Final Paper: Team Cookie-Bot

ME 495: Mechanical Engineering Design & Capstone Project

By John D'Angelo, Jeremy Ridge, Seymareh Roomiany,

Laura van Kolck, & Nicholas Wright

Table of Contents

Abstract

The goal of this project was to create a prototype machine to print a user specified design on cookie dough and then subsequently bake the cookie, all as an automated process. In the process, optimal baking curves using pulse width modulation (PWM) were implemented with a variety of different cooking chamber configurations. The design to be printed onto the cookie is taken through a graphical user interface. Thermocouple feedback was not achieved due to a faulty thermocouple, but based on the perfection of the baked cookies, the PWM heating profile is assumed to have been achieved.

The machine itself was built successfully, although not perfectly. Two of the stepper motors did not work properly, but otherwise the machine operated as specified. An X-Y coordinate system was implemented using threaded rods and nuts. An M&M dispenser was manufactured to drop M&Ms onto the cookie dough using an optical sensor for feedback. A baking chamber was built using a Pyrex glass cover to hold a resistive heating element.

After manufacture and assembly, numerous baking tests were run on raw cookie dough to achieve perfect cooking and post baking appearance. Initially, the heating element was simply connected into 120V AC wall power for 5 minutes to bake the cookie. This caused undercooking on the bottom side and center of the cookies. After implementing infrared (IR) radiation shields and a 12V DC convection fan, along with PWM to control the baking curve, optimal baking of cookies was achieved. These optimally baked cookies appear golden brown on the outside and are thoroughly cooked on the top, bottom, and center, as to be slightly chewy, but not crunchy.

Introduction

The main goal of this project is to print a user-specified design on a cookie (using edible materials), and subsequently bake the cookie. Specifically, for this project, a fully edible, freshly-baked cookie is outputted by the machine. This project can be broken down into 3 different systems: a baking system with control feedback, a system used to move the cookie, and a dispensing mechanism. All actuators within all three segments of our machine are controlled through the use of an Arduino Mega microcontroller (Arduino, 2013).

This project is important as it has many applications outside of being our senior capstone. For example, it could be used in the future (provided more iterations of design) as a vending machine application, where a user inputs currency, and a freshly baked cookie is mixed, baked, printed and given to the user. This design could also be applied within the house to create and bake various oven-cooked foods, where for example, the user could initiate the baking process remotely while leaving work, and come home to a freshly baked confection.

Significant funds were required in order to purchase all the parts required to create the prototype. The budget and complete parts list of all the components used in this project is given in Appendix A: Budget and Parts List. In order to manage the team properly, the work required to complete this project was broken down into several separate tasks and assigned to members of the group. The complete breakdown of the tasks and assignments is given in Appendix B: Gantt Chart. Also, in order to perform this project, the prototype went through different iterations of design. Previous iterations and failed concepts are described in Appendix C: Previous Iterations of Design.

Control feedback is incorporated by baking the cookie with an optimal baking curve. The temperature within the baking module in the machine is fed back through a sensor and into the microcontroller in order to implement the control system. The feedback temperature is used to regulate the temperature within the baking module, and to bake the cookie according to the optimal baking curve (Piazza & Masi, 1997). The goal regarding the temperature system is to follow the optimal baking temperature curve as close as possible.

Background

In order to look for further inspiration when designing aspects of our project, a review of similar work done in this field was performed. A similar project under the name of Let's Pizza has been created (Torghele, 2009), which is a pizza vending machine that makes pizza from scratch, given user inputs. Additionally, research has also been performed on the optimal baking temperature curve for a cookie, which is referenced and include in this project (Piazza & Masi, 1997).

There are many opportunities for risks and liabilities to occur. Three main risks for this project are identified as the following:

i. Food contamination due to bacterial growth
ii. External environmental hazard: hot surfaces due to the baking process and motors
iii. Bacterial growth on equipment due to the combination of stray food particles and warm temperatures

In order to accommodate and minimize any potential liabilities from these risks, the following steps were taken for mitigation:

i. To avoid food contamination, the NSF/ANSI 51 food safety standard for food equipment materials is followed, which helped to determine the appropriate materials. Additionally, any perishable materials were kept chilled until use.
ii. To avoid burns, external guards were placed along the outside of any hot surfaces
iii. Dispensable protective materials were placed between any perishable food materials and the equipment to avoid food contamination in the equipment. Additionally, between sessions of usage, any materials in direct contact with food are either disposed of or cleaned.

In relation to ethical issues, according to the ASME Code of Engineering Ethics (Dieter, 1997), "Engineers shall hold paramount the safety, health and welfare of the public in the performance of their professional duties." In order to apply this to our project, the steps above were taken, as described in the risks and liabilities section.

There are many ways this project could have an impact on society. By performing this project as a senior capstone, it was possible to:

i. aid the advancement of 3D food printing and additive manufacturing
ii. assists in making 3D printing more common in everyday households
iii. raises awareness of Mechanical Engineering and Mechatronics in a familiar and friendly way

Because of these reasons, our project definitely has the capacity to positively impact society. In regards to this project's impact on the environment, there are very minimal effects that it will have. In order to conserve energy, our process will aim to reduce power consumption as

compared to conventional cooking. For conventional cookie baking the entire oven is preheated to one temperature at which the cookies are baked and then the thermal mass is allowed to dissipate after baking. For our project, the cooking compartment is sized for a single cookie so there is no excess of heated volume. Our project utilizes the initial heating of the cooking element to create a baking curve that enhances the cooking of the cookie, while at the same time not wasting any energy with preheating or post bake dissipation. The oven is brought up to temperature and allowed to cool all while the cookie is present in the compartment.

Mechanical System Design

The mechanical system designed for the Cookiebot had two primary objectives. First it needed to be able to print a design on a cookie with eatable material and subsequently it needed to bake the decorated cookie. These systems then needed to be integrated so that they could all work together as one cohesive system. Figure 1: System Design Evolution displays the evolution of the design from the initial design stages up to the working prototype. During the initial stage of the design the team decided to displace the cookie platform instead of the dispenser head. This decision was made because the cookie needed to be baked at the end and the decision was made to separate the printing area from the baking area. Hence, during the intermediate design the printing area was made wider and more spacious to contain all of the different systems at work. Besides a few minor changes in the component designs the final prototype ended up with most of the features of the intermediate design. In total the team machined 25 five components and put together 20 assemblies. The features and details of the sub systems of this prototype are discussed in the subsequent sections.

Figure 1: System Design Evolution

X-Y Planar Motion of the Cookie Platform

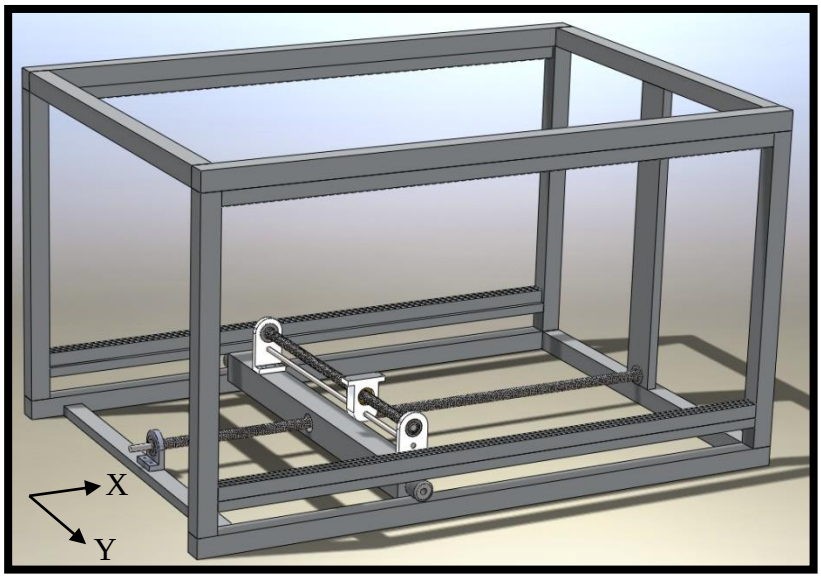


Figure 2: 2D Motion System of the Robot

Figure 2 displays the mechanism for the 2D motion of the Cookiebot. The X-Y planar motion was designed to perform two major tasks as follows:

1) Carry the cookie dough to its printing coordinates during the dispensing stage
2) Deliver the decorated cookie to the heater during the baking stage

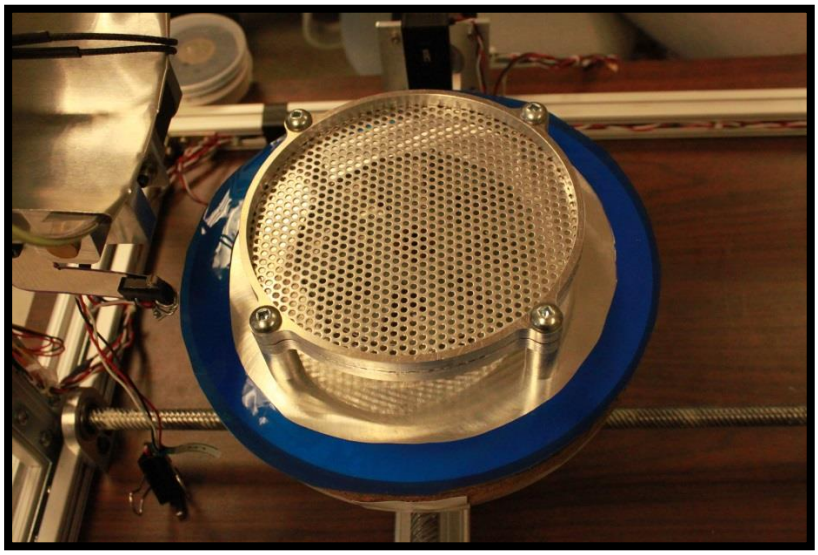


Figure 3: Baking/Printing Platform

Figure 3 displays the dual purpose platform that was designed to carry the cookie between the different stages. This platform sits directly on top of the y-axis. Precision threaded rods and traveling nuts, as seen in Figure 4: Fast Traveling Nut and , are used to convert the rotational

motion of the stepper motors to linear motion in each axis.

Figure 4: Fast Traveling Nut and Rod

The stepper motors used for linear motion are capable of micro-stepping; therefore high precision in linear motion can be achieved through this setup. As seen from Figure 2, rotation of the x-axis threaded rod shifts the entire y-axis along the x-axis path. The rotation of the y-axis threaded rod directly moves the baking/printing platform along the y-axis path. The combination of the X-Y axis motions creates the planar motion of the platform and makes it possible to print design on a cookie. The details of how this 2D platform is explained in the Printing System section.

Dispenser Mechanism

Initial dispenser design centered upon the goal of using miniature chocolate chips as the printing media. After collecting data on chocolate chip sizes, the printing media was changed to mini M&Ms. Later, after testing an initial prototype, the dispenser design was changed to better work with the M&M geometry.

As it stands, the dispenser design is shown in Figure 9 below. The M&Ms sit in a clear polycarbonate tube whose inner diameter is slightly larger than the major diameter of an M&M. The M&Ms stack inside the tube and are kept inside by a diaphragm at the end with a hole cut out of it. The dispenser mechanism itself relies on a rack and pinion to actuate. The pinion is connected to a stepper motor programmed to step through an angle that moves the rack through a distance equal to the minor diameter of a mini M&M. The motor and pinion are connected via set screw and an aluminum connecting sleeve. A gearbox assembly attaches to the motor to mesh the rack and pinion. The rack nests inside of the M&M tube and plays the role of actuator. The M&M tube connects to a metal rod that in turn attaches to an 8020 bar on the machine. The gearbox attaches to another piece of 8020 (not shown) that is vertical on the machine. Additional detail of the dispenser can be seen in the exploded view in Figure 6.

The result of this system is that once the cookie platform gets to its prescribed location, the Arduino sends the required number of steps the to the pinion motor to eject one M&M. The dispenser diaphragm allows one M&M to pass through and then contains the remaining column of M&Ms until the next actuation procedure. An optical sensor is attached to the dispenser below the diaphragm to detect whether or not an M&M has been properly ejected. If the optical sensor does not detect an M&M, the program increments the motor by single micro steps until an M&M is ejected and sensed in the optical sensor. The optical sensor is not shown in the dispenser model below, but it is shown in the picture of the realized dispenser.

There is a 3D printed part that is used to hold the optical sensor to the M&M tube as shown in Figure 5 below. A printed part is also used to hold the diaphragm to the tube of M&Ms. The completed dispenser is shown in Figure 5 on a following page. It looks slightly different than because the dispenser had to be moved up in order to leave clearance room for the cookie platform.

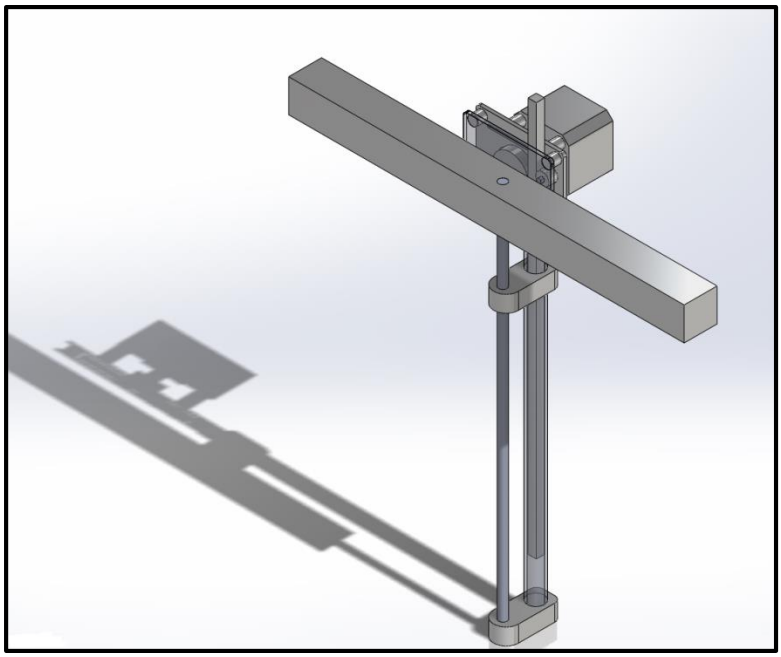


Figure 5: Current Dispenser Design

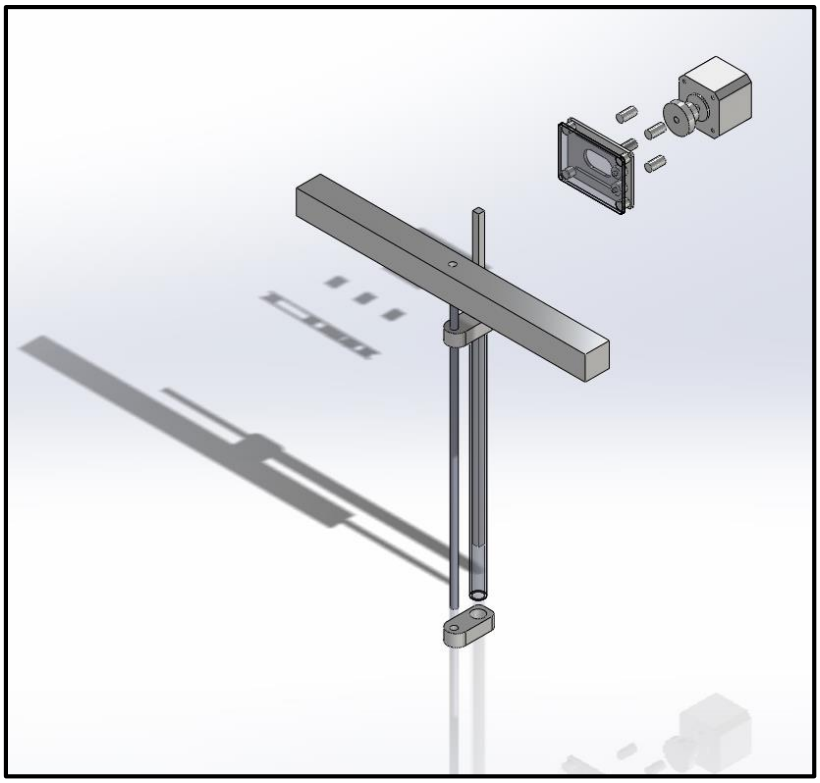


Figure 6: Exploded Dispenser View

Figure 7: Completed Dispenser Prototype

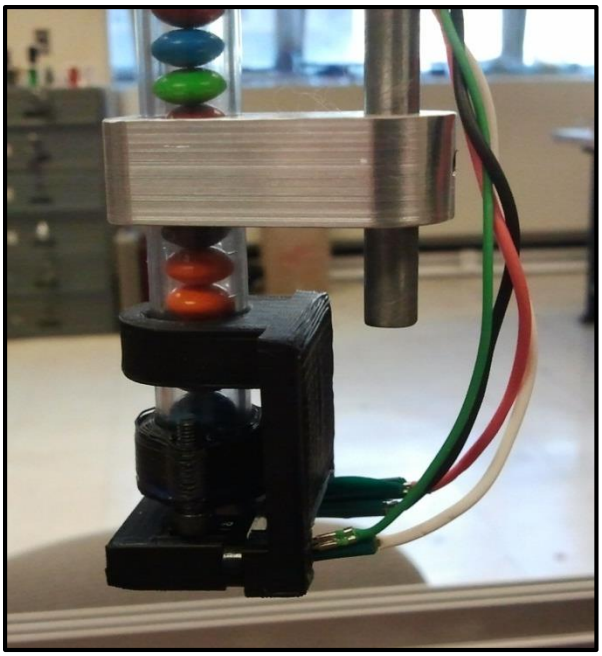


Figure 8: Dispenser Optical Sensor

Dispenser Requirements

Initial engineering decisions required the use of miniature chocolate chips to be printed on the cookie. Many ideas were brainstormed about different possibilities for achieving accurate placement of the candy such as belts and rollers, pneumatic actuation, and spring loaded mechanisms.

