-0.003462</td>
</tr>
</tbody>
</table>
TABLE V: Raw Data for Y-axis Cuts

<table>
<thead>
<tr>
<th></th>
<th>Y, Expected (in)</th>
<th>Y, Actual (in)</th>
<th>Y, Difference (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2.005</td>
<td>0.005</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1.993846</td>
<td>-0.006154</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2.011538</td>
<td>0.011538</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1.992692</td>
<td>-0.007308</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2.004615</td>
<td>0.004615</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1.997307</td>
<td>-0.002693</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1.99423</td>
<td>-0.00577</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2.008461</td>
<td>0.008461</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2.010769</td>
<td>0.010769</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>1.996538</td>
<td>-0.003462</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>2.011923</td>
<td>0.011923</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1.999615</td>
<td>-0.000385</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>2.008076</td>
<td>0.008076</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2.012692</td>
<td>0.012692</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>2.001538</td>
<td>0.001538</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>2.010384</td>
<td>0.010384</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>2.000384</td>
<td>0.000384</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>2.011538</td>
<td>0.011538</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>1.99423</td>
<td>-0.00577</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>2.012307</td>
<td>0.012307</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
<td>2.011153</td>
<td>0.011153</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>2.001153</td>
<td>0.001153</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>2.012692</td>
<td>0.012692</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>2.001923</td>
<td>0.001923</td>
</tr>
</tbody>
</table>
TABLE VI: Raw Data for Biaxial Cuts

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1.992307</td>
<td>2</td>
<td>2.007307</td>
<td>-0.007693</td>
<td>0.007307</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2.006923</td>
<td>2</td>
<td>2.003461</td>
<td>0.003461</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2.027692</td>
<td>2</td>
<td>2.012307</td>
<td>0.0153</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2.017307</td>
<td>2</td>
<td>2.001153</td>
<td>0.016153</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1.998076</td>
<td>2</td>
<td>1.983846</td>
<td>-0.001924</td>
<td>-0.016154</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1.991153</td>
<td>2</td>
<td>1.990769</td>
<td>0.000407</td>
<td>-0.009231</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1.998846</td>
<td>2</td>
<td>2.005769</td>
<td>0.006923</td>
<td>0.005769</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2.018846</td>
<td>2</td>
<td>1.987307</td>
<td>0.018846</td>
<td>-0.012693</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2.006538</td>
<td>2</td>
<td>2.015</td>
<td>0.006538</td>
<td>0.015</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>1.98923</td>
<td>2</td>
<td>1.983076</td>
<td>-0.01077</td>
<td>-0.016924</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>2.010384</td>
<td>2</td>
<td>2.017307</td>
<td>0.010384</td>
<td>0.017307</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1.990384</td>
<td>2</td>
<td>1.975769</td>
<td>-0.004616</td>
<td>-0.024231</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>2.022692</td>
<td>2</td>
<td>2.021923</td>
<td>0.022692</td>
<td>0.021923</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>1.990384</td>
<td>2</td>
<td>2</td>
<td>-0.009616</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2.009615</td>
<td>2</td>
<td>2.015</td>
<td>0.009615</td>
<td>0.015</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>1.992692</td>
<td>2</td>
<td>1.976153</td>
<td>-0.007308</td>
<td>-0.023847</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>2.003076</td>
<td>2</td>
<td>2.013461</td>
<td>0.003076</td>
<td>0.013461</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>1.991153</td>
<td>2</td>
<td>1.998461</td>
<td>-0.008847</td>
<td>-0.001539</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>2.010769</td>
<td>2</td>
<td>2.016153</td>
<td>0.010769</td>
<td>0.016153</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>1.993461</td>
<td>2</td>
<td>1.977692</td>
<td>-0.006539</td>
<td>-0.022308</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>2.005</td>
<td>2</td>
<td>2.023076</td>
<td>0.023076</td>
<td>0.023076</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>1.993846</td>
<td>2</td>
<td>1.983076</td>
<td>-0.006154</td>
<td>-0.016924</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
<td>2.017692</td>
<td>2</td>
<td>2.02423</td>
<td>0.017692</td>
<td>0.02423</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>2.018461</td>
<td>2</td>
<td>2.008076</td>
<td>0.018461</td>
<td>0.008076</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>2.009615</td>
<td>2</td>
<td>2.02423</td>
<td>0.009615</td>
<td>0.02423</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>2.02</td>
<td>2</td>
<td>2.006538</td>
<td>0.02</td>
<td>0.006538</td>
</tr>
<tr>
<td>---</td>
<td>----------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>-------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.042307</td>
<td>0</td>
<td>0.022307</td>
<td>0.034704</td>
<td>0.014704</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.040769</td>
<td>0</td>
<td>0.003076</td>
<td>0.0335</td>
<td>-0.004193</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.023846</td>
<td>0</td>
<td>0.021538</td>
<td>0.023381</td>
<td>0.021073</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.041923</td>
<td>0</td>
<td>0.019615</td>
<td>0.036955</td>
<td>0.014647</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.047692</td>
<td>0</td>
<td>0.015</td>
<td>0.038402</td>
<td>0.00571</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0.041153</td>
<td>0</td>
<td>0.012692</td>
<td>0.03201</td>
<td>0.003549</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0.041153</td>
<td>0</td>
<td>0.016923</td>
<td>0.034724</td>
<td>0.010494</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0.040769</td>
<td>0</td>
<td>0.046538</td>
<td>0.036065</td>
<td>0.041834</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0.025769</td>
<td>0</td>
<td>0.023846</td>
<td>0.023609</td>
<td>0.021686</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0.033076</td>
<td>0</td>
<td>0.020769</td>
<td>0.030606</td>
<td>0.018299</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0.040769</td>
<td>0</td>
<td>0.038846</td>
<td>0.038388</td>
<td>0.036465</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0.050384</td>
<td>0</td>
<td>0.030769</td>
<td>0.045212</td>
<td>0.025597</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0.025384</td>
<td>0</td>
<td>0.020384</td>
<td>0.020877</td>
<td>0.015877</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0.052592</td>
<td>0</td>
<td>0.008461</td>
<td>0.048893</td>
<td>0.004762</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0.05</td>
<td>0</td>
<td>0.01923</td>
<td>0.047336</td>
<td>0.016566</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0.055</td>
<td>0</td>
<td>0.015</td>
<td>0.052841</td>
<td>0.012841</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0.06923</td>
<td>0</td>
<td>0.037307</td>
<td>0.067269</td>
<td>0.035346</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0.056923</td>
<td>0</td>
<td>0.00376</td>
<td>0.051795</td>
<td>-0.001368</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0.076538</td>
<td>0</td>
<td>0.010769</td>
<td>0.070666</td>
<td>0.004897</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0.061538</td>
<td>0</td>
<td>0</td>
<td>0.057359</td>
<td>-0.004179</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>0.065384</td>
<td>0</td>
<td>0.011538</td>
<td>0.065089</td>
<td>0.011243</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>0.064615</td>
<td>0</td>
<td>0.029615</td>
<td>0.057481</td>
<td>0.022481</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>0.035</td>
<td>0</td>
<td>0.010384</td>
<td>0.02862</td>
<td>0.004004</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0.066923</td>
<td>0</td>
<td>0.007692</td>
<td>0.062278</td>
<td>0.003047</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>0.069615</td>
<td>0</td>
<td>0.013461</td>
<td>0.064252</td>
<td>0.008098</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>0.045</td>
<td>0</td>
<td>0.029615</td>
<td>0.037581</td>
<td>0.022196</td>
</tr>
</tbody>
</table>