Enhancing Culinary Precision: Students Embarking on a Project-Based Learning Adventure

Simon Zhang, Northeastern University
Joshua Dennis, Northeastern University
Dr. Haridas Kumarakuru, Northeastern University

Haridas Kumarakuru, PhD, MInstP Department of Physics, College of Science, Northeastern University, 360 Huntington Ave, Boston, MA 02115 E.Mail: h.kumarakuru@northeastern.edu

Hari has 18+ years of educational leadership experience amplifying academic and scientific endeavours in the higher education setting that has brought him to four separate continents. He capitalizes on his in-depth competencies in curriculum implementation, instructional delivery, scientific research, technical writing, and student mentoring to provide students with the tools for academic and professional success. Since 2007, he has had the privilege of mentoring numerous undergraduate and master's students, a pursuit he is most passionate about. He has applied his established teaching skills to a wide range of undergraduate courses in general physics, engineering physics, electronics for scientists, advanced physics labs and specialized courses in the fields of functional nano material science and nanotechnology. Hari is a member of IOP (UK), JSA, AAPT and ASEE and he is a reviewer for several scientific journals.

Dr. Bala Maheswaran, Northeastern University

Bala Maheswaran, PhD Northeastern University 367 Snell Engineering Center Boston, MA 02115

Enhancing Culinary Precision:
Students Embarking on a Project-Based Learning (PBL) Adventure

Simon Zhang[1], James Lewis[1], Krish Gupta[1], Jeje Dennis[1], Ajith George[1], Haridas Kumarakuru[2], and Bala Maheswaran[1]

College of Engineering[1]
Department of Physics[2]
Northeastern University

Abstract

In the dominion of Project-Based Learning (PBL), we embarked on a journey to create an innovative time prediction thermometer tailored for food systems. Throughout this endeavor, we explored a range of fundamental principles that have proven invaluable for our lifelong learning journey. As students, this project provided fertile ground for honing our problem-solving skills and immersing ourselves in the intricate world of engineering design.

Our journey began with identifying a culinary challenge, followed by brainstorming potential solutions, selecting an efficient approach, and executing it meticulously. A project-based learning (PBL) curriculum was highly compatible with the goals of our thermometer development. A range of technical skills were introduced in the classrooms as tools available to tackle a broader problem, giving students a more flexible mindset than a traditional learning curriculum might have. In pursuit of projection execution, this endeavor equipped us with a wealth of practical skills, including fabrication, design, analysis, and the art of technical writing. It served as a platform for us to refine our expertise in computer-aided design (CAD), research methodologies, and the dynamics of collaborative teamwork. The main implication to be gained from a PBL methodology is a set of multi-faceted skills, not limited to technical expertise but also maintenance of project timelines and collaborative cohesiveness, applicable to future real-world engineering problems.

Without temperature prediction models, cooks tend to rely on inaccurate temperature gauging heuristics, often resulting in suboptimal culinary outcomes that are either undercooked or overcooked. Our team made a deliberate choice to construct a temperature probe that would mitigate this issue by precisely measuring the internal temperature of food and forecasting its future temperature, which we have aptly named a "smart thermometer." In this paper, we elucidate the design criteria for ThermoChef++ (TC++), a budget conscious smart thermometer, in comparison to existing time prediction models in literature and smart thermometers available in the consumer market. We elucidate the development of software and Arduino circuits in the realization of our project.

During the culinary process, TC++ continuously monitors real-time temperature data and employs regression analysis to construct thermal models that predict the future behavior of the food system. An additional feature of TC++ is a library containing standard cooking models for common foods, promoting heightened awareness of cooking safety. The current iteration of our product is tailored for home cooks. However, an enhanced iteration of TC++ holds potential implications for the food manufacturing industry.

Introduction & Background

When it comes to cooking, novice chefs can come into the problem of undercooking or overcooking the food. This can be dangerous because undercooking food heightens the risk of foodborne illnesses. Untrained cooks can look to using a thermometer as a tool to help with their cooking, but knowledge of the additional subtleties around recipe optimization, such as combining ingredients that finish cooking around the same time, only develops with experience and intuition. There are smart thermometers on the market such as Yummly, Meater, and more but the price of these products ranges from $80-$370. TC++ aims to deliver reliable results for a lower price.