After much preliminary design work, it was decided to go down a path in which a solenoid actuated and pushed candy through a dispenser one by one, as can be seen in Figure 9 below. The chocolate chips were idealized as spherical balls and Figure 10 shows the desired operation of the dispenser. A chocolate chip rests on a diaphragm at the bottom of the dispenser and holds back other chips in a tube the same of the chip diameter from entering the chamber. A push type solenoid is actuated which ejects the chocolate chip and causes it to be placed on the cookie medium. As soon as the solenoid retracts, a new chip is allowed to enter the chamber and the process starts anew.

After rejecting chocolate chips as the candy of choice, a prototype of the original dispenser was created using mini M&Ms as their shape is closer to the spherical idealization than chocolate chips are. The M&Ms move to lowest energy state while stacking in the tube, which means they align with their minor axis vertical. This causes the prototype dispenser to let two M&Ms into the chamber at the same time and the prototype does not work as designed.

Candy Choice

The main requirement for the candy was a size small enough to make a picture with the highest resolution possible. The three top contenders that fit this criterion were miniature chocolate chips, miniature M&Ms, and Sixlets. Initially, miniature chocolate chips were chosen because of preference of taste. It was later found that their geometry did not meet the requirements for successful operation of the dispenser design. The determination of the candy type is shown in Appendix D : Candy Diameter Distributions.

Sixlets have a spherical shape and were the perfect choice if ideal geometry was desired. However, many anecdotal reviews report that consumers are not satisfied with the taste of Sixlets and this was the consideration that caused M&Ms to be the chosen printing media (Ragan, n.d.).

M&Ms are spheroidal in shape. That is to say that they have two principal axes and are created by rotating an ellipse about its minor axis. This is not as ideal as a sphere, but it is much more ideal than the random droplet shape of a chocolate chip.

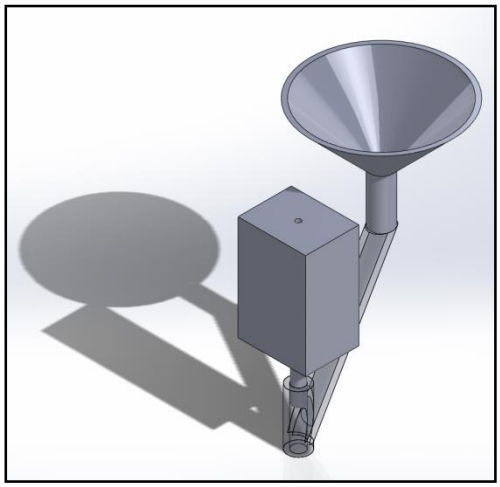


Figure 9: Original Dispenser Design

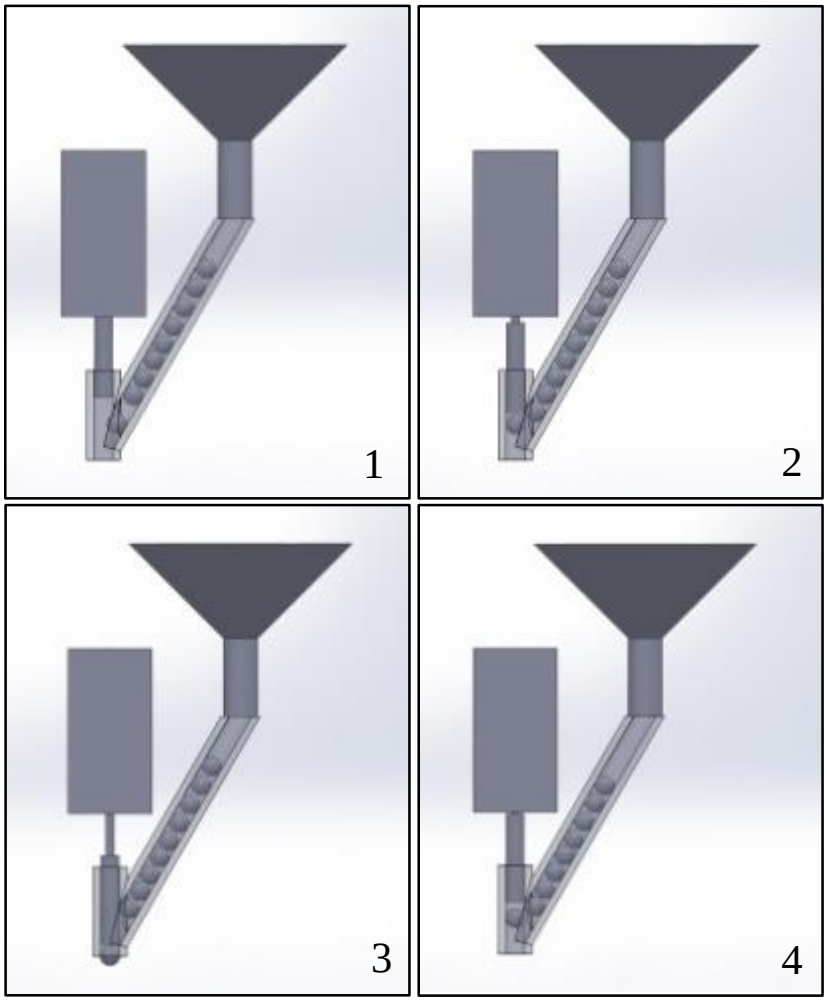


Figure 10: Idealized Initial Dispenser Operation

Heating Mechanism

Figure 11 displays the heating chamber and the platform design. The functions of these mechanisms are as follows:

- The base platform
  - Responsible for carrying the cookie dough through the printing stage and to the heating stage.
  - Holding the thermocouple.
  - Holding the convection fan.
  - Insulating the rest of the mechanism from the heat of the oven.
- The heating chamber
  - Enclosing the oven area

- Holding the infrared element

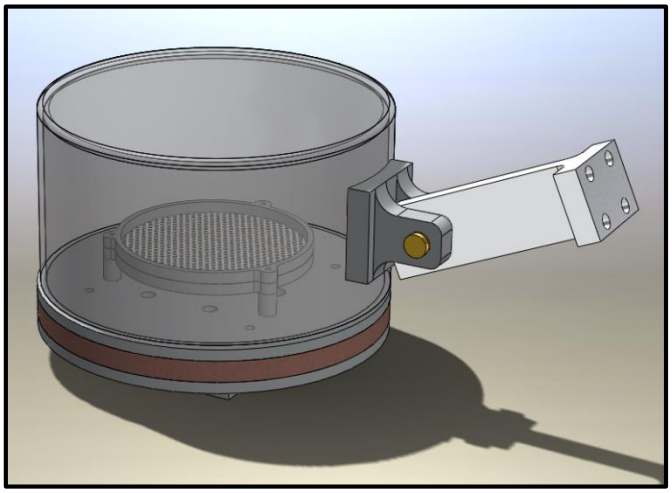


Figure 11: Oven Chamber and Platform Design

Some design considerations that the team had during the while designing were as follows:

- It was most important to use food safe material in building the components that go inside of the baking chamber and the components that came into contact with the cookie dough.
- The process needed to be visually appealing
- The base platform needed to be light weight and compact
- The convection fan needed to be dual purpose
    1. Act as a convection fan when the enclosure is closed
    2. Cool down the cookie when the enclosure is opened

Based on these design criteria, the prototype in Figure 12 was machined and assembled. The NSF/ANSI standard 51 for Food Equipment materials suggests the use of 6065 series of aluminum as structural material; hence, the structural components of the oven were machined out of this series. One concern with using aluminum in food equipment is the corrosion that may occur with baking acidic food (NSF/ANSI 51 Food Equipment Materials, 1997). To eliminate this issue the base of the platform where the cookie dough comes into contact with was made out of a perforated stainless steel T-304 which is used in most industrial food equipment and has excellent corrosion resistance characteristics.

To make the process more visually appealing it was decided to use a tempered glass as the oven enclosure, as tempered glass can withstand any thermal shocks from the fast heating and cooling of the oven. The most economical type of tempered glass that was available was a lab quality 190 mm x 10 mm crystalizing dish. The sizing of our platform was based around the diameter of this dish to make sure that when it rests on top of the platform it would seal the oven properly. A silicon oven liner was used to seal the gap between the glass enclosure and the platform, as seen in Figure 12.

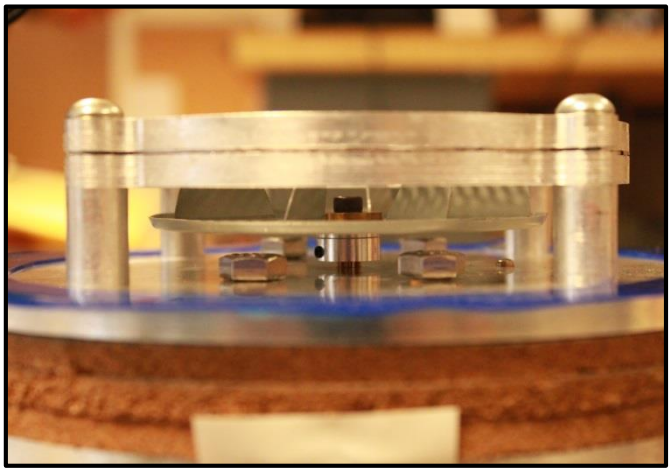


Figure 12: Close Up of Printing/Baking Platform

The primary form of heat from the infrared element generated was radiation. Due to the properties of the k-type thermocouple, it is not particularly sensitive to the effects of radiation, hence, it was decide to create forced convection inside of the oven. To create forced convection a small 12 VDC motor was integrated to the base of the platform, as seen in Figure 13. The preliminary thermocouple testing is described in Appendix E: Thermocouple Test.

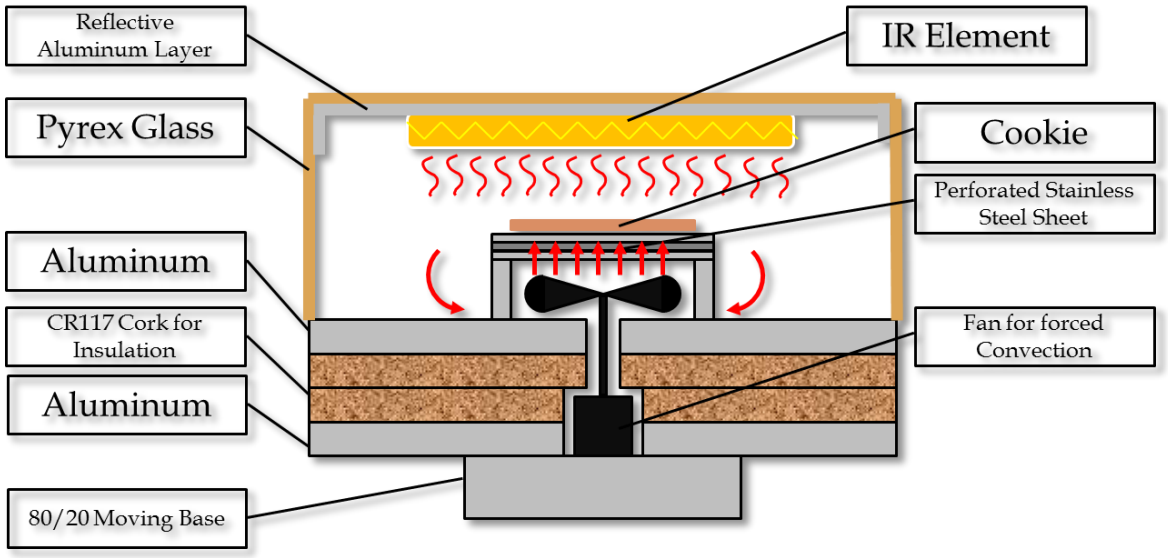


Figure 13: Oven Chamber Schematic

Figure 13 summarizes the whole heating mechanism. Notice the different layers of the platform. The CR117 cork insulation layers were added to prevent damage to the DC motor and the other

mechanism below the heating compartment. A reflective aluminum layer was added above the IR element to help focus the radiation towards the cookie at the center of the baking platform. This thin layer of reflective aluminum also acted as a mount for the element. The convective fan stirred the air inside of the oven and also assisted in baking the bottom of the cookie that never receives the radiation from the element.

The majority of machining and assembly time of the whole project was spent on the heating compartment, as it was the primary focus of the team. The controller and the functionality of the heating system are described further in the Heating System section below.

Control System Design

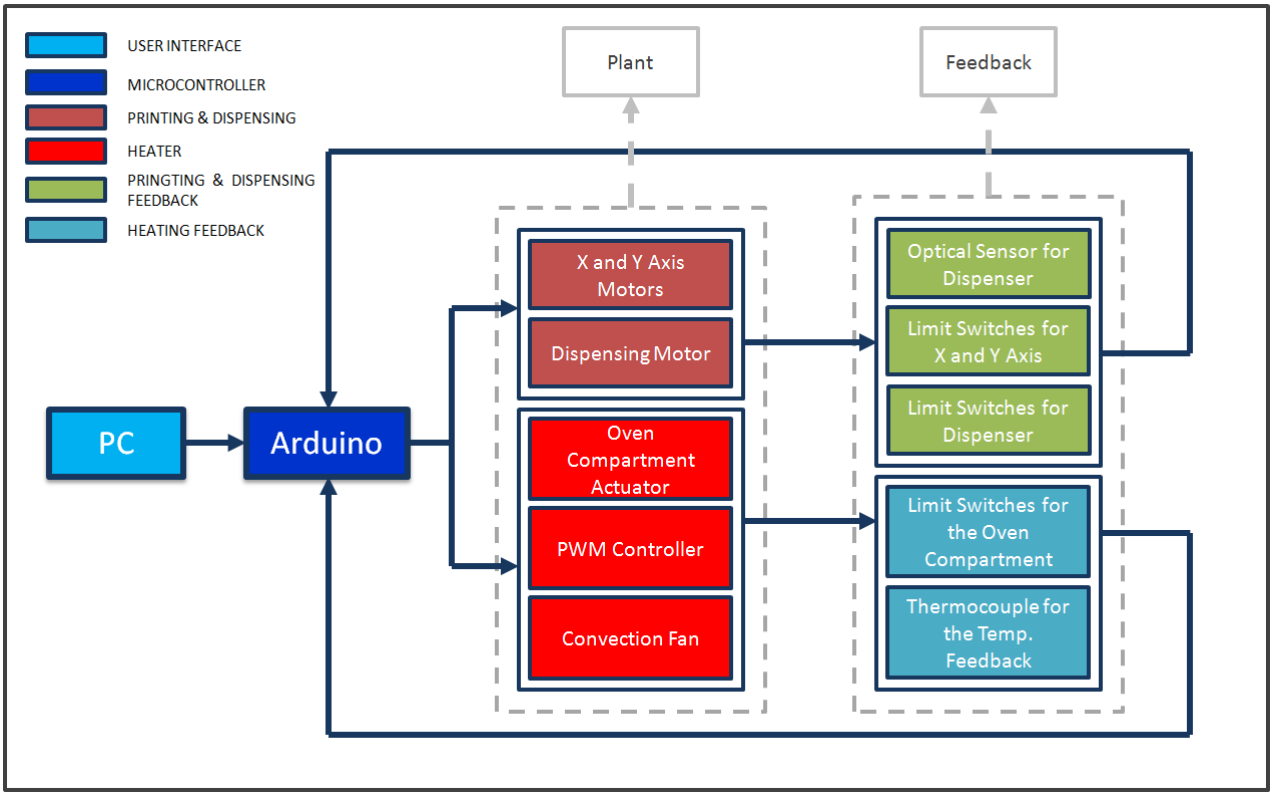


Figure 14: Block Diagram of the Closed-Loop Control System of the Cookiebot

Figure 14 above describes the control system for the Cookiebot. The input into the system is a customized design made by the user through a graphical user interface (GUI). This design is then sent to the microcontroller to be interpreted as coordinate system for the X-Y positioning system. The controller has three primary objectives, as described by the plant, 1) controlling the X-Y positioning system 2) controlling and synching the dispensing mechanism with the positioning system 3) controlling the baking process. Three forms of feedback are used to assure the proper functionality and safety of the user and the machine.

Limit switches were used for the X-Y axis to ensure that the system would not run out of its physical boundaries as well as providing feedback for the auto-calibration algorithm. Two limit switches were implemented in the baking process to prevent it from overturning the glass compartment and damaging the glass. One limit switch was dedicated to the dispensing mechanism to be triggered when the rack has reached its lower limit.

An optical sensor was placed at the tip of the dispenser head to make sure that the mechanism would actually dispense an M&M.

For the temperature control a thermocouple was embedded into the baking platform as a feedback.
All the feedback signals were sent to the microcontroller to create a closed loop system. At the end the Output of the system is a 5 inch baked cookie with customized designed printed on it with M&M.

There are two major sub systems that are integrated in the Cookiebot. The first system is the printing system that is made up of a dispensing and an X-Y positioning system. The second system is a heating system that bakes the cookie based on a preset optimal baking profile. The details of these systems are described individually as follows:

Printing System

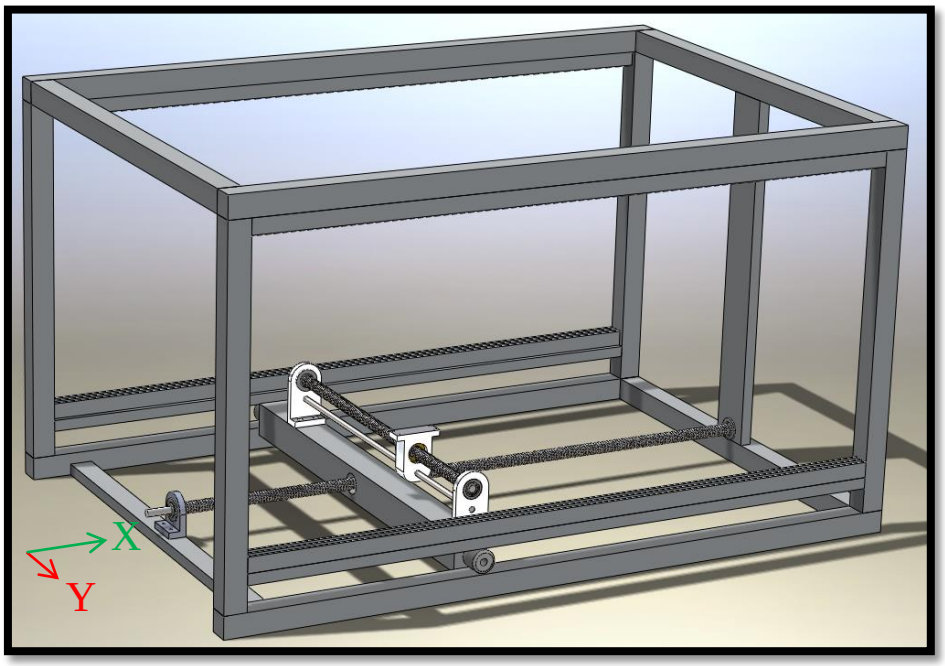


Figure 15: X-Y Positioning Mechanism

Figure 15 displays the mechanism for the X-Y positioning system. The printing/baking platform sits directly on top of the Y-axis.

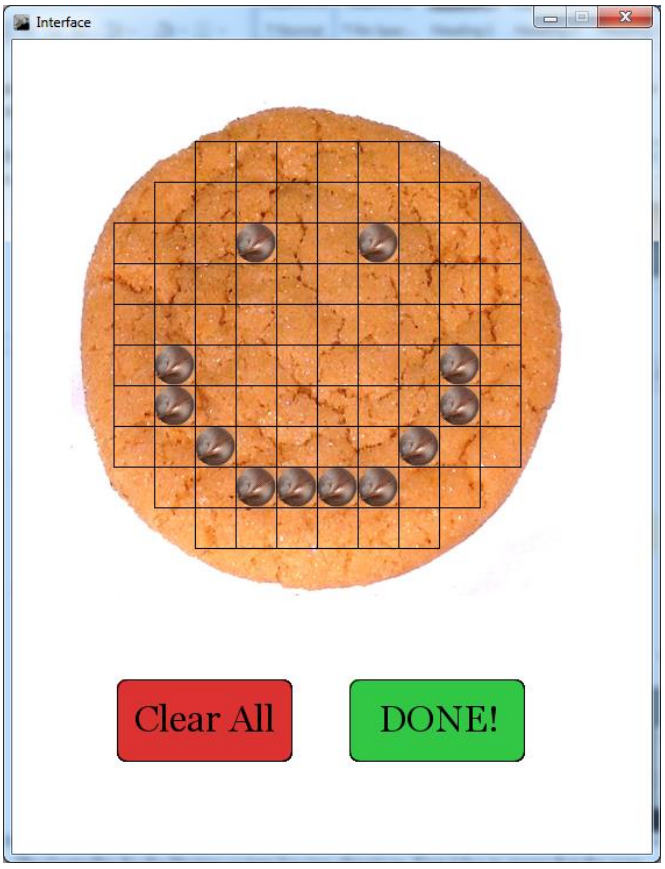


Figure 16: User Interface

Interface

The user interface program (depicted in Figure 16) was made through a programming language and development environment called Processing (Processing 2, n.d.). Processing was chosen for this task because it had a shallow learning curve and the development environment is the same as the Arduino's so the two are frequently paired together. The program is relatively simple. It allows for the user to add and delete chip onto a calculated grid to make their desired design. The grid locations activated by user are stored in a two dimensional array so that once the user hits the "DONE!" button of the interface, the information can be easily sent out to the Arduino. The full details of the code required to build this interface can be found in Appendix D : Candy Diameter Distributions

Communication between Processing and the Arduino was established using the serial port. The serial port is the USB cable that is used for uploading the code to the Arduino. Both Processing and the Arduino have built in serial port interface libraries which make reading and writing

between the two programs easy to implement. The drawback to this approach was that the computer was required to be connected to the system when it was in use.

When the user indicates (presses the "DONE!" button) that they are finished, the interface iterates through the 2D array and sends out the array index value for each coordinate system that the user placed a chip on. It send the X-coordinate out first followed by the corresponding Y-coordinate though the port. Figure 17 illustrates the index value scheme used in the interface.

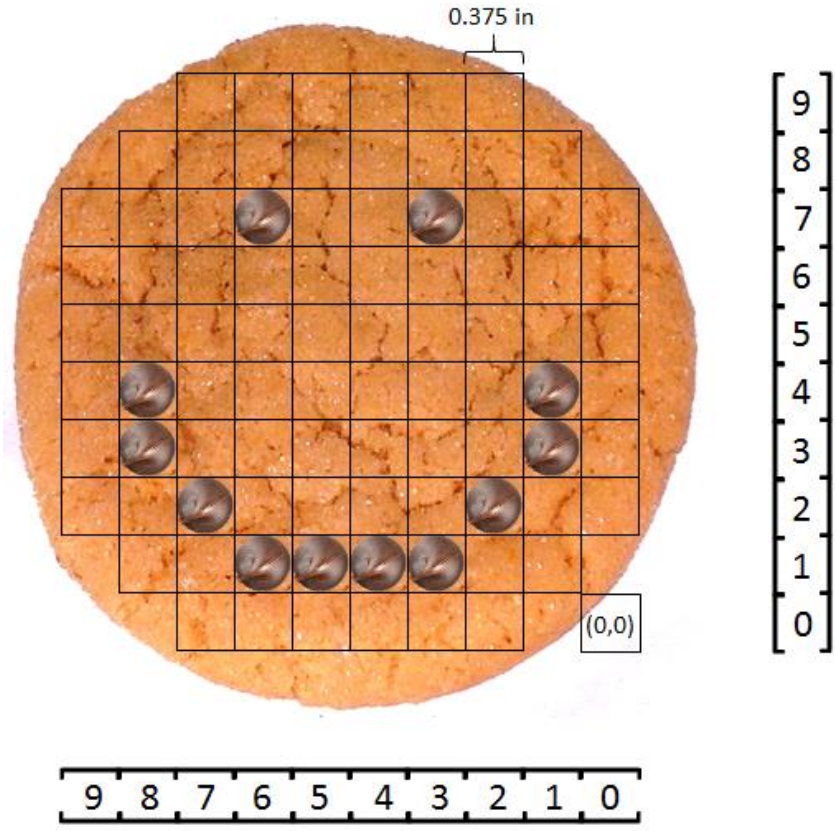


Figure 17: Cookie Grid.

Arduino

While the user is working in the interface the Arduino essentially does nothing. The way the Arduino works is similar to putting an infinite loop around everything inside the main function in C. Once a code has been uploaded, the Arduino will loop indefinitely until it is either unplugged (no power), a new program is uploaded or the reset button is pushed. For this reason, the Arduino does nothing but check if there is information waiting to be read through the serial port. Once there is, the first thing the program does is calibrate itself. This is a very important set of the process, without it there would be no way of knowing where the platform is when the program

starts. The complete details of the Arduino code can be found in Appendix G: Embedded Circuit Code.

Calibration and Coordinate System

Calibration consists of three parts: zeroing the heating lid, X-axis and Y-axis. Before ever moving the platform, the heating compartment lid must be lifted. This is to ensure that the platform does not collide with the glass lid. This is accomplished by rotating the lid up until the limit switch is activated. The action taken by the limit switches will be explained in detail later in this section. When the lid is lifted the X-axis will move and arbitrarily long distance (long enough to clear the entire X-axis distance) to the right until the right limit switch is depressed. Finally the Y-axis also moves an arbitrarily large amount to the front, again until the front limit switch is depressed. The X and Y limit switches are shown in Figure 18. The X and Y limit switches are primarily used for the calibration process however they are also very valuable for insuring that the platform does not collide into anything. Figure 19 shows both a depressed and not pressed limit switches used for both the X and Y axis.

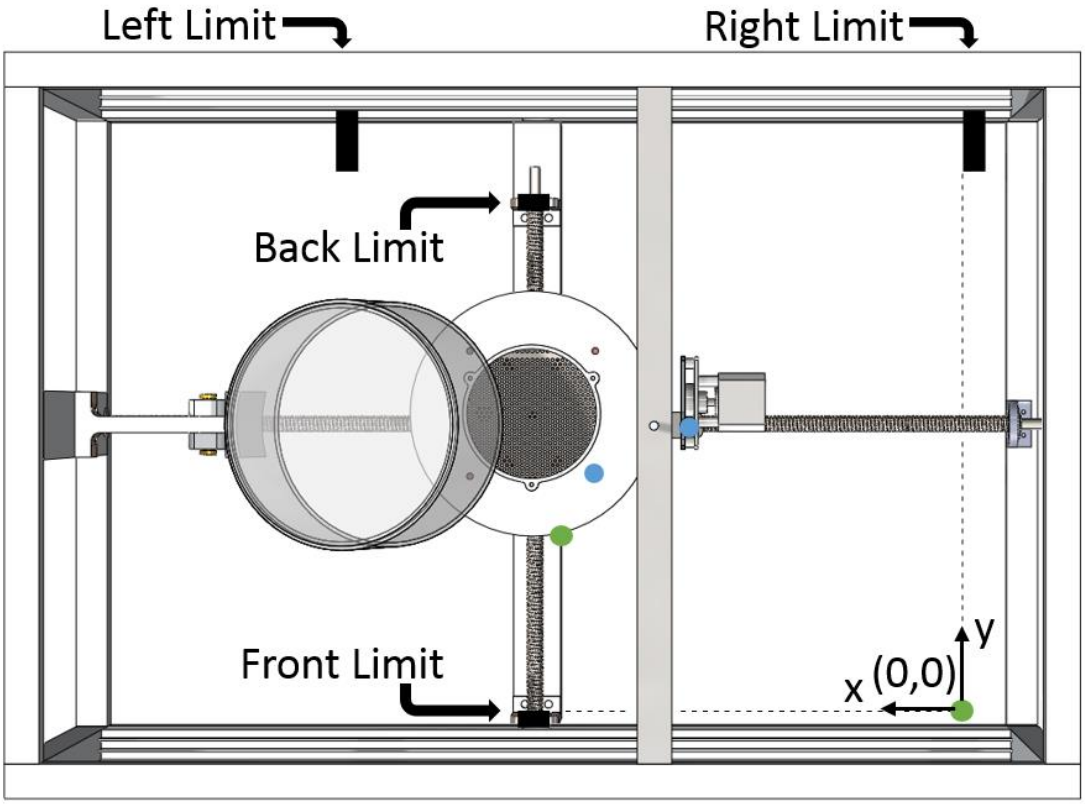


Figure 18: Top down View with Limit Switches

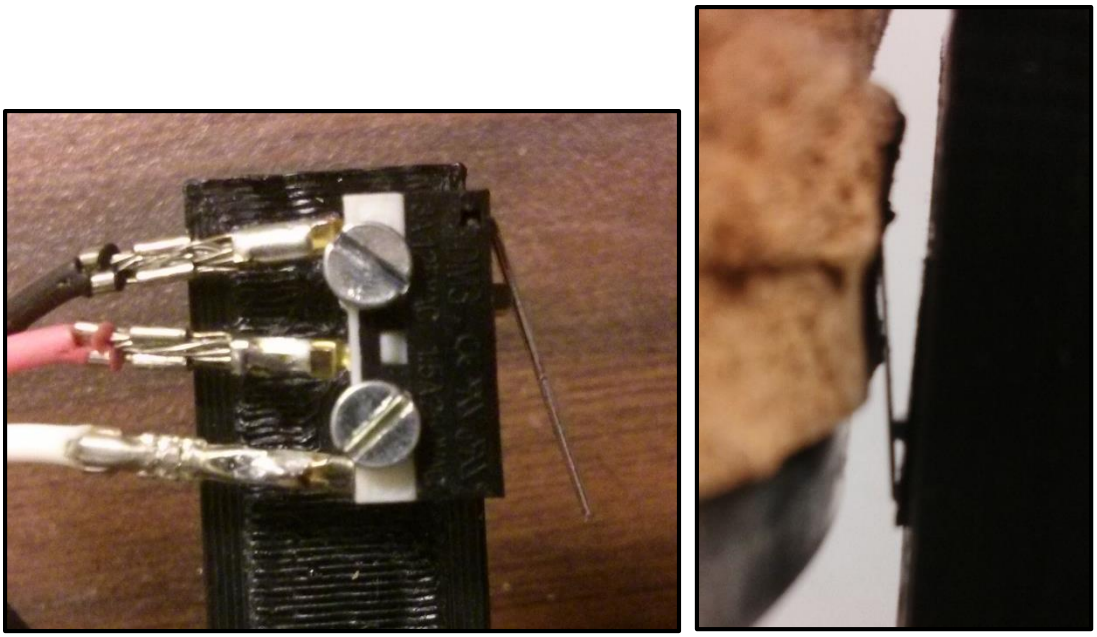


Figure 19: X-axis Limit Switch (Left), Y-axis Depressed Limit Switch (Right)

In Figure 18 there is a green dot in the lower right corner labeled "(0,0)". There is a corresponding green dot on the platform. After calibration takes place, those two greed dots will be perfectly aligned and will be marked as the absolute zero position on the platform.

Interrupts

As mentioned above, all limit switches, except for one mounted on the dispenser, cause an interrupt to occur within the program. This is so that the motor affecting that switch can be stopped as quickly as possible. The switches were wired were to produce no voltage to the Arduino pin when not activated. With that, the Arduino was programmed to trigger any of the limit switch interrupts upon a raising voltage. While this worked for the situation that the limit switch was initially not pressed to being pressed it would not work it the switch was initially pressed because it would read a constant high voltage and not a rising voltage (change from low to high), not triggering the interrupt. For this reason, the platform had to be checked to make sure it was not depressing any of the X-Y limit switches because this would cause the program to run directly into the switches. Unfortunately the Arduino Mega only supports rising voltages as an interrupt trigger but not high voltage. The Mega does support low voltage trigger however, at the point of this realization, the circuit had been designed for the opposite without time for rewiring.

When the motors are moving, (which is the case when a limit switch gets activated) they are periodically sending out pulses to the motors until they reach a target destination. During the calibration process, it was mentioned that the motors moves an arbitrarily long distance to ensure limit switch activation. To make sure that the motor does not continue moving once the switch is

pressed the interrupt sets both the current and target locations to be 0. This simultaneously zeros our system and changes the target location to the current location so that when it return to the moving function it will think it did its job (stop sending pulses) and exit.
Aside from Limit switches, one of the interrupts can be activated but the optical sensor installed at the end of the dispenser. Unlike the limit switches the optical sensor pin will read high until something obstructs the path between the IR LED and phototransistor. This type of obstruction causes a dip in the voltage. For this situation, the interrupt was programmed to occur when the photo interrupter interrupt pin falls from a high reading to low. To dispense an M&M, the program rotates the motor so that the rack moves in the amount equivalent to the thickness of an M&M. At times this may not be enough to fully dispense one M&M (e.g. Friction against the silicon diaphragm or varying M&M thicknesses). In the program, a boolean value is set to true (meaning the M&M has not been dispensed) before even attempting to dispense. When an M&M falls from the dispenser, the interrupt gets triggered and it changes this value to false. After the initial attempt, the program waits for half a second and checks if the boolean is still true. If it is, it will continue to move the rack in very small increments until that interrupt gets triggered.

Printing and Motor Control

The X, Y and heater motors are all controlled by a library downloaded from the Arduino community called AccelStepper (McCauley, 2010). This library was ideal for controlling the motors because it allowed for several motors to be controlled together. Prior to using this library the motors moved from point A to point B first all the way in X and then Y separately. While AccelStepper allowed for the motors to move together to that the path from A to B resulted in a line. This is illustrated in Figure 20.

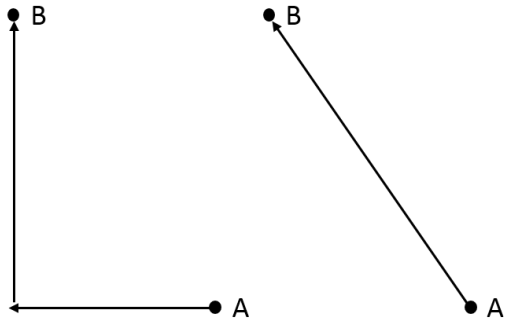


Figure 20: Initial Motor Control (Left), AccelStepper Motor Control (Right)

The function that controls the motors takes in four arguments, which are the X and Y coordinates of the starting point and X and Y coordinates of the end point. The function takes the difference between the X coordinates and the Y coordinates, sets the target position to the end points passed in, and determines the correct ratio of pulses needed to move each motor at the correct speed.

After the system has been calibrated, it proceeds to the next step in the process, printing. Looking back at Figure 18, the blue dot on the right is the location of the dispenser for printing. The blue dot on the left is the printing zero from Figure 17 located on the platform. In order to print, the grid on the platform must match up with the grid on the dispenser (line up the two blue dots).

Once the printing grid has been aligned, the print function goes into effect. This just cycles through the info waiting in the serial port from the interface and plugs that into the end point of the function described above for moving the motors together. After moving the motors, the dispenser releases an M&M and this continues for all M&M in the design.