There are many tools used to measure temperature, one of the tools is the thermocouple. In 1821, Thomas Johann Seebeck was able to use his discovery of the Seebeck effect to create a thermocouple. The Seebeck effect is the thermoelectric phenomenon where when a junction is heated between two dissimilar materials it creates a change in the electrical operation of the integrated circuit [1]. A thermocouple uses this idea where it has two different electrical conductors that form an electrical junction that produces a temperature-dependent voltage to be interpreted to measure temperature.

Recently scientists have been working on methods to mathematically model of cooking different foods. For example, one group investigated how using a nonuniform heating source cooks pancakes. Their proposed model incorporated the transfer of heat to the product using this equation:

$$\lambda p \nabla^2 T = \rho p Cp \frac{\partial T}{\partial t} + Q_{evp}$$

The proposed model also observed the transfer of mass using these two equations:

$$\rho_p \frac{\partial x_l}{\partial t} = \nabla \cdot (\rho_p D_l X_l) - R_{evp}$$
$$\rho_p \frac{\partial x_v}{\partial t} = \nabla \cdot (\rho_p D_v X_v) + R_{evp}$$

We note the high complexity of these models, which describe temperature as an aggregate of multivariable factors. The group validated their model as successful and could be used for predicting the cooking result when using a given domestic cooker with a non-uniform heating in the food. We explored the practicalities of implementing such complex systems in the real world with our project.

Methods & Approach

In a preliminary brainstorm, we cemented the objectives that all design choices for TC++ would follow: TC++ must be cheap to reproduce and available for public modification. Smart thermometers in the consumer market at the time of project conception cost at least $100. Because of the high fixed costs of smart thermometer usage, there is opportunity for TC++ to advance the market's low-end sector. The outcome of our budgeting objective was the choice to use a SparkFun Redboard (Arduino Uno derivative) in data collection. We uploaded all code to GitHub to enable easy access to the project files necessary for replicating TC++. A secondary benefit to GitHub was that it enabled project development on multiple code bases and provided a place of code consolidation. GitHub allowed us to move from a planning phase to a testing phase quickly, boosting efficiency in the design cycle.

A review of the scientific literature regarding heat transfer in cooking systems was insightful to understanding the dynamics of the cooking process. In evaluating the accuracy of our thermometer's

temperature prediction models, replication of or comparison to proven thermal models was essential to the project's success. Another consequence of reviewing literature was simply our familiarization with a unique application of mathematics that may guide our future interests in product engineering and design.

Previous demonstrations in literature that model internal temperature of food systems often described temperature via a variation of the heat equation [2].

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial z^2} = \frac{\rho c_p}{k} \cdot \frac{\partial T}{\partial x}$$

.

While we anticipate that solving the heat equation for time would increase accuracy of TC++'s time prediction, the cost of computing power and multiple thermocouples, used by solutions generated by finite difference analysis, could potentially cause TC++ to be as expensive as current consumer smart thermometers.

Our conclusion was to develop the time prediction via regression analysis performed on real time thermocouple data sent to an Arduino Uno. The tradeoff of accuracy is especially notable during the initial stages of cooking, but we anticipated that the error between predicted times and real-world times would lower substantially as the system progresses in time; we evaluated the truth of this premise in experiment, as per the test phase of the engineering design process.

Once we established the outline for TC++'s prediction model, we approached the task of physical construction. The project was primarily software oriented. The safety concern of developing a product to be used in the kitchen emphasized careful material choice in that building a thermometer out of materials prone to melting would not only cause the project to break but also create toxic, inedible food. We subverted concerns over material stability by placing all electronics inside a box that would be separate from a heating source, such as a stove burner or oven. All physical interaction between a heating source and TC++ was limited to the probe used to measure temperature, which would naturally withstand elevated temperatures.

A lack of physical stress on our box translated into reduced construction costs as we could source cheaper parts or even retrofit existing boxes, conforming to our overarching design philosophy. The box's main purpose was for organization and cleanliness, so as not to expose bare wires and circuits in the kitchen. However, we emphasize that it has no impact on the functionality of TC++ and can be omitted should others create their own version of the project.