The final step in the positioning is to move to the platform over to the heating position so that the lid can safely be rotated down and the cooking can commence. The left limit switch is positioned exactly so that when the platform hits it, the X position will be in place for cooking. The Y coordinate was just measured and passed through to the motor control function. Once in position the program rotates the heating lid down and the cooking control takes over.

Heating System

The heating system was implemented with the goals of being able to implement a temperature profile (over a set period of time) within the baking environment such that optimal baking cookie quality and baking time could be achieved. Through research into industrial cookie baking processes (Piazza & Masi, 1997), it was found that implementing a baking temperature profile (i.e. varying the temperature over time, as opposed to just a constant baking temperature) produces the best results for cookie baking quality and time.

The heating system implemented to facilitate the cookie baking process was comprised of 6 primary components:

- ATmega2560 microcontroller (Arduino Mega 2560 processor)
- Solid State Relay (SSR)
- Infrared (IR) Halogen Heating Element
- K-type Thermocouple (ungrounded)
- Analogue to Digital Converter IC (ADC) for Thermocouple Readings (MAX31855)
- Convection Fan Unit

The microcontroller was utilized to implement the baking temperature profile (through a discrete time control law), read data temperature data from the thermocouple and control the convection fan. Utilizing the pulse width modulation (PWM) function of the ATmega2560 microcontroller, a PWM signal was sent to the control end of the SSR which was utilized to regulate the AC wall

power (120 VAC) sent to the IR heating element (subsequently achieving temperature control). By sending PWM signals with a sequence of duty ratio values, different baking temperature profiles could be achieved. The frequency utilized for the PWM signal was 30 Hz, as this provided optimal control for regulating wall power supplied to the element (as the frequency of all power is approximately 60 Hz). The thermocouple was utilized with the intent to measure temperature data within the baking environment during the baking process (for the purpose of feedback). The thermocouple would produce a small analog signal which was sent to the ADC chip (MAX31855) that would filter, boost, and convert the signal to a digital signal to be sent to the microcontroller. Through experimentation performed (discussed in later sections) it was discovered that convection was also necessary for optimal baking quality and time, so the convection fan was implemented (powered by 12 VDC in the final design) to create forced convection within the baking enclosure. The microcontroller controlled the fan by simply sending an on or off signal to the fan circuit (no PWM). The implementation of the initial and final designs for the control scheme and control law is discussed subsequently.

Initially, there were two main functions that the controller for the heating system was to perform. First, it had to insure that the oven temperature stayed within safe limits (determined by the safety rating of the epoxy used for attaching the heating enclosure), and second it had to control the power input into the Infrared (IR) halogen element. During the initial design stage it was decided that a PID controller would be implemented for the heater control law, however during the intermediate design stage it was discovered that a simple open loop controller could be implemented to achieve the desired results with far less complexity. This was discovered as a result of complications with obtaining accurate temperature feedback from the thermocouple utilized in the backing enclosure (discussed further in the following). Figure 21 shows the block diagram of the initial closed loop PID controller that was intended to be used (Note, convection was not initially planned to be part of the implementation). This PID controller scheme would have utilized the convectional general PID control law shown in Equation 1.

$$G(s) = k_1 + \frac{k_2}{s} + k_3 s \qquad \text{(Eq.1)}$$

In Equation 1, k_1 is the proportional gain, k_2 is the integral gain, and k_3 is the derivative gain. The gain values for the control law would have been determined through modeling computer aided modeling of the system (utilizing MATLAB) and then subsequent refining through testing of the actual system; i.e. attempting to achieve input temperatures and temperature profiles through temperature feedback from the thermocouple inside the enclosure.

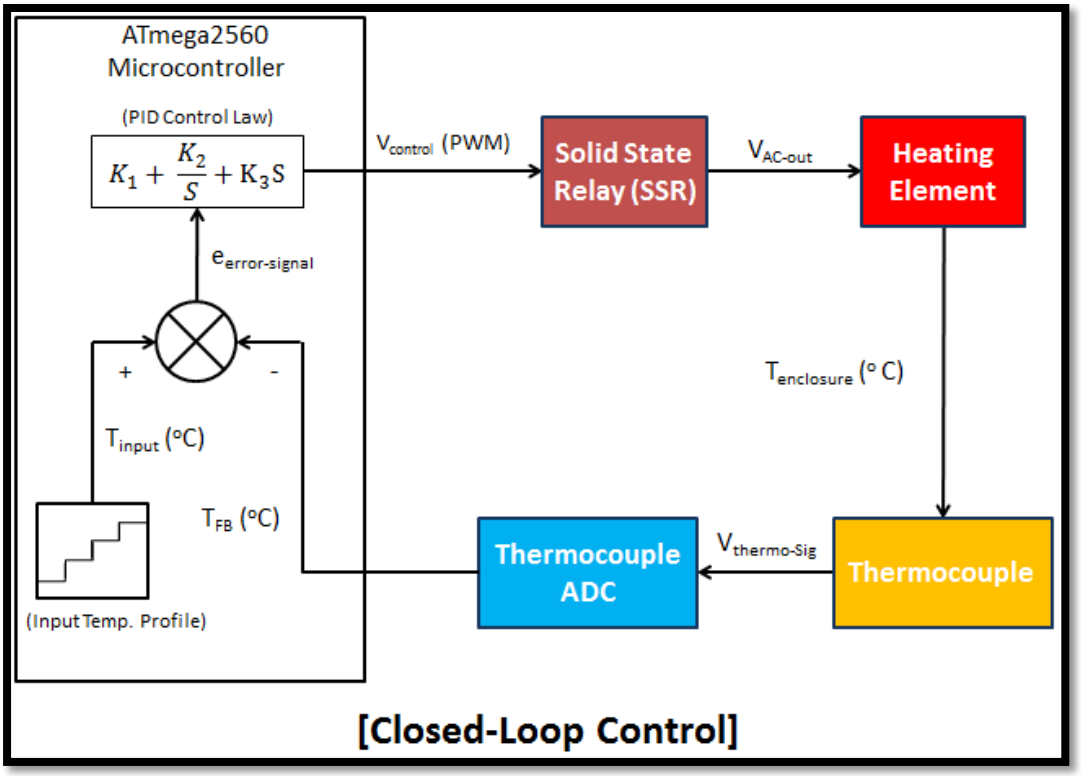


Figure 21: PID controller Block diagram for the heater

Due to the various issues experienced with the implementation of the thermocouple, accurate temperature feedback could not be achieved. There were three primary factors that affected the performance of temperature feedback obtained from the thermocouple. The first was the placement of the thermocouple within the baking environment (below the baking platform). Since infrared radiation was the primary source of heat and baking energy within the enclosure, the temperatures achieved in the enclosure above the baking platform significantly differed from the temperatures achieved below the platform (due to the underside being shielded from the majority of the IR radiation). This situation was confirmed utilizing a commercial thermocouple measurement setup, taking temperature measurements above and below the baking platform. The maximum temperature measured above the platform was around 270^{o}C (518^{o}F) and the maximum temperature below as around 105^{o}C (221^{o}F). Because of this, the temperature measured by the thermocouple was not accurate to the actual baking temperature. The second factor was that the type of thermocouple utilized (k-type, ungrounded) did not provided a fast enough response time to temperature changes under the conditions in the baking enclosure. And lastly, the third factor that may have contributed to inaccurate temperature feedback was a thin layer of corrosion on the exterior of the thermocouple that could have influenced the thermal conductivity of the couple. This corrosion was caused during initial calibration testing, in which the thermocouple was immersed in an ice bath (0^{o}C) and also in a bath of oil heated to 100^{o}C (commercial measure setup). The combination of oil and water may have caused reaction with

thermocouple material, as the listed stainless steel construction by manufacture was thought to be inaccurate.

Testing data collected during baking tests and the thermocouple calibration test data is given in Appendix E: Thermocouple Testing. The data indicates that while the thermocouple could produce accurate temperature data when immersed in a liquid (as in the calibration testing), however under the conditions in the baking environment (open-air convection) it could not provide accurate measurements. The temperatures measured in the enclosure significantly differed from the actual enclosure temperatures (both below and above the platform). It was attempted to incorporate a compensation algorithm to correct the inaccuracies in the readings, however the results produced by the thermocouple were not consistent enough to apply a satisfactory algorithm.

Due to these difficulties with feedback, open-loop control for the baking process was explored. Through experimentation implementing different power profiles (and subsequently temperature profiles, through data could not be taken), it was discovered that satisfactory results could be obtained (discussed further in Baking Testing Results). This greatly simplified the control law and coding implementation for the baking control. In addition, it was discovered that convection significantly improved the results of baking, so the 12 VDC convection fan (mentioned previously) was incorporated into the design. A block diagram of the open-loop control implementation utilized is shown in Figure 22.

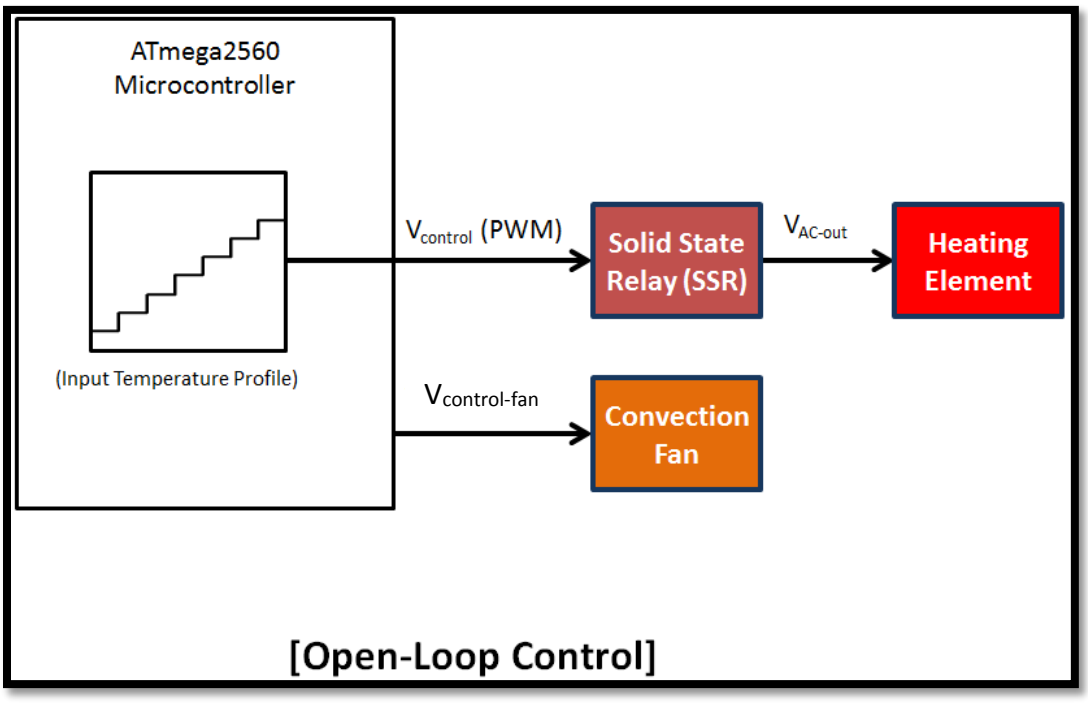


Figure 22: Open loop controller block diagram for the heater

In the open-loop control implementation, the microcontroller directly sends a predetermined power/temperature profile through SSR to the heating element. The controller would send an on-signal to the fan circuit during the baking period and then would keep the fan running for a predetermined cool-down period after baking was complete. The Arduino code written to implement the power/temperature profile is given in Appendix E. The code implements the profile by taking series of corresponding duty ratio values (represented as integer values ranging from 0 to 255) and time values (in seconds) and utilizing them as control points at which the rate of change of the duty ratio with respect to time changes (i.e. points at which the power/temperature slope of the profile changes). Between the control points the code linearly interpolates duty ratio values, utilizing a 1 second time step between interpolation points. This results in approximately linear duty cycle to time slopes (stair-step shape between interpolation points) giving the assumed approximate shape of the power/temperature profile achieved (though temperature data could not be taken to confirm). The general shapes of the profiles implement are illustrated in Figure 23.

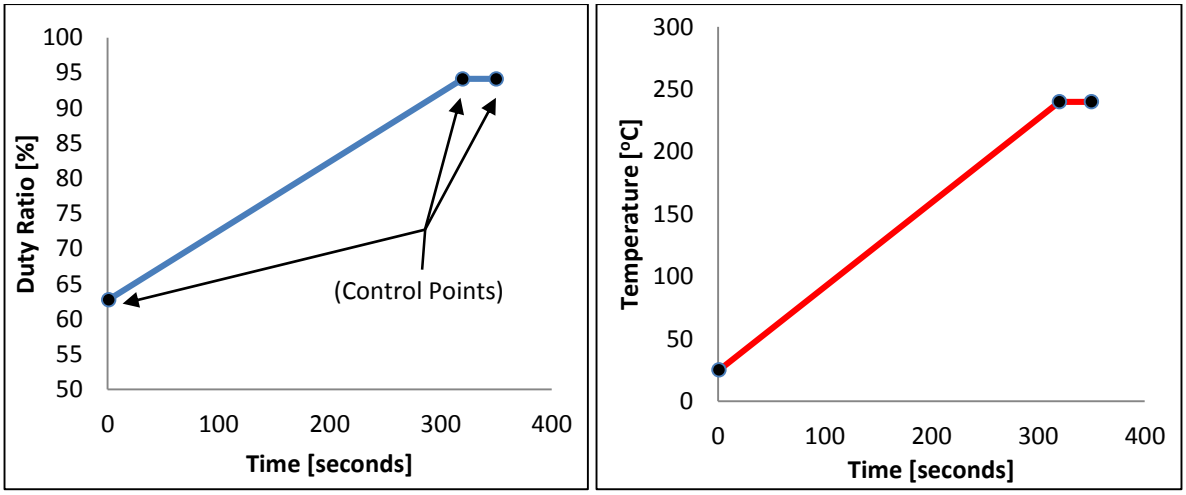


Figure 23: Illustration of duty ration profile (Left), with control points indicated, and assumed approximate corresponding temperature profile (Right)

Rationale for Selection of Microcontroller

In order for our circuit to perform with maximum simplicity and efficiency, the microcontroller was chosen appropriately. Some criteria required for our microcontroller are given below:

1. The output voltage of the microcontroller would ideally be 5 volts. Any output voltage smaller than that would create additional circuit complexities (such as a transistor) in order to boost it up above the 5 voltage threshold.
2. Due to the design of the circuit, there are multiple locations in our system that can trigger interrupts. Because each interrupt delivers a unique signal and corresponds to special instructions, there are multiple interrupt pins required.

3. In addition to the extra interrupt pins needed, there is also a significantly large amount of output pins also required.
4. In order to learn the most from the project, the microcontroller selected would be ideally be a microcontroller in contemporary use. Also, the selected microcontroller would also have a large amount of resources available for usage.
5. Finally, our selected microcontroller would ideally be physically portable, and small enough to be able to easily move along with our prototype unit.

After performing research, we narrowed our selection of microcontrollers to potentially use to the Arduino Uno, the Arduino Mega, and the HC12 microcontroller. The HC12 microcontroller violates criteria 4 and 5 above, while the Arduino Uno violates criteria 1, 2 and 3. This left the remaining Arduino Mega microcontroller (depicted below in Figure 24), which operated successfully for our purposes. The complete implementation of the circuit is described in Appendix I : Circuit.

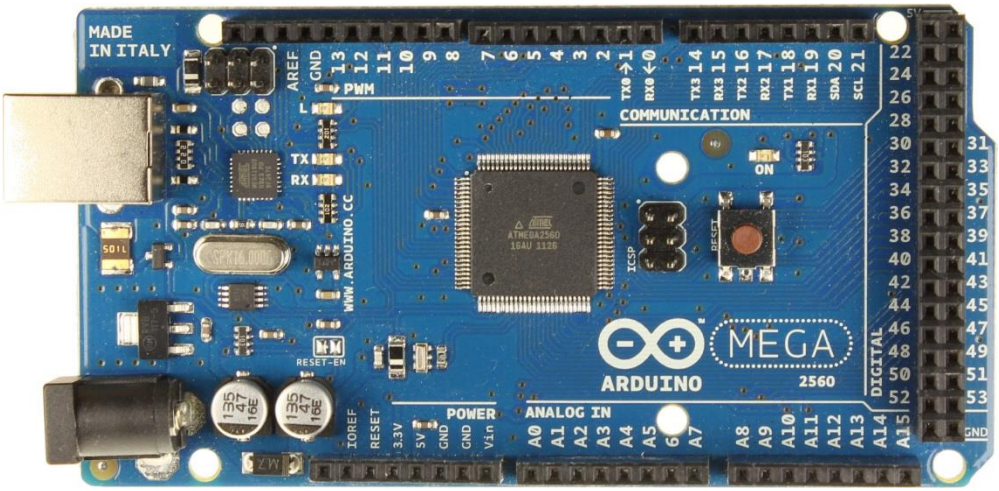


Figure 24: Arduino Mega

Baking Testing Results

Testing of the baking mechanism was conducted with the goal obtaining fully baked cookies (i.e. no raw portions internally or externally) within a time baking period of approximately five minutes. This testing was performed progressively, utilizing two different types of cookie dough and three different baking enclosure setups. The various tests performed were done to analyze the effects of the various parameters that effect cookie baking process in order to optimize these parameters to achieve the desired baking time and quality. The primary parameters found to most significantly affect the baking process were: distance from the infrared (IR) halogen heating element to the cookie; amount of IR radiation captured and directed toward the cookie in the enclosure; convection within the baking enclosure; and the temperature profile implemented. It

should be noted that the experimental results were analyzed empirically, assessing the outcome of baking tests in terms of the degree to which a cookie was baked (i.e. presence, amount and location of uncooked dough) and on the external appearance of the cookie (i.e. whether ideal golden-brown surface color was achieved). An additional goal for the baking process was to make it a "visual experience", so the enclosure setups were designed also with the intent that the operator could watch the cookie baking process take place.

Initial baking testing was performed utilizing was performed utilizing the baking enclosure setup shown in Figure 25.

Figure 25: Initial baking testing enclosure

This initial testing enclosure did not implement forced convection (only natural convection with in the enclosure) and variable power/temperature control had not yet been implemented (i.e. only full and zero power could be achieved). Cookies baked with the enclosure were baked at full power (i.e. direct connection of the element to 120 VAC wall power). Aluminum foil was utilized to reflect and capture the majority of the infrared radiation produced by the element toward the cookie. The distance from the cookie to the element was approximately 3.5 inches. The cookies that were baked utilizing this setup were fully cooked on the top and had ideal

coloration, however the underside and middle internal portion of the cookie remained undercooked. A top view of a cookie baked utilizing with the initial baking enclosure is shown in Figure 26. Cookies were baked over approximately five minutes to achieve the results in Figure 27, however longer cooking time could not be implemented as the surface of the cookie would burn.

Figure 26: Top view of cookie baked with initial baking enclosure (bottom of cookie undercooked)

Based on the results produced by the initial baking enclosure setup, the second enclosure setup incorporated forced convection through use of a small fan located below the baking platform. The second enclosure setup was initially tested without the use of reflective material to capture the IR radiation of the heating element, as the second enclosure was significantly smaller than the initial testing enclosure; setup is shown in Figure 27.

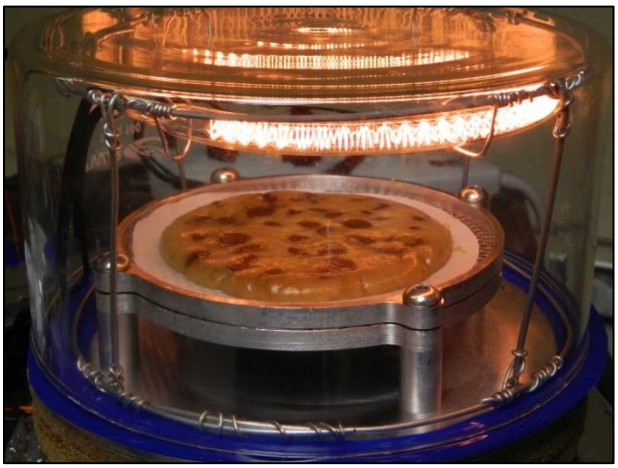


Figure 27: Second baking enclosure setup without reflective shielding for IR radiation capture

The distance from the element to the cookie was approximately 2.5 inches and the convection fan voltage was 5 VDC (providing low convective circulation in the enclosure). The second setup incorporated variable power/temperature control, and various power/temperature profiles were evaluated. All profiles were implemented over baking time periods ranging from five to six

minutes. The resulting cookies baked were undercooked in the middle section on the cookie (on top through to the bottom), with only the outer edges being partly cooked. The baking pattern observed is illustrated in Figure 28.

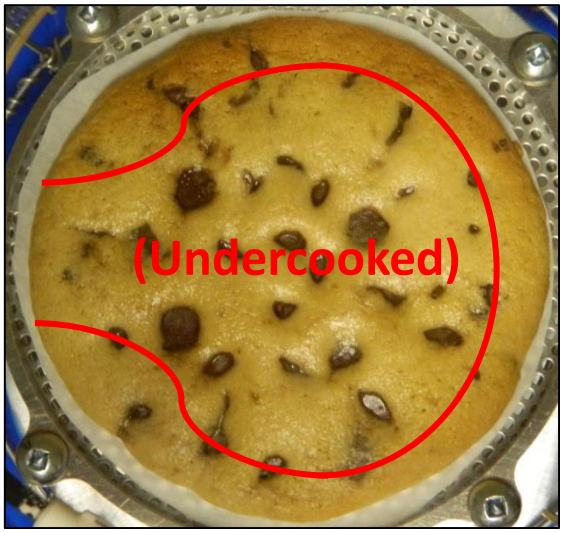


Figure 28: Top view of cookie baked with second baking enclosure setup without IR shielding (undercooked region of cookie outlined)

The cooked portions of the cookies baked with the setup shown in Figure 27 resembled shape of the heating element (i.e. the baked portions of the cookies on the outer edge were in the shape of the element). This was due to a large amount of the IR radiation from the element (which was the primary baking mechanism) being lost through the baking enclosure glass, resulting in only the edges of the cookies being exposed to significant amounts of radiation due to their closer proximity to the element. To decrease the radiation loss a reflective aluminum shield was placed over the upper portion of the baking enclosure as shown in Figure 29. To further improve the results, convection in the enclosure was also significantly increased by providing higher voltage (12 VDC) to the convection fan.

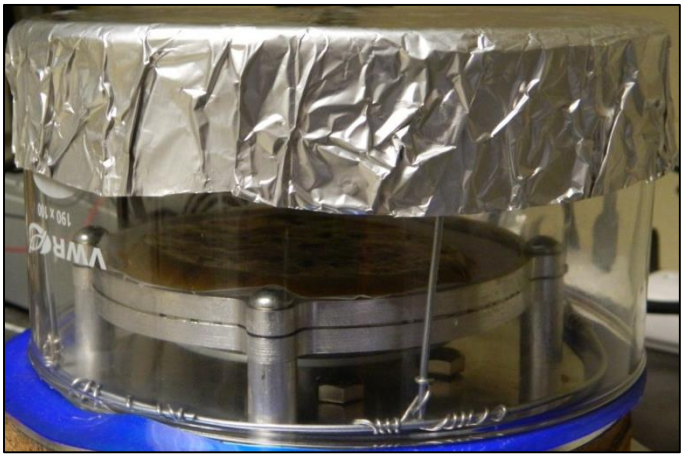


Figure 29: Second baking enclosure setup with partial IR reflective shield

Utilizing similar power/temperature profiles and baking times as for the previous setup, more thorough baking was achieved with a significantly smaller portion in the center of the cookies remaining undercooked consistently (in the same shape as for the previous setup). The observed baking pattern is shown in Figure 30.

Figure 30: Top view of cookie baked with second baking enclosure setup with IR shielding (reduced undercooked region indicated)

Utilizing the information gathered from the previous two baking enclosure setups, a final enclosure setup was implemented. The enclosure was designed to have a balance between IR radiation retention and aesthetic appeal for the user (i.e. designed so the user can watch baking take place). The top of the enclosure and certain portions of the sides were fitted with reflective shielding, while the rest of the enclosure remained unshielded for viewing of the baking process. The final baking enclosure setup is shown in Figure 31.

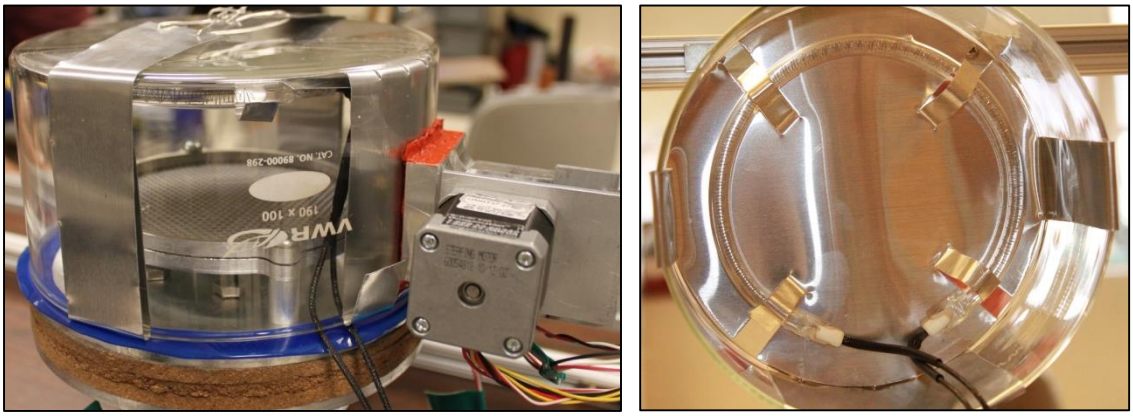


Figure 31: Final baking enclosure setup (balancing aesthetic and IR reflective shielding)

With the convection fan powered at 12 VDC, through a number of baking trials an optimal baking profile was determined, containing the following PWM and time control points (Note, DT = duty ratio): [62.75% DT, 1 sec], [94.12% DT, 320 sec], and [94.12% DT, 350 sec]. The total baking time was approximately 5 minutes and 50 seconds, with a cool-down period of 30 seconds. Utilizing this profile with the final enclosure setup thoroughly baked cookies with desired golden-brown coloration could be obtained. The final baking time achieved was five minutes and fifty seconds, slightly longer then the desired baking time of five minutes. An example of a thoroughly baked cookie produced using the given implementation is shown in Figure 32.

Figure 32: Top view of a thoroughly baked cookie produced utilizing optimized baking profile with final baking enclosure setup

Additional testing data collected to optimize the baking profile for the final enclosure is given in Appendix K: Additional Cookie Baking Test Data. It should also be noted that two different kinds of cookie dough were used during the course of the baking testing: (1) store-purchased chocolate chip cookie dough (used in testing for first two enclosure setups); and (2) homemade plain dough (used for testing for final enclosure setup). It was found that quality and time also was a function of dough thickness and type, so cookie dough pucks were made with relatively uniform dimensions; approximately 4 inches in diameter and 3/16 inch (0.1875 inch) thick.

Conclusion

Summary

CookieBot is a prototype cookie design and baking robot. Numerous results were discovered in the process of building and testing the prototype. Cookie baking was optimized over the course of the project.

Initially, with a radiation shield but no convection, the cookies were perfectly browned on the outside, but the bottom and inside were undercooked. This was rectified by adding a convection fan run on 5V DC power. Upon further testing, the convection fan did improve baking results, but the centers and bottoms of cookies remained undercooked after baking. A temperature power profile was implemented using PWM to control the intensity of the heating element. Another, smaller, radiation shield was also added in this iteration and the cookie baking was greatly improved, but still not perfect. In the final baking setup, PWM was used to control the heating, 12V DC was used to run the fan, and a minor heat shield was implemented to achieve a balance between aesthetics and cooking functionality. An example of a cookie baked using the final setup is shown in Figure 32: Top view of a thoroughly baked cookie produced utilizing optimized baking profile. Cookies baked in the final setup were thoroughly cooked, top and bottom, and were also a pleasant golden brown color: inviting themselves to be eaten.

Future Improvements

There are numerous improvements that can be made to the overall CookieBot system prototype. Due to time and budget constraints, certain elements could not be included in the initial project. There were also many design improvements that were conceptualized during the construction and testing phases. Some of the improvements are needed for proper functioning of the prototype as designed. The following is a list of possible improvements that could be made to CookieBot with short explanations behind their rationale.

1. Replacement of the faulty X-axis stepper motor.
    - The current X-axis motor is slightly broken. When sent a command to turn one full rotation, it only turns approximately ¾ of a revolution, but the exact amount is not reproducible. Linear offsetting was used to try and rectify this problem, but the problem was non-linear. Coming from each direction, the amount of ‘backlash’ (it was a broken motor, not true backlash) was different. It was also not reproducible was either direction. After connecting the Y-axis motor and having it work perfectly, the team came to the conclusion that the X-axis motor was broken. Replacing this stepper motor with a comparable one would make the printing much more accurate.

2. Upgrading to a higher torque stepper motor for lifting the oven lid.
    - The stepper motor used for lifting the lid on the oven compartment was not up to the task of torqueing at the hinge of the oven to lift the glass off the moving printer platform. This did not allow our system to complete the printing and cooking sequence in a continuous fashion. Two possible solutions to this problem are replacing the motor with a higher toque stepper motor or by creating a pulley system connected to the current motor that would allow the cover to be lifted off of the platform.

3. Installation of cable tracks.
    - The CookieBot prototype system has many wires hanging out all over the place and which have nothing to guide them or protect them during movement. Installation of cable tracks would rectify this problem. Cable tracks are plastic containers that surround wires and only allow them to move usually in one linear direction. They maintain a minimum bend radius so that wires do not experience fatigue over time. They move with the system so that wires have a predetermined path to take and have no chance of getting caught and pinched in the moving system.

4. Sealing the oven chamber using gasket.
    - The blue silicon gasket which can be seen in pictures above does not adequately seal the oven chamber while also letting the wires for the oven element escape. This causes the cookie not to cook as thoroughly as it would otherwise with a completely sealed chamber. The reason that a better gasket would improve the system is by sealing in all of the convective air and causing the cookie to cook faster.

5. Central placement of a heating element.
    - Cookies naturally cook from the outside to the inside, which causes the center to always be the least cooked portion. A possible solution to this would be to centrally place a resistive heating element in the cooking chamber. As it stands, the heating element surrounds the outside of the cookie and causes it to cook from the edges inward. The concept is that a centrally placed element would pinpoint the heat where needed to heat the cookie center. Presumably, the cookie would then cook primarily from the center, with the edges cooking at a higher rate, leaving the cookie perfectly baked throughout. This would have to be further investigated.

6. Multiple M&M dispensers for color printing.
    - The M&M dispenser successfully and repeatedly dispenses M&Ms on command. However, there is only one dispenser for all of the colors together. It would be relatively simple, but time consuming, to create 5 more dispensers, one for each color of M&M: red, orange, brown, blue, green, and yellow. The code would then have to be modified to account for the different colors. The GUI would need to have a color option in order to communicate color choices to the system. The code would then have to be modified to account for the placement of each dispenser. This would be a relatively simple change for a large feature gain. The M&Ms would also have to be physically sorted, but this could be solved by developing a sorting machine, such as exist for coin sorting.

7. Making the Y-axis longer by using a different motor mounting scheme.
    - Some range of movement for the Y-axis was lost by adding the motor within the frame of our system, as can be seen in the pictures. Ideally, the stepper motor would be mounted off of the side of the machine and would not impede in the range of motion of the machine. This would allow for more room to move the cookie platform and for printing.

Bibliography

Arduino. (2013, June). *Arduino - Homepage*. Retrieved from Arduino.cc: http://www.arduino.cc/

Dieter, G. E. (1997). *ASME Handbook, Volume 20 - Materials Design and Selection.* ASME.

McCauley, M. (2010). *AccelStepper library for Arduino* . Retrieved from AccelStepper: http://www.airspayce.com/mikem/arduino/AccelStepper/

NSF/ANSI 51 Food Equipment Materials. (1997). NSF/ANSI.

Piazza, L., & Masi, P. (1997). Development of Crispness in Cookies During Baking in an Industrial Oven. Milan, Italy.

Pololu. (2001). *A4988 Stepper Motor Driver Carrier*. Retrieved from Pololu Robotics & Electronics: http://www.pololu.com/catalog/product/1182

Processing 2. (n.d.). *Processing.org*. Retrieved from Processing: http://www.processing.org/

Ragan, J. (n.d.). *Bad Candy*. Retrieved from http://www.bad-candy.com/old/bc3/reader/000904.shtml

Schmalz, B. (2012). *EasyDriver Stepper Motor Driver*. Retrieved from Schmalz Haus LLC: http://schmalzhaus.com/EasyDriver/

Torghele, C. (2009). *Let's Pizza*. Retrieved from www.letspizza.co.uk

Appendix A: Budget and Parts List

Shown below in Table 1 is the budget and parts list of all the components that was used in the assembly. The total allowance of money was $1192, which ended up nearly matching exactly.