DESIGN DETAILS

In order to improve end-user experience and increase the accessibility of our project, we decided to create a publicly available Android application. Android, as a platform, was more suitable than IOS for our purposes due to the accessibility of development tools and physical testing on Android devices, which can install external applications unlike IOS. To create the app, we coded in Java using Android Studio, initially building the UI layout in XML (Figure 1-2).

Figure 1: App home page

Figure 2 – Graphing Output

The purpose of the app was to wrap informative data in an approachable interface. We also intended for the app to facilitate ease of usage between a user and TC++ by enabling all communication and data processing through the app. Our Bluetooth module of choice was the Bluetooth Low Energy 4.0 HM-10 module. As a BLE module, the HM-10 required implementation of a master-slave style communication protocol [3]. The built-in Bluetooth library in Android studio was difficult to work with and caused a lot of bugs which we had to work to fix. As it currently stands, the app has the complete UI framework to display data to the user but the technical aspects of BLE must be solved for full integration with TC++.

Figure 3: Initial design

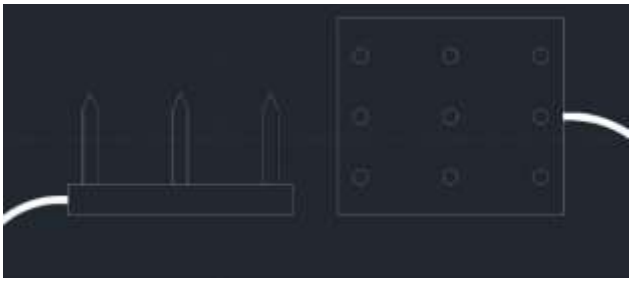


Figure 4: Alternative multi-prong design

Our initial design was designed in AutoCAD and includes the temperature probe connected with a wire to the electronics housing unit. The temperature probe displays the temperature that it is detecting. The housing unit displays the time that it will take to cook the food and the battery life remaining for the device. Not directly included in Figure 1 but included in the design is an app interface that will display the current temperature and the time until the food is cooked.

An alternative design considered was using a multi-prong array to increase temperature data accuracy. Because heating food induces a temperature gradient and uneven temperature rise throughout the food, tracking a single point in the system is less accurate than reading temperatures from multiple points [4]. However, there were several constraints that prevented us from building a prong array. Primarily, we did not possess the mathematical background nor the computing power to solve the differential equations

necessary to integrate the prong data into our modeling system. A tool like MATLAB would only be able to solve one dimensional boundary value partial differential equations, so we were not convinced that we had the necessary software to thoroughly model in real-time [5]. Regardless, we posit that a home chef would only be concerned with precision in the span of integers. Our alternative design was also substantially more expensive than our initial because it required nine separate thermocouples, as well as a transformer chip for each of them.

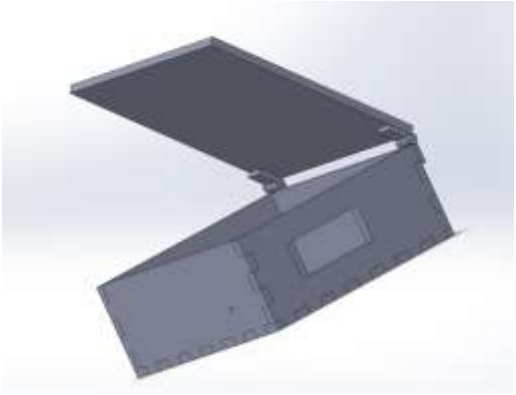


Figure 5: Final box CAD drawing

We wrote a program for our regression analysis model in Java, using the Apache Commons Library (version 3.6.1). We first set up the program to intake the numbers coming from the Arduino to be interpreted as data points of the independent variable (temperature). The Java program then fits a polynomial function by iteratively solving least squares problems, with the goal of minimizing the overall differences between the real data and trial data generated by fitted functions. The specific algorithm that describes the optimization process is the Levenberg–Marquardt algorithm [6]. We depict the resulting polynomial function using the Java Graphics library and generate time-temperature predictions by iteratively testing when the polynomial function evaluates to a desired temperature.