Table 1: Budget and Parts List

Part	Quantity	Price/Unit	Total Price (w/Shipping)	Seller
Thermocouple IC ADC Chip	1	$18.80	$18.80	Digi-key
Thermocouple	1	$15.89	$15.89	Auber Instruments
High Temperature Silicone Sealent	1	$20.03	$20.03	Amazon.com
Halogen Heating Element	1	$27.22	$27.22	Amazon.com
Heating Enclosure Glass	1	$55.12	$55.12	UW Chemistry Store
Printer Parts	1	$10.00	$10.00	RePC
Fast-Travel Precision ACME Round Nuts	2	$32.98	$65.96	McMaster Carr
Fast Travel Threaded Rod	1	$55.92	$55.92	McMaster Carr
Ball Bearings	5	$6.81	$34.05	McMaster Carr
X-axis roller wheels	2	$19.98	$39.96	McMaster Carr
Solenoid	1	$30.27	$30.27	McMaster Carr
Dispenser Tubing	1ft	$-	$2.57	McMaster Carr
PVC Cement, Clear	1	$4.37	$4.37	McMaster Carr
Dispenser Funnel	1	$2.43	$2.43	McMaster Carr
Shipping		$-	$5.79	McMaster Carr
Cork Insulation for Heater	1	$23.43	$23.43	Grainger
Easy Drivers	3	$16.18	$48.54	SparkFun
Breadboard Jumper Connectors	1	$6.45	$6.45	Amazon.com
80/20	1 order	$-	$221.86	80/20
Arduino Mega	1	$44.95	$44.95	Amazon.com (MP3Carr)
MAX31855 breakout board		$-	$21.32	Adafruit.com
Perforated Plate		$-	$21.75	onlinemetals.com
Easy Drivers	2	$15.95	$31.90	Amazon.com
Stepper Motor	1	$25.22	$25.22	sparkFun
Dispenser Tubing	1ft	$-	$2.57	McMaster Carr
Shipping		$-	$5.18	McMaster Carr
Nylon Gear Rack	1ft	$-	$6.39	McMaster Carr
Nylon Gear Pinion	1	$4.11	$4.11	McMaster Carr

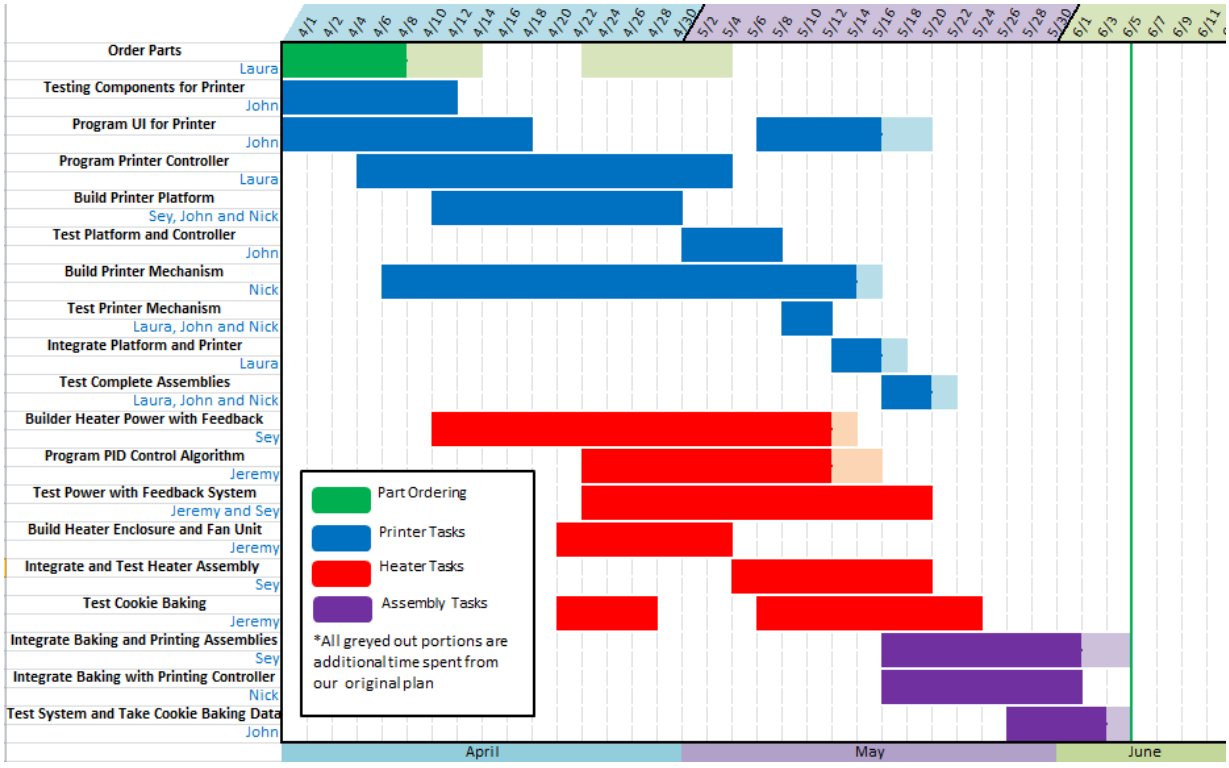
Structure fasteners, fittings, machine screws, miscellanious components		$-	$102.10	Hardwicks
Logic IC's		$-	$19.68	Radioshack
Easy Driver	1	$18.94	$18.94	Amazon.com
Pololu Stepper Driver	1	$10.00	$10.00	Classmate
Mini-M&M's	4.5 lb	$-	$16.73	Amazon.com
Electronic Components		$-	$70.39	Vetco
Total Spent		**$-**	**$1,089.89**	
Remaining Budget		**$-**	**$2.11**	

The budget and parts list of previous iterations can be found in Appendix C: Previous Iterations of Design.

Appendix B: Gantt Chart

Shown below in Figure 33 is the Gantt Chart as designed and modified by our group through the quarter. Several significant changes had to be made due to some unforeseen factors that were not accounted for. The original Gantt Chart is shown below as all the solid colored blocks. The faded-out portions signifies the extra time that was taken on each task. On the left of the figure, the task is shown along with each task lead.

Figure 33: Gantt Chart



Significant delays were caused due to confusion and last-minute changes when ordering parts, which had delays that cascaded into multiple other tasks. Also, the integration and debugging phase of the project took significantly longer than expected.

Appendix C: Previous Iterations of Design

Initial Budget before Scholarship Acceptance

Shown below in
Table 2 is the original budget and parts list that we created before we obtained the Mary Gate Undergraduate Research Award. The maximum budget available was $350.

Table 2: Original Budget

Part Name	Quantity	Price/Unit	Total Price	Seller
Fast-Travel Precision Acme Round Nuts	2	$32.98	$65.96	McMaster
Threaded Rod	1	$30.00	$30.00	McMaster
80/20	150	$0.23	$34.50	80/20
Ball Bearings	7	$6.81	$47.67	McMaster
Glass Cover	1	$10.00	$10.00	UW Chem Store
Heating Element	1	$15.17	$15.17	Amazon
Printer Parts	2	$5.00	$10.00	RE-PC
Motor Drivers	2	$14.95	$29.90	SparkFun
X-axis Wheels	0	$19.98	$-	McMaster
Thermocouple	0	$27.00	$-	omega.com
Amplifier Chip	1	$3.48	$3.48	DigiKey
Guide Rods		$12.69	$-	McMaster
Stepper Motors	0	$53.00	$-	Omega.com
Convection Oven Fan	0	$90.13	$-	appliancezone.com
Glass tube	1	$4.76	$4.76	McMaster
Plastic Funnel	1	$1.12	$1.12	McMaster
Solenoid	1	$15.27	$15.27	McMaster
Cookie Dough	1	$25.00	$25.00	
Shipping Costs	1	$50.00	$50.00	
Arduino	0	$40.00	$-	Amazon
Total			**$342.83**	

Appendix D : Candy Diameter Distributions

A representative bag of miniature chocolate chips was bought and the diameters of several dozen were measured using digital calipers. The resulting distribution is shown below in Figure 34: Mini-Chocolate Chip Distribution below Figure 10. The red curve is a normal curve fitted to the data. As can be seen, the range of the distribution is 0.05 inches, approximately 25% of the mean value of 0.21 inches.

This is a fairly wide distribution and skewed as well with more chocolate chips tending towards smaller diameters but larger diameters occurring much past the mean. This most likely is due to the manufacturing process that chocolate chips are created with. A fixed width nozzle extrudes a dab of chocolate onto a baking sheet. The dab of chocolate is at least a certain size, but could be larger depending on if more chocolate was drawn out by friction from melted chocolate's high viscosity.

The same experiment was repeated with a representative tube of miniature M&Ms. the results of this can be seen in Figure 35: M&M Distribution Data, which is a histogram of the data. The range as a percentage of the mean is $(0.032/0.362) * 100\% = 8.8\%$, which a large improvement is over the 25% for chocolate chips.

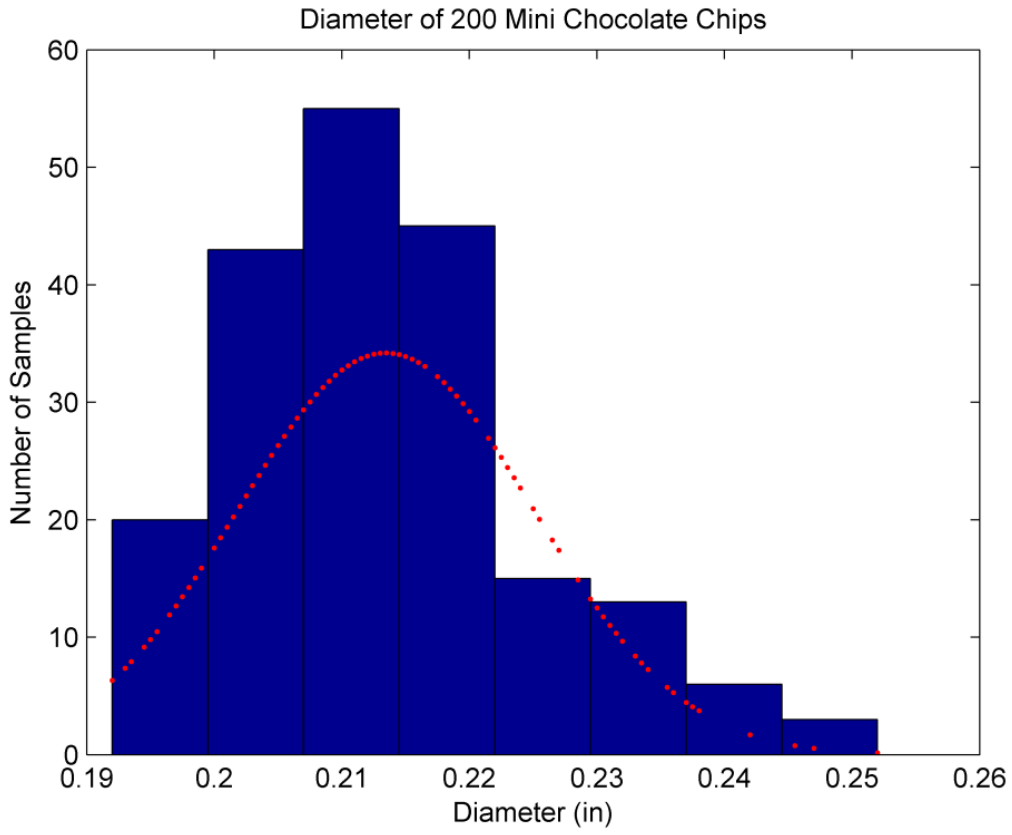


Figure 34: Mini-Chocolate Chip Distribution

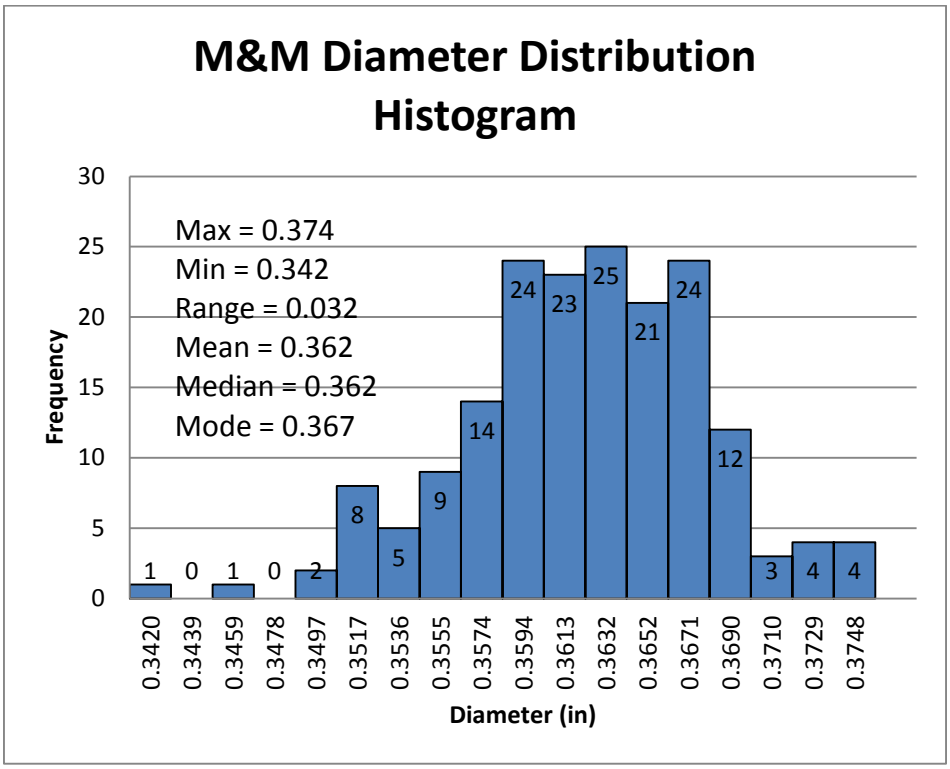


Figure 35: M&M Distribution Data

Appendix E: Thermocouple Testing

To monitor the temperature of the oven it was decide to use a K-Type thermocouple. This thermocouple was going to be used as a feedback for a closed loop control system implementation. To collect the data from the thermocouple an Analog to Digital Converter (ADC) IC was used. The team selected MAX31855 Cold-Junction Compensated Thermocouple-to-Digital Converter to perform temperature conversion for the thermocouple. Once the IC chip was mounted and the microcontroller was able to communicate with it, the team ran some temperature tests to check integrity of the setup (calibration testing). Figure 36 displays the different test for a shielded thermocouple wire and an unshielded wire. These thermocouples were tested at ice bath, room temperature, and boiling water. The results of the tests show that a lot of noise was introduced to the reading by the unshielded thermocouple, as seen in red color in Figure 36. Hence it was decide to use the shielded thermocouple as the temperature feedback for the close-loop PID controller.

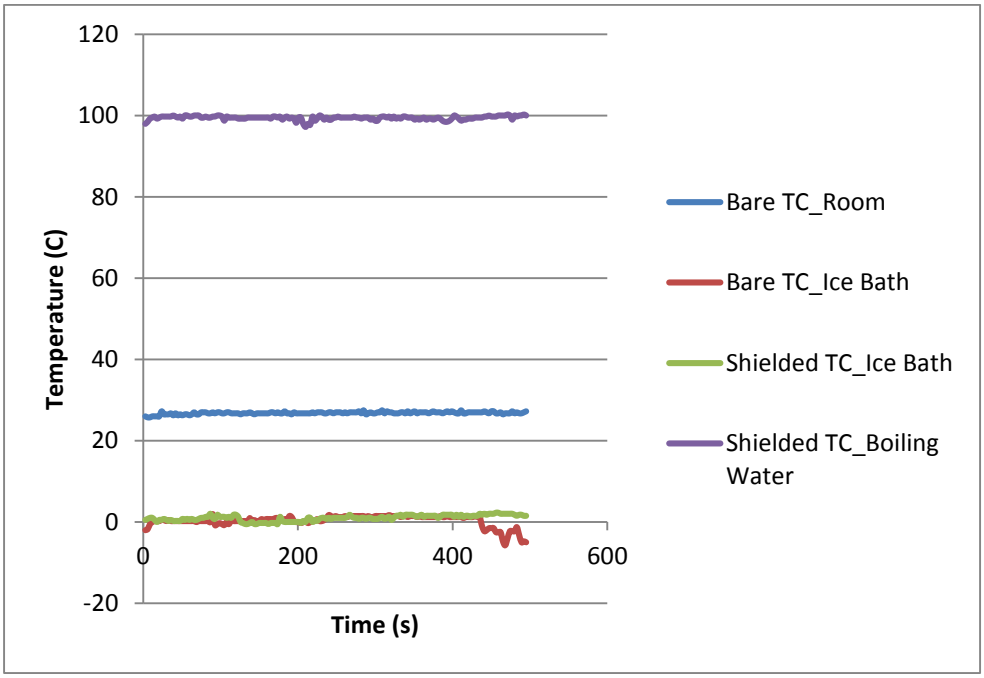


Figure 36: Thermocouple test data at different temperatures (calibration test data)

A graph of temperature data obtained utilizing the thermocouple for a baking test run is shown in Figure 37. The graph illustrates the general shape of the temperature data obtained from the various baking testing runs performed, and also the typical temperature values obtained. The temperature values and shape of the graph differed significantly from the expected profile. Utilizing a comercial thermocouple temperature measurement setup to compare values, it was determined that the thermocouple utilized for the baking enclosure was producing inaccurate readings.

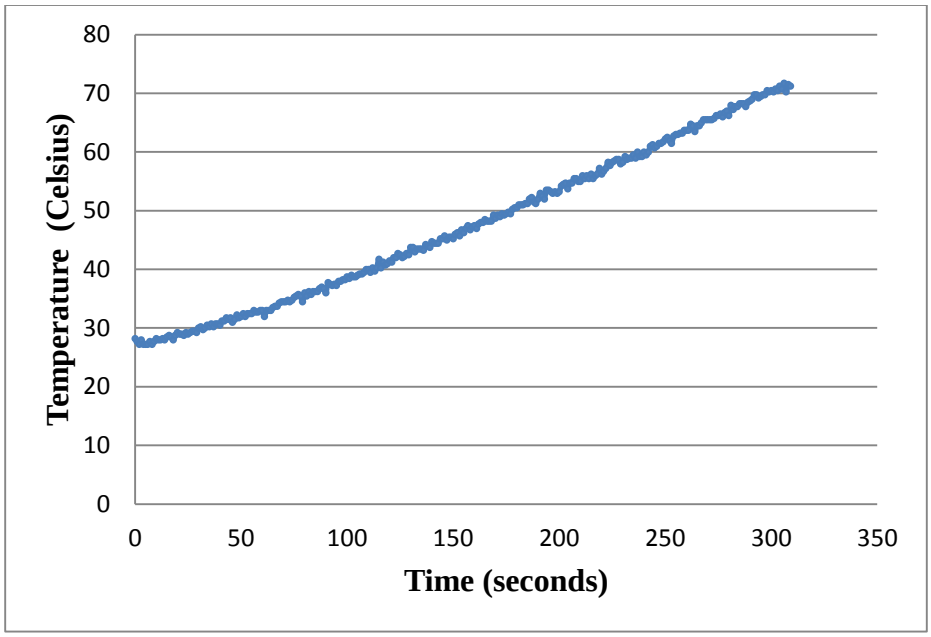


Figure 37: Example set of temperature data recorded by thermocouple during baking testing (bake time was 5 minutes, convection power at 12 VDC)

Appendix F: GUI Program

Shown below is the code for the program that creates the graphical user interface, used to accept input from the user to define the design to be printed on the cookie. This program was developed in a language called Processing.

```
/*******************************************************************************/
import processing.serial.*;

private int scaling = 35;                    // Grid size
private int numChips = 11;                   // Height and width of 2D array
private Chip[][] chips;                      // 2D grid
private int circleW = 428, circleH = 428, circleX = 260, circleY = 260;
                                             // Cookie bounds
private int rad = circleW/2;                 // Cookie radius
private PImage chipPic;                      // Picture of chocolate chip on cookie
private PFont buttonFont;                    // Button fonts
private Serial arduino;                      // Serial Port Communication with Arduino

void setup() {
       // Initialize window
       size(550,700);
       background(255);
       image(loadImage("cookie.jpg"), 50, 50, circleW, circleH);
       //ellipse(circleX, circleY, circleW, circleH);

       // Initialize serial port
       arduino = new Serial(this, Serial.list()[1], 57600);

       chips = new Chip[numChips][numChips]; // Initialize 2D grid
       chipPic = loadImage("chip.jpg"); // Load chip image
       buttonFont = createFont("Georgia", 32); // Initialize buttons

       // Initialize cookie grid that lie inside the cookie area
       int parseX = 0;
       for(int x = 52; x < 66+numChips*scaling; x += scaling) {
              int parseY = 0;
              for(int y = 52; y < 66+numChips*scaling; y += scaling) {
                     if(circleCheck(x,y)) {
                            chips[parseX][parseY] = new Chip(x, y,
                                                  get(x, y, scaling, scaling));
                     }
                     parseY++;
              }
              parseX++;
       }
}

void draw() {
       while(arduino.available() > 0) {
              println(arduino.read());
       }
```

```
	// Checks if chip display needs to be updated
	for(int x = 0; x < numChips; x++) {
		for(int y = 0; y < numChips; y++) {
			if(chips[x][y] != null){
				chips[x][y].update();
				chips[x][y].display();
			}
		}
	}

	// Clear all chips button
	fill(220, 50, 50);
	rect(90, 550, 150, 70, 10);
	textFont(buttonFont);
	fill(0);
	text("Clear All", 105, 595);
	if(mousePressed && mouseX > 90 && mouseX < 240 &&
	   mouseY > 550 && mouseY < 620) {
		clearAll();
	}

	// Done button
	fill(50, 200, 70);
	rect(290, 550, 150, 70, 10);
	textFont(buttonFont);
	fill(0);
	text("DONE!", 315, 595);
	if(mousePressed && mouseX > 290 && mouseX < 440 &&
	   mouseY > 550 && mouseY < 620) {
		submit();
	}
}

/**
 * Checks if grid size defined by x and y is located inside cookie region
 **/
boolean circleCheck(float x, float y) {
	if ((sq(x-circleX) + sq(y-circleY)) < sq(rad-6) &&
	     (sq((x+scaling)-circleX) + sq((y+scaling)-circleY)) < sq(rad-6) &&
	     (sq(x-circleX) + sq((y+scaling)-circleY)) < sq(rad-6) &&
	     (sq((x+scaling)-circleX) + sq(y-circleY)) < sq(rad-6)) {
	       return true;
	} else {
		return false;
	}
}

/**
 * Clears all chip images from cookie
 **/
void clearAll() {
	for(int x = 0; x < numChips; x++) {
		for(int y = 0; y < numChips; y++) {
			if(chips[x][y] != null){
```

```
                        chips[x][y].clicked = false;
                }
            }
        }
}


/**
 * Send out chip coordinate positions to Arduino
 **/
void submit() {
        for(int x = 0; x < numChips; x++) {
            for(int y = 0; y < numChips; y++) {
                if(chips[x][y] != null && chips[x][y].clicked == true){
                    arduino.write(", " + (10 - x) + ", " + (10 - y));
                }
            }
        }

        while(arduino.available() <= 0) {
        }

        while(arduino.available() > 0) {
            println(arduino.read());
        }
}

/**
 *
 **/
class Chip {
        public int x,y;
        private boolean clicked;
        private PImage back;

         /**
          * Creates chip object
          **/
         Chip(int x_, int y_, PImage pic) {
              x = x_;
              y = y_;
              clicked = false;
              back = pic;
        }

         /**
          * If clicked has been set to true, chip image will be placed on cookie
          **/
        void display() {
              stroke(0);

              if(clicked) {
                    image(chipPic, x, y, scaling, scaling);
              } else {
                    image(back, x, y);
```

```
            }
            noFill();
            rect(x, y, scaling, scaling);
        }

        /**
         * Checks if mouse was pressed in valid grid location. If so, sets varable
         * clicked to true
         **/
        void update() {
            if(mousePressed) {
                if(mouseButton == LEFT) {
                    if((mouseX > x) && (mouseX <= x + (scaling)) &&
                        (mouseY > y) && (mouseY <= y + (scaling))) {
                            clicked = true;
                    }
                } else {
                    if((mouseX > x) && (mouseX <= x + (scaling)) &&
                        (mouseY > y) && (mouseY <= y + (scaling))) {
                            clicked = false;
                    }
                }
            }
        }
}
```

Appendix G: Embedded Circuit Code

Shown below is the code used to control all of the interrupt, driver, and sensor signals. This program was developed in the native Arduino language.

```
/*******************************************************************************/
#include <Stepper.h>
#include <SoftwareSerial.h>
#include <AccelStepper.h>
#include "Adafruit_MAX31855.h"

// X Motors
#define X_MOTOR_DIR_PIN 22            // Direction of the X position motor
#define X_MOTOR_STEP_PIN 25           // Steps sent to the X position motor
#define X_MICROSTEPS_LEFT 1860L       //
#define X_MICROSTEPS_RIGHT 1860L      //
#define X_MICROSTEPS_PER_INCH 2013L   //
#define X_GRID_LENGTH_MICROSTEPS 800L //
// Y Motors
#define Y_MOTOR_DIR_PIN 26            // Direction of the Y position motor
#define Y_MOTOR_STEP_PIN 29           // Steps sent to the Y position motor
#define Y_MICROSTEPS_PER_INCH 1600L   // 16 microsteps per Inch
#define Y_GRID_LENGTH_MICROSTEPS 600  // Microsteps (0.375 inch_grid * 1600
                                      // microsteps/inch)
// Accel Motors
#define MAX_SPEED 2000.0              // |
#define SPEED 200.0                   // | Big Motors
#define ACCEL 2000.0                  // |
// Dispenser
#define DISPENSE_STEPS 95             // Place M&M
#define DISPENSE_STEPS_AGAIN 10       // Try again
#define DISPENSE_DIR_PIN 24           // Direction of the chip belt
#define DISPENSE_STEP_PIN 23          // Steps sent to the chip belt
#define DISPENSE_LIMIT_PIN 41         // Rack length limit
#define MAX_CHIPS 88                  // Max number of M&M's that can be dispensed on
                                      // one cookie
#define ROTATE_BACK 1600
// Coordinate System
#define PRINT_X0_MICROSTEPS 9813L     // Starting X for printer (in)
#define PRINT_Y0_MICROSTEPS 10920L    // Starting Y for printer (in)
#define HEAT_X0_MICROSTEPS 40000L     // Starting X for heater (in)
#define HEAT_Y0_MICROSTEPS 9720L      // Starting Y for heater (in)
// Heating
#define HEAT_MOTOR_DIR_PIN 28
#define HEAT_MOTOR_STEP_PIN 27
#define HEAT_FAN 38
#define ROTATE_LID_MICROSTEPS 200
#define MAX_HEAT_SPEED 100.0          // |
#define HEAT_SPEED 50.0               // | Heat Motors
#define HEAT_ACCEL 100.0              // |

// Global Variables
AccelStepper stepperX(AccelStepper::DRIVER, X_MOTOR_STEP_PIN, X_MOTOR_DIR_PIN);
```

```
AccelStepper stepperY(AccelStepper::DRIVER, Y_MOTOR_STEP_PIN, Y_MOTOR_DIR_PIN);
AccelStepper stepperHeat(AccelStepper::DRIVER, HEAT_MOTOR_STEP_PIN,
                         HEAT_MOTOR_DIR_PIN);
volatile int currentX;               // Current X printing position
volatile int currentY;               // Current Y printing position
int serialData[MAX_CHIPS];
int chipCount;
int currentChip;
volatile boolean toPrint;
volatile boolean heatLid;
volatile boolean notDispensed;
boolean toHeat;
boolean calibrate;

// Heating Global Variable
// Setting up arduino PWM pin
int PWMpin = 8;                  // PWM pin
int clearPrescalers = 7;         // binary 00000111 to clear bits of timer register
int prescaler = 5;               // prescaler to set PWM frequency to 30Hz

// Oven Fan
int fanPin = 13;                 //convection fan control pin
int coolDownPeriod = 30;         //time that convection fan is on after baking[sec]
int coolDownCounter = 0;         //counter to implement cool-down period

//Setting up the temperature profile
int tempProfile[] = {180,240,240};    //PWM duty ratio values for profile; range of
                                      //integer values: 0 to 255
int timeArray[] = {1,320,350};        //Time profile in [sec];
int tempPoints = 3;                   //Points where dT/dt changes
int MaxTempLevel = 255;               //Max input integer for PWM (100% duty cycle)
double MaxOvenTemp = 240.00;          //Max safe oven temperature 9 (degrees C)
int bakingComplete = 0;               //indicates if one bake cycle is complete
int timeInc = 1;                      //Time increment between inter

//Setting up thermocouple pins
int thermoDO = 3;
int thermoCS = 4;
int thermoCLK = 5;

Adafruit_MAX31855 thermocouple(thermoCLK, thermoCS, thermoDO);


void setup() {
       Serial.begin(57600);

       // Motors
       pinMode(X_MOTOR_DIR_PIN, OUTPUT);
       pinMode(X_MOTOR_STEP_PIN, OUTPUT);
       pinMode(Y_MOTOR_DIR_PIN, OUTPUT);
       pinMode(Y_MOTOR_STEP_PIN, OUTPUT);
       // Dispenser
       pinMode(DISPENSE_DIR_PIN, OUTPUT);
       pinMode(DISPENSE_STEP_PIN, OUTPUT);
       pinMode(DISPENSE_LIMIT_PIN, INPUT);
```

```
// Heating
pinMode(HEAT_MOTOR_DIR_PIN, OUTPUT);
pinMode(HEAT_MOTOR_STEP_PIN, OUTPUT);
pinMode(HEAT_FAN, OUTPUT);

// Interrupts
attachInterrupt(0, heater, RISING);       // Pin 2
attachInterrupt(1, noDispense, FALLING);  // Pin 3
attachInterrupt(2, leftLimit, RISING);    // Pin 21
attachInterrupt(3, rightLimit, RISING);   // Pin 20
attachInterrupt(4, frontLimit, RISING);   // Pin 19
attachInterrupt(5, backLimit, RISING);    // Pin 18

// Set up motor control
// X Motor
stepperX.setMaxSpeed(MAX_SPEED);
stepperX.setSpeed(SPEED);
stepperX.setAcceleration(ACCEL);
stepperX.setCurrentPosition(0);

// Y Motor
stepperY.setMaxSpeed(MAX_SPEED);
stepperY.setSpeed(SPEED);
stepperY.setAcceleration(ACCEL);
stepperY.setCurrentPosition(0);

// Heater Motor
stepperHeat.setMaxSpeed(MAX_HEAT_SPEED);
stepperHeat.setSpeed(HEAT_SPEED);
stepperHeat.setAcceleration(HEAT_ACCEL);
stepperHeat.setCurrentPosition(0);

// Starting position
currentX = 0;
currentY = 0;
chipCount = 0;
currentChip = 0;
toPrint = true;
toHeat = false;
notDispensed = true;
calibrate = true;
heatLid = true;

// Heating Setup
//PWM
TCCR4B &= ~clearPrescalers; // clears last 3 bits in timer register for pins 9
                            // and 10
TCCR4B |= prescaler;        // sets PWM freq. for pins 9 and 10 to 30Hz
pinMode(PWMpin,OUTPUT);         // sets the pin as output

// Oven Fan
pinMode(fanPin,OUTPUT);

//Thermocouple
Serial.println("Oven Temperature (degrees C):");
```

```
	delay(500); //wait for MAX chip to stabilize
}

void loop() {
	static boolean coolDown = true;

	if(Serial.available() > 0) {
	// Calibrate
		if(calibrate) {
			zeroHeat();
			zeroX();
			zeroY();
			calibrate = false;
			toPrint = true;
		}

		//Read coordinates from serial ports
		while(Serial.available() > 0) {
			serialData[chipCount] = Serial.parseInt();
			chipCount++;
		}

		printMM(serialData, chipCount, true);

// Move to heating! ////////////////////////////////////////////////////////////////
		if(toHeat) {
		moveMotorsTogether(stepperX.currentPosition(),
					stepperY.currentPosition(),
					stepperX.currentPosition(),
					-HEAT_Y0_MICROSTEPS);
		moveMotorsTogether(stepperX.currentPosition(),
					stepperY.currentPosition(),
					HEAT_X0_MICROSTEPS,
					stepperY.currentPosition());

		delay(1000);
		}

		// Bring glass down
		heatLid = true;
		stepperHeat.moveTo(1600);
		while(heatLid) {
			stepperHeat.run();
		}

		// COOK!
		if(bakingComplete == 0) {
			ovenFan(HIGH);
			bakingProfile(1);
			bakingComplete = 1;
			ovenFan(LOW);
		}

		// Cool Down
		while(coolDown) {
```

```
                        if(coolDownCounter <= coolDownPeriod) {
                                ovenFan(HIGH);
                        } else {
                                ovenFan(LOW);
                                coolDown = false;
                        }

                        analogWrite(PWMpin,0);
                        coolDownCounter++;
                        delay(1000);
                }
        }
}

void rotate(int steps, float speed, char dirPin, char stepPin) {
        // Rotate a specific number of microsteps (8 microsteps per step) - (negitive
        // for reverse movement) speed is any number from .01 -> 1 with 1 being fastest
        // - Slower is stronger
        int dir = (steps > 0)? HIGH:LOW;
        steps = abs(steps);

        digitalWrite(dirPin, dir);

        float usDelay = (1/speed) * 70;

        for(int i=0; i < steps; i++){
                digitalWrite(stepPin, HIGH);
                delayMicroseconds(usDelay);

                digitalWrite(stepPin, LOW);
                delayMicroseconds(usDelay);
        }
}

void printMM(int coordinates[], int mmCount, boolean goToPrint) {
        // Move to printing!
        if(goToPrint) {
                Serial.write(5);
                moveMotorsTogether(stepperX.currentPosition(),
                                   stepperY.currentPosition(),
                                   PRINT_X0_MICROSTEPS, -PRINT_Y0_MICROSTEPS);
        }

        delay(1000);

        // Print!
        int i;
        for(i = 0; i < mmCount; i = i + 2) {
                moveMotorsTogether(stepperX.currentPosition(),
                                   stepperY.currentPosition(),
                                  (coordinates[i] * -X_GRID_LENGTH_MICROSTEPS),
                                  (coordinates[i + 1] * Y_GRID_LENGTH_MICROSTEPS));

                delay(1000);
```

```
                // Dispense!
                rotate(DISPENSE_STEPS, 0.1, DISPENSE_DIR_PIN, DISPENSE_STEP_PIN);
                delay(500);
                while(notDispensed) {
                        rotate(DISPENSE_STEPS_AGAIN, 0.1, DISPENSE_DIR_PIN,
                                DISPENSE_STEP_PIN);
                        delay(500);
                }
                notDispensed = true;

                if(digitalRead(DISPENSE_LIMIT_PIN) == HIGH) {
                        i = mmCount;
                        rotate(ROTATE_BACK, 0.1, DISPENSE_DIR_PIN, DISPENSE_STEP_PIN);
                }

                if(i == (chipCount - 1)) {
                        toHeat = true;
                        toPrint = false;
                }
        }
}

void moveMotorsTogether(long X0, long Y0, long X1, long Y1) {
        long diffX = X1 - X0;
        long diffY = Y1 - Y0;

        stepperX.moveTo(X1);
        stepperY.moveTo(Y1);
        long deltaX = abs(diffX);
        long deltaY = abs(diffY);

        // Need to move only in the Y
        if(deltaX == 0) {
                while(stepperY.distanceToGo() != 0) {
                        stepperY.run();
                }
        // Need to move only in the X
        } else if(deltaY == 0) {
                while(stepperX.distanceToGo() != 0) {
                        stepperX.run();
                }
        } else {
        // Need to move more in the X
                if(deltaX > deltaY) {
                        float deltaErr = ((float)deltaY) / ((float) deltaX);
                        stepperY.setMaxSpeed(deltaErr * MAX_SPEED);
                        stepperY.setAcceleration(deltaErr * ACCEL);
                // Need to move more in the Y
                } else if(deltaX < deltaY) {
                        float deltaErr = ((float) deltaX) / ((float)deltaY);
                        stepperX.setMaxSpeed(deltaErr * MAX_SPEED);
                        stepperX.setAcceleration(deltaErr * ACCEL);
                }

                while(stepperY.distanceToGo() != 0 && stepperX.distanceToGo() != 0) {
```

```
                    stepperY.run();
                    stepperX.run();
             }
       }
}

// INTERRUPTS ////////////////////////////////////////////////////////////////////
void heater() {
       heatLid = false;
}

void noDispense() {
       Serial.write(3);
       notDispensed = false;
}

void leftLimit() {
       stepperX.setCurrentPosition(0);
       stepperX.moveTo(0);
}

void rightLimit() {
       currentX = 0;
       stepperX.setCurrentPosition(0.0);
       stepperX.moveTo(0);
}

void frontLimit() {
       currentY = 0;
       stepperY.setCurrentPosition(0.0);
       stepperY.moveTo(0);
}

void backLimit() {
       stepperY.setCurrentPosition(0);
       stepperY.moveTo(0);
}
////////////////////////////////////////////////////////////////////////////////

// CALIBRATION FUNCTIONS /////////////////////////////////////////////////////////

void zeroHeat() {
       stepperHeat.moveTo(-1600);
       while(heatLid) {
              stepperHeat.run();
       }
       stepperHeat.setCurrentPosition(0);
}

void zeroX() {
       moveMotorsTogether(stepperX.currentPosition(), stepperY.currentPosition(),
                          (-30L * X_MICROSTEPS_PER_INCH), stepperY.currentPosition());
}

void zeroY() {
```

```
	moveMotorsTogether(stepperX.currentPosition(), stepperY.currentPosition(),
			stepperX.currentPosition(), (17L * Y_MICROSTEPS_PER_INCH));
}

void checkStatus() {
	if(toPrint) {
		currentX = stepperX.currentPosition();
		currentY = stepperY.currentPosition();
	}
}
////////////////////////////////////////////////////////////////////////////////


// HEATING FUNCTIONS ///////////////////////////////////////////////////////////

//Baking control algorithm
void bakingProfile(int timeInc) {
	int counter = 0;
	int currTemp = 0;
	int tempInc = 1;	//(degrees C)/(timeInc)
	int prevTemp = 0;
	int prevTime = 0;
	double ovenTemp = 0;

	for (int i = 0; i < tempPoints; i++) {
		tempInc = (tempProfile[i] - prevTemp)/(timeArray[i] - prevTime)*timeInc;
		while(counter < timeArray[i]) {
			ovenTemp = readTemp();
			if (ovenTemp > MaxOvenTemp) {
				break;
			}

			if (currTemp < MaxTempLevel && currTemp > 0) {
				currTemp += tempInc;
			} else if (currTemp >= MaxTempLevel && tempInc <= 0){
				currTemp += tempInc;
			} else if (currTemp <= 0 && tempInc >= 0) {
				currTemp += tempInc;
			}

			if (currTemp > MaxTempLevel) {
				currTemp = MaxTempLevel;
			} else if (currTemp < 0) {
				currTemp = 0;
			}

			analogWrite(PWMpin,currTemp);
			counter++;
			delay(1000*timeInc);
		}
		prevTemp = tempProfile[i];
		prevTime = timeArray[i];
	}
}
```

```
//Thermocouple measurement function
double readTemp() {
	// basic readout temperature readout
	double c = thermocouple.readCelsius();
	if (isnan(c)) {
		Serial.println("Something wrong with thermocouple!");
		c = 0;
	} else {
		Serial.print("Oven Temperature (C) = ");
		Serial.println(c);
	}
	return c;
}

//Fan control function
void ovenFan(int on_off) {
	digitalWrite(fanPin,on_off);
}
```