To create a box for our Arduino wiring, we decided to laser cut the main body of the unit because wood is more food safe for high temperatures and still easily attainable. The types of plastic used in 3D printing could potentially melt in the kitchen, with PLA known to lose rigidity at around 60°C [7]. Wood may char, but without a spark, will remain relatively safe in a kitchen even near high temperatures from an oven. We utilized an interlocking "puzzle-piece" technique to join the walls and floor of the box together. This involved cutting out interlocking edges along the panels that we cut using the laser cutter. All the larger pieces were cut out of ¼" plywood. The hole for the LCD screen was precisely measured so that the screen could be friction fit. The hinges were our only 3D printed component. They were printed initially as two separate pieces with a pin that was also 3d printed but the pin proved to be too flimsy. We replaced the pins with a small wooden dowel. We attached the hinges to their respective dowels with adhesives because nails or screws could crack the thin material. However, not every glue was suitable for our project. Hot glue proved to be too flimsy while PVA glue simply did not adhere properly; but wood glue was able to provide a rigid, secure hold.

Work on the smart thermometer required us to become familiar with a wide variety of programming languages and tools and CAD tools, making the project an interdisciplinary project on both a mechanical and electrical engineering level. While some skills such as AutoCad and SolidWorks were introduced in the curriculum, the PBL nature of our work gave us an opportunity to practice those skills outside of

assignments. The utilization of Android Studio and BLE were examples of skills learned outside of the standard curriculum that the PBL opportunity gave us.

RESULTS & DISCUSSION

In testing our time prediction model, we cooked a chicken drumstick in an oven set to 350°F. The thermocouple probe was placed in the center of the drumstick meat. Rather than an experiment, our testing phase more closely resembled an observational study, as we intended to evaluate TC++'s behavior in an entropic, natural environment where multiple uncontrolled factors influence the cooking of food. We aimed to gauge the general usefulness and adaptability of our project despite the high degree of randomness inherent in cooking systems. A non-exhaustive list of some uncontrolled factors is the instability of consumer oven heating; whether the cooking system is closed (wrapped in foil or closed with a lid) or open (pan frying); moisture loss and shape changes as cooking occurs.

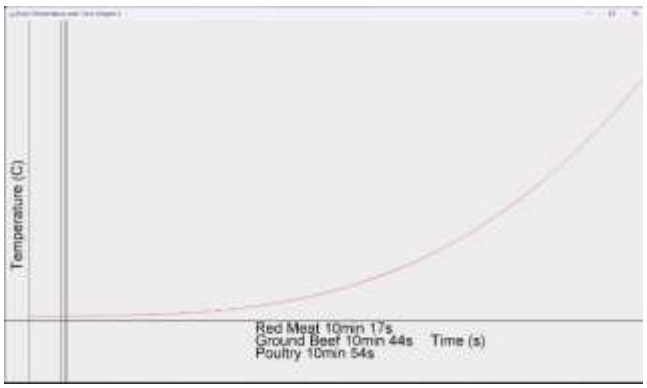


Figure 6.a.: Third degree fitting at 250 seconds

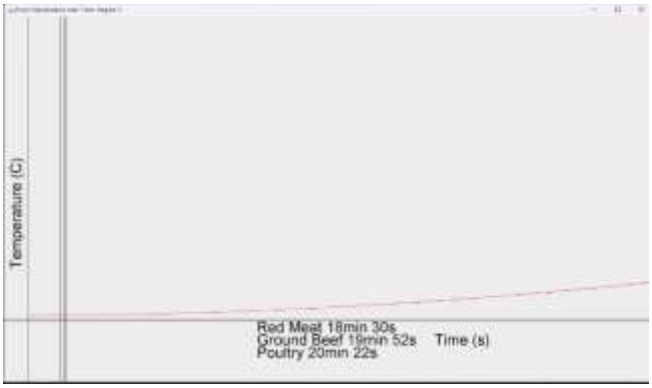


Figure 6.b.: Second degree fitting at 250 seconds

Comparing figure 6.a. and figure 6.b., the variation between projecting cooking times varied dramatically dpeending on the order of the polynomial fit used to predict cooking times. Given the relatively low sample of data points, and the inertia of the food's temperature at the start of cooking, accuracy of the final cooking times was limited.

Figure 6.c.: First degree fitting at 250 seconds

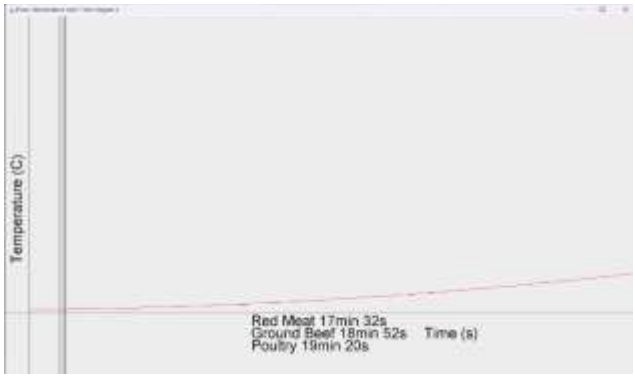


Figure 7.a.: Second degree fitting at 650 seconds

The collection of figures 6.a through 6.c display a general shape of temperature in Celsius as a function of time in seconds when four minutes and 10 seconds had elapsed since the start of cooking. We generated

the graphs using the Java Graphics library by third degree through first degree polynomial fitting of real time temperature data. We state that the total cooking time of the drumstick lasted 23 minutes. From there, Figure 6.c illustrates a nearly 270% error between predicted and actual cooking times. Foods often exhibit a kind of temperature inertia in the first stages of cooking where the rate of temperature change is much slower [8], [9]. This effect seems agnostic to cooking methods. According to our results, linear fitting poorly describes temperature evolution in the initial cooking phase. The data agrees with our assumption that linear functions are inappropriate in describing food temperature. Furthermore, we inference that the lack of variation in temperature data was caused by temperature inertia. The second and third degree curve fits predicted more accurate cooking times, with the second degree prediction having only a 10% error from the actual time. The higher order fits better described the more dynamic nature of temperature evolution that we expected in a cooking system.

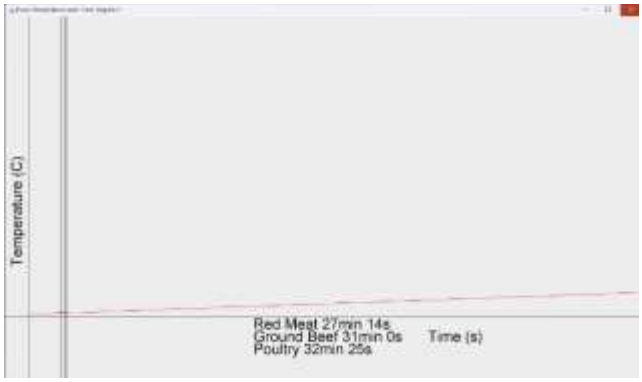


Figure 7.b.: First degree fitting at 650 seconds

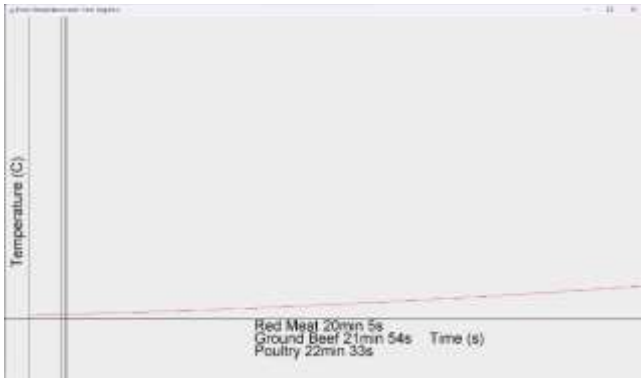


Figure 8: Second degree fitting at 1,000 seconds

Figures 7.a. and 7.b. are the graphs generated by TC++ six minutes after the graphs generated in Figures 6. In accordance with our hypothesis in the design phase, the linear model experienced a major correction in its predicted cooking time as it TC++ read more temperature data. Interestingly, just as the linear curve's predicted time had decreased, the second order curve also exhibited the same behavior to a lesser degree. Percent error increased to 16% for the second order fit, while it decreased to 41% for the linear fit. The overall reduction in predicted times may suggest some variability in the oven's temperature stability affecting the data.