Appendix H: Arduino Mega

Basic Mega Specs :
Microcontroller : ATmega2560
Input Voltage (recommended) : 7-12 V
Input Voltage (max) : 6-20 V
Digital I/O Pins : 54 with 15 PWM
Analog Input Pins : 16
External Interrupts : 6
DC Current per I/O Pin : 40 mA
DC Current for 3.3V Pin : 50 mA
Pin Output Voltage : 5 V

Table 3: Interrupt Layout

Interrupt #	*Pin #*	*Action*	*Trigger*
0	2	Rising	Heating Lid Limit Switches
1	3	Falling	Photo Interrupter
2	21	Rising	Left Limit
3	20	Rising	Right Limit
4	19	Rising	Front Limit
5	18	Rising	Back Limit

Table 4: Pin Layout

Pin #	*Output/Input*	*Description*
2	Input	Stops the heater lid when limit switch is activated
3	Input	Determines if M&M has been dispensed or not
18	Input	Back Y-axis limit switch
19	Input	Front Y-axis limit switch
20	Input	Right X-axis limit switch
21	Input	Left X-axis limit switch
22	Output	X motor direction pin
23	Output	Dispenser motor direction pin
24	Output	Dispenser motor step pin
25	Output	X motor step pin
26	Output	Y motor direction pin
27	Output	Heat motor step pin
28	Output	Heat motor direction pin
29	Output	Y motor step pin
38	Output	Heater fan
41	Input	Dispenser limit switch

Appendix I : Circuit

Shown below in Figure 38 is the entire circuitry required for our prototype to work. The

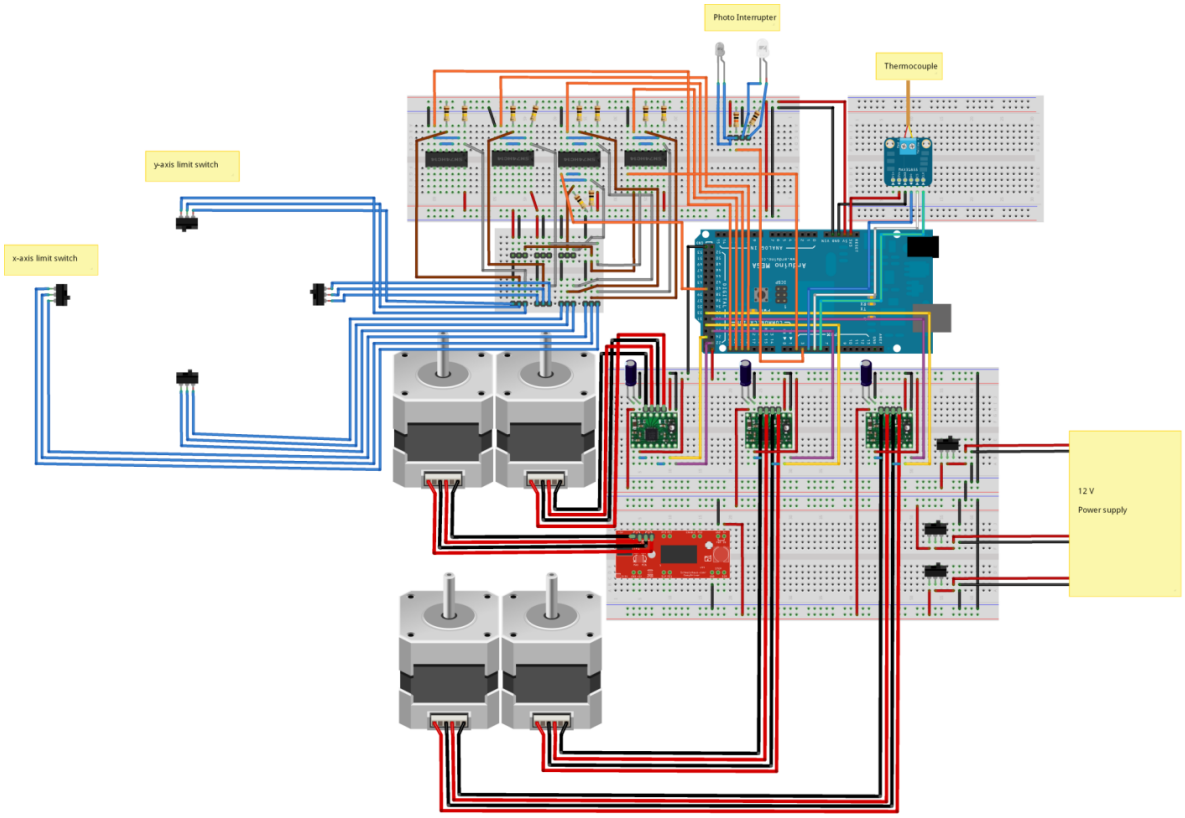


Figure 38: Circuitry of the Entire Control Panel for the Cookiebot

Debouncing Circuit

In the circuit shown above, a debouncing circuit is used in conjunction with the limit switches. The debouncing circuit was chosen to be used in conjunction as the limit switches are very sensitive to being pressed. When the limit switch is pressed, it produces an undesirable phenomenon called being ‘bounced’, which means that the output signal indicates that the switch is being rapidly switched on and off.

The debouncing circuit included in our electrical setup incorporates Transistor-Transistor-Logic (TTL) Quad open-collector NAND gates in order to increase circuit simplicity and accuracy.

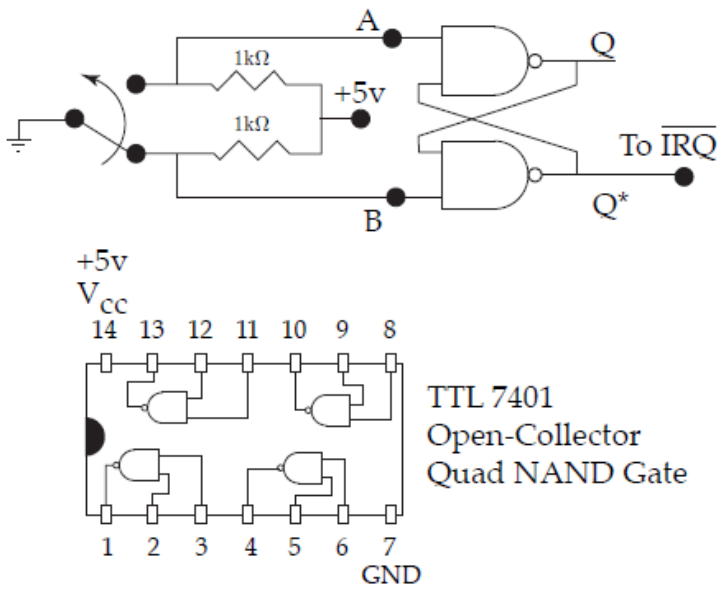


Figure 39: Debouncing Circuit

Stepper Motor Drivers

Figure 40 and Figure 41 are the breakout boards of the two types of stepper motor drivers that were used for controlling the four stepper motors that are shown in the Figure 38. The input into these circuits is the number of the steps sent by the Arduino. The stepper driver is responsible for sequentially activating the phases of the stepper motors in order to drive them.

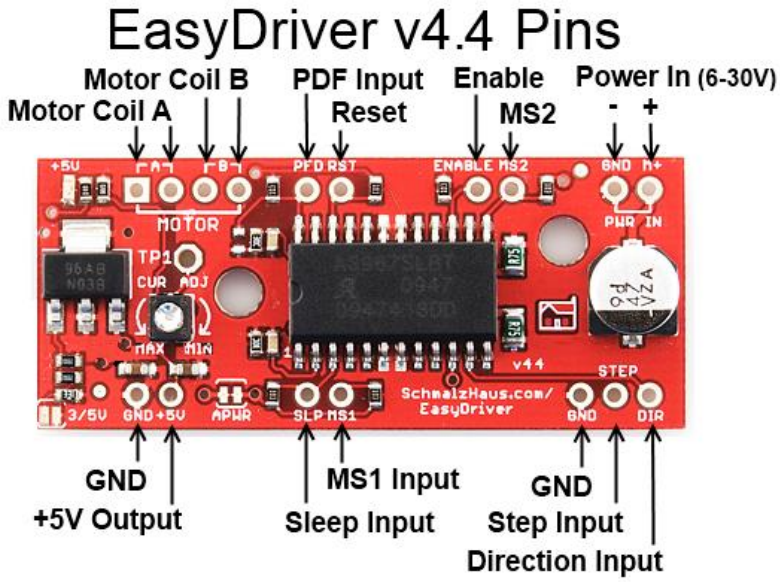


Figure 40: EasyDriver

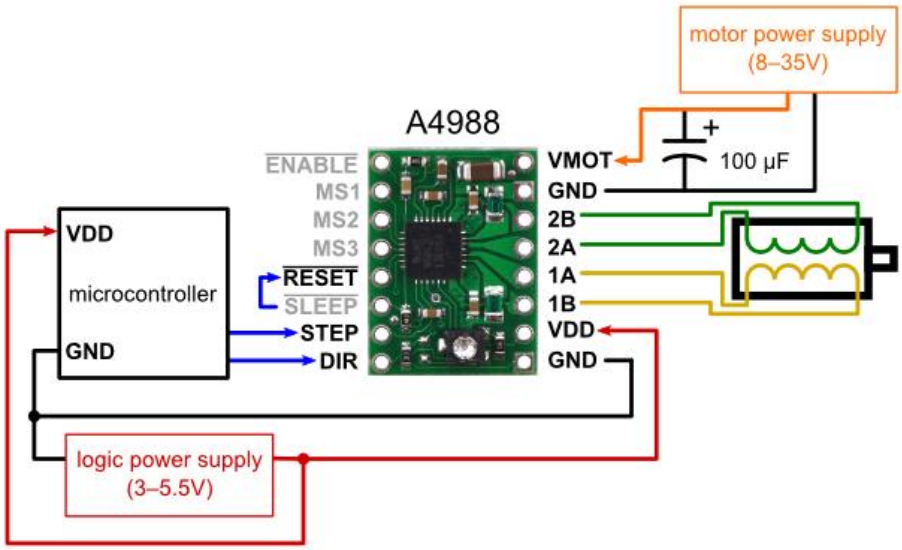


Figure 41: A4988 Stepper Motor Driver

Table 5: Step Resolution Pin Setup

MS1	*MS2*	*MS3*	*Microstep Resolution*
Low	Low	Low	Full Step
High	Low	Low	Half Step
Low	High	Low	Quarter Step
High	High	Low	Eighth Step
High	High	High	Sixteenth Step

Thermocouple

The circuit that allows the microcontroller to read the temperature for the K-Type thermocouple is shown at the top left corner of Figure 38. The input into this circuit is the analog signal generated by the thermocouple wires, which is then sent to the MAX31855 Cold-Junction Compensated Thermocouple-to-Digital Converter breakout board. This IC board processes the analog signal from the thermocouple and converts these voltage readings to the temperatures in degree Celsius. The output of this circuit is then read by the Arduino, which can either be used in monitoring the temperature or as the feedback for the PID controller.

Photo Interrupter

The photo Interrupter has four lead pins. It is made up of two components, an IR LED and a phototransistor. The LED continuously transmits IR light through a small slit within the sensor. The phototransistor reads this IR light which completes the circuit to allow current to flow through. One of the leads to the LED is connected to +5 V through a resistor. The other lead is

connected to ground. One lead of the phototransistor is connected to +5 v while the other is connected to ground through a resistor as well as pin 3 on the Arduino.

Appendix J : Cookie Recipe

The Nestle Toll House original chocolate chip cookie recipe (Nestle Toll House, 2001) was used to make final test cookies for baking. The recipe as follows was doubled and the chocolate chips were omitted in order to facilitate M&Ms to be printed on and stick to the cookies .

Original NESTLÉ® TOLL HOUSE® Chocolate Chip Cookies

Prep: 15 mins
Cooking: 9 mins
Cooling: 15 mins
Yields: 60
This famous classic American cookie is a treat no matter what the age or occasion. Enjoy it with a glass of cold milk.

Ingredients

- 2 1/4 cups all-purpose flour
- 1 teaspoon baking soda
- 1 teaspoon salt
- 1 cup (2 sticks) butter, softened
- 3/4 cup granulated sugar
- 3/4 cup packed brown sugar
- 1 teaspoon vanilla extract
- 2 large eggs
- 2 cups (12-oz. pkg.) NESTLÉ® TOLL HOUSE® Semi-Sweet Chocolate Morsels
- 1 cup chopped nuts

Directions

PREHEAT oven to 375° F.

COMBINE flour, baking soda and salt in small bowl. Beat butter, granulated sugar, brown sugar and vanilla extract in large mixer bowl until creamy. Add eggs, one at a time, beating well after each addition. Gradually beat in flour mixture. Stir in morsels and nuts. Drop by rounded tablespoon onto ungreased baking sheets.

BAKE for 9 to 11 minutes or until golden brown. Cool on baking sheets for 2 minutes; remove to wire racks to cool completely.

PAN COOKIE VARIATION: Grease 15 x 10-inch jelly-roll pan. Prepare dough as above. Spread into prepared pan. Bake for 20 to 25 minutes or until golden brown. Cool in pan on wire

rack. Makes 4 dozen bars.

SLICE AND BAKE COOKIE VARIATION:
PREPARE dough as above. Divide in half; wrap in waxed paper. Refrigerate for 1 hour or until firm. Shape each half into 15-inch log; wrap in wax paper. Refrigerate for 30 minutes.* Preheat oven to 375° F. Cut into 1/2-inch-thick slices; place on ungreased baking sheets. Bake for 8 to 10 minutes or until golden brown. Cool on baking sheets for 2 minutes; remove to wire racks to cool completely. Makes about 5 dozen cookies.

* May be stored in refrigerator for up to 1 week or in freezer for up to 8 weeks.

FOR HIGH ALTITUDE BAKING (5,200 feet): Increase flour to 2 1/2 cups. Add 2 teaspoons water with flour and reduce both granulated sugar and brown sugar to 2/3 cup *each*. Bake drop cookies for 8 to 10 minutes and pan cookie for 17 to 19 minutes.

Appendix K: Additional Cookie Baking Test Data

The data given in Table 6 was obtained from the baking tests conducted with the final baking enclosure implemented. The observations were made by visually inspecting the top and bottom of the cookies produced and by examining their insides. The convection power was 12 VDC.

Table 6: Cookie baking testing data

Run #	PWM Duty Ratio Control Points [%]	Time Control Points [Seconds]	Observations of Resulting Baked Cookie
1	[68.63, 100, 100]	[1, 120, 300]	Middle uncooked: Slightly visible from top, only slightly still raw on the bottom
2	[70.59, 100, 100]	[1, 270, 360]	Thoroughly cooked: Slightly overcooked on the edges, middle completely cooked
3	[72.55, 100, 100]	[1, 300, 360]	Overcooked: thoroughly cooked, but top of cookie is too dark
4	[70.59, 100, 100]	[1, 300, 330]	Cooked thoroughly: Cooked thoroughly, center is perfect brown, sides are still slightly too dark
5	[68.63, 90.20, 100]	[1, 330, 360]	Overcooked: Thoroughly cooked, but top of cookie is too dark
6	[68.63, 86.27, 86.27]	[1, 280, 310]	Thoroughly cooked: A little overcooked on the edges
7	[62.75, 94.12, 94.12]	[1, 320, 350]	Thoroughly cooked: Perfectly browned over entire top region, and through in all areas