By 16 minutes and 40 seconds, the percent error between observed and predicted time had stabilized to 2%. In a practical perspective, we observed a fluctuation of about three minutes in the total cooking time that TC++ predicted with the second order polynomial fit to live temperature data. We consider the error in the prediction of figure 6.b. small enough that users of TC++ will have a pragmatic estimate of when their food will finish cooking.

FUTURE WORK & IMPROVEMENTS

There are several improvements that could have been applied to our final design if we were not limited by time or money constraints. As mentioned earlier in the paper, we could have implemented a multi prong design for increased temperature precision. The addition of audiovisual alerts would be beneficial for distracted cooks or consumers with disabilities. Additional features could include a reference library of standard cooking models for commonly cooked foods or saving of previous cooking data which could be viewed by the user.

Bluetooth success was middling, as the end product was not able to communicate with the Android app successfully. We planned for the thermometer hardware to stream the temperature that it read with corresponding elapsed time since the start of usage. Upon receiving the data stream, the app would superimpose projected cooking times on a graph (see Figure 2). The user would be able to see a visual representation of the food's current and future temperatures.

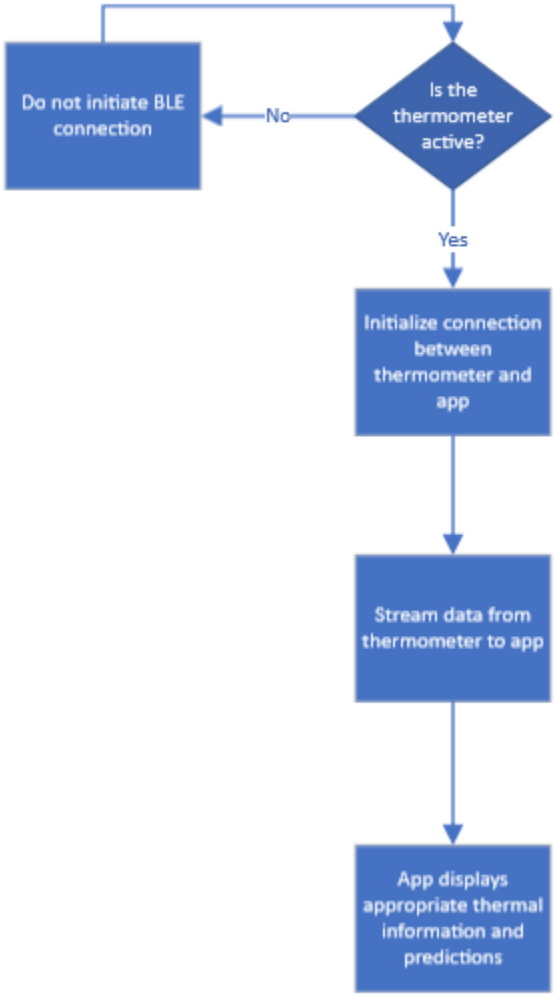


Figure 9: Flowchart of BLE and app

To make for a simple UI we reduced the number of menus and buttons to a minimum and made all the vital information easily accessible. Overall, the app interface was largely successful, but the problems with Bluetooth cause the app to tangentially integrate with TC++ at the moment.

As the communication path between TC++ hardware and software is unidirectional, the Bluetooth functionality could be forgone using I2C instead. This would allow TC++ to work without a BLE module and app; visual representations could potentially be shown on LCD displays that would communicate with the Arduino's built in I2C capabilities.

CONCLUSION

The outcome of the PBL experience was considered successful, as a working prototype of our intentions was catalyzed. The first version of TC++ implements the core functionalities needed to assist cooks. Our main goal of a replicable, cheap smart thermometer undercuts the most affordable consumer smart thermometer by $10 in our real experience but expands to a theoretical $20; there is some flexibility in constructing TC++, especially with the electronics box and electric component sourcing.

Although TC++ has the ability to perform regression analysis with polynomials of any order, our testing shows that polynomial fits of the second order most accurately describe food internal temperature as a function of time. Our data also suggests that regression analysis techniques adapt to most foods and cooking conditions, though the amount of time before predictions stabilize may vary depending on situation.

A possible consideration for the wider implications of TC++ is some contribution to industry. Knowledge of how their products may help food manufacturers optimize their factory processes. While industry usage would require a great amount of precision that TC++ currently offers, there is a real consideration in using a project like ours for systems prototyping and testing within the field of industrial engineering or logistics. Another potential for industry is to not necessarily use our project as it currently stands but to integrate the foundational idea of real time regression analysis in the context of more elaborate food quality models. The ideas set forth by TC++ can be used in conjunction with a complex, multivariate temperature system, which itself will be part of a greater scheme of models that help manufacturers quantitatively measure their products.

The style of our PBL experience was generally unfettered that the group determined their own goals and areas of focus for the project. The timeline of the project was confined to the duration of course with interspersed work demonstrations to motivate continual progress with the project; we recommend that any PBL curriculum has checkpoints for student project demonstrations.

An advantage of the creative freedom in our PBL experience was that students were allowed to pick a project that they personally wanted to see to fruition; we believe that this creative freedom is helpful to facilitate student motivation and interests. However, we recognize that different project goals will have a range of difficulties, and supervision of project goals may be necessary to keep projects manageable within the scope of the course time.

ACKNOWLEDGMENTS

We would like to acknowledge all of the Northeastern faculty and staff that lent us their time and materials to make our project possible. We especially want to thank FYELIC and their workers in this regard.

REFERENCES

[1] D. Beretta *et al.*, "Thermoelectrics: From history, a window to the future," *Materials Science and Engineering: R: Reports*, vol. 138, p. 100501, Oct. 2019, doi: 10.1016/j.mser.2018.09.001.

[2] K. Siripon, A. Tansakul, and G. S. Mittal, "Heat transfer modeling of chicken cooking in hot water," *Food Research International*, vol. 40, no. 7, pp. 923–930, Aug. 2007, doi: 10.1016/j.foodres.2007.03.005.

[3] "Intro to Bluetooth Generic Attribute Profile (GATT)," *Bluetooth® Technology Website*, Mar. 26, 2017. https://www.bluetooth.com/bluetooth-resources/intro-to-bluetooth-gap-gatt/ (accessed Apr. 20, 2023).

[4] R. Rocca-Poliméni, N. Zárate Vilet, S. Roux, J.-L. Bailleul, and B. Broyart, "Continuous measurement of contact heat flux during minced meat grilling," *Journal of Food Engineering*, vol. 242, pp. 163–171, Feb. 2019, doi: 10.1016/j.jfoodeng.2018.08.032.

[5] "Solving Partial Differential Equations - MATLAB & Simulink," *Solving Partial Differential Equations*. https://www.mathworks.com/help/matlab/math/partial-differential-equations.html (accessed Apr. 20, 2023).

[6] "AbstractCurveFitter (Apache Commons Math 3.6.1 API)," *Class AbstractCurveFitter*. https://commons.apache.org/proper/commons-math/javadocs/api-3.6.1/org/apache/commons/math3/fitting/AbstractCurveFitter.html#fit(java.util.Collection) (accessed Apr. 20, 2023).

[7] "Thermal analysis of polylactic acid".

[8] J. Moya *et al.*, "Development and validation of a computational model for steak double-sided pan cooking," *Journal of Food Engineering*, vol. 298, p. 110498, Jun. 2021, doi: 10.1016/j.jfoodeng.2021.110498.

[9] E. Purlis and V. O. Salvadori, "Bread baking as a moving boundary problem. Part 1: Mathematical modelling," *Journal of Food Engineering*, vol. 91, no. 3, pp. 428–433, Apr. 2009, doi: 10.1016/j.jfoodeng.2008.09.037.

APPENDIX

Java Class to Receive Arduino Temperature Data

```
package com.tc;

import com.fazecast.jSerialComm.*;
import java.io.InputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.lang.String;

public class ArduinoConnect {
    private static SerialPort comport;
    private static InputStream data;
    public static ArrayList<Integer> x;
    public static ArrayList<Integer> y;
    public static boolean read = true;

    public static void readArduino() throws IOException, InterruptedException {
        x = new ArrayList<>();
        y = new ArrayList<>();
        SerialPort[] allAvailableComPorts = SerialPort.getCommPorts();



        // Lists all available serial and connects to the first one
        // Prerequisite to function: Arduino Uno is the only free Serial connection
        for (SerialPort eachComPort : allAvailableComPorts)
            System.out.println("List of all available serial ports: " +
eachComPort.getSystemPortName());
        comport = SerialPort.getCommPort(allAvailableComPorts[0].getSystemPortName());
        // No timeout
        // 9600 for Serial and expect 8 bits
        comport.setComPortTimeouts(0, 0, 0);
        comport.setComPortParameters(9600, 8, 1, 0);
        if (comport.openPort()) {
            System.out.println("Port open");
            out:
            while(read) {
                data = comport.getInputStream();
                if (data.available() > 0) {

                    // Creates a byte array to store all bytes in the Serial connection

                    byte[] serialBuffer = new byte[data.available()];
                    // Assigns bytes from Serial connection to byte array
                    data.readNBytes(serialBuffer, 0, data.available());
                    if(serialBuffer.length>2){
                    if(serialBuffer[1]==28 && serialBuffer[2] == 32)
                    {
```

```
                        System.out.println("Two hour limit reached");
                        break out;
                    }
                    // Graph if receive 10,000 from Arduino
                    if(serialBuffer[1]==39 && serialBuffer[2] == 16)
                    {
                        System.out.println("Generating graph");
                        Graph graph1 = new Graph(x,y,1);
                        Graph graph2 = new Graph(x,y,2);
                        Graph graph3 = new Graph(x,y,3);
                        graph1.drawGraph();
                        graph2.drawGraph();
                        graph3.drawGraph();
                    }
                    }
                    // Correct time bytes once they are over 256
                    if(serialBuffer.length==3)
                    {
                        int combinedByte = (Byte.toUnsignedInt(serialBuffer[1])<<8) |
Byte.toUnsignedInt(serialBuffer[2]);
                        System.out.println("Time: " + combinedByte);
                        System.out.println(serialBuffer[0] + " C");
                        x.add(combinedByte);
                        y.add(Byte.toUnsignedInt(serialBuffer[0]));
                    }
                    else
                    {
                        System.out.println(("Time: " +
Byte.toUnsignedInt(serialBuffer[1])));
                        System.out.println((serialBuffer[0] + " C"));

                        x.add(Byte.toUnsignedInt(serialBuffer[1]));
                        y.add(Byte.toUnsignedInt(serialBuffer[0]));
                    }
                }
                // Timeout enables proper reading
                Thread.sleep(100);
            }
        }
    }
}
```

Java Class to Create Temperature Evolution Graph

```java
package com.tc;

import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;
import org.apache.commons.math3.fitting.PolynomialCurveFitter;
import org.apache.commons.math3.fitting.WeightedObservedPoints;
import java.util.ArrayList;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Graph
{
        private final int X_AXIS = 900;
        private final int Y_AXIS = 50;
        private ArrayList<Integer> x;
        private ArrayList<Integer> y;
        PolynomialFunction tempFun;
        private int xSize;
        private int ySize;
        private int degree;
    public Graph(ArrayList<Integer> x, ArrayList<Integer> y, int degree)
    {
        PolynomialCurveFitter polyboy= PolynomialCurveFitter.create(degree);
        WeightedObservedPoints data = new WeightedObservedPoints();
        this.x = x;
        this.y = y;
        this.degree = degree;
        for(int i = 0; i<x.size(); i++)
        {
            data.add(x.get(i),Integer.valueOf(y.get(i)));
            // System.out.println(x.get(i) + " " + y.get(i));
        }
        // Polynomial fit function created
        double[] coef = polyboy.fit(data.toList());
        tempFun = new PolynomialFunction(coef);
        // Drawing the graph
        xSize = x.size();
        ySize = y.size();
    }
        public void drawGraph()
        {
            JFrame frame = new JFrame("Food Temperature over Time: Degree " + degree);
        frame.setSize(2500,2500);
        frame.setVisible(true);
```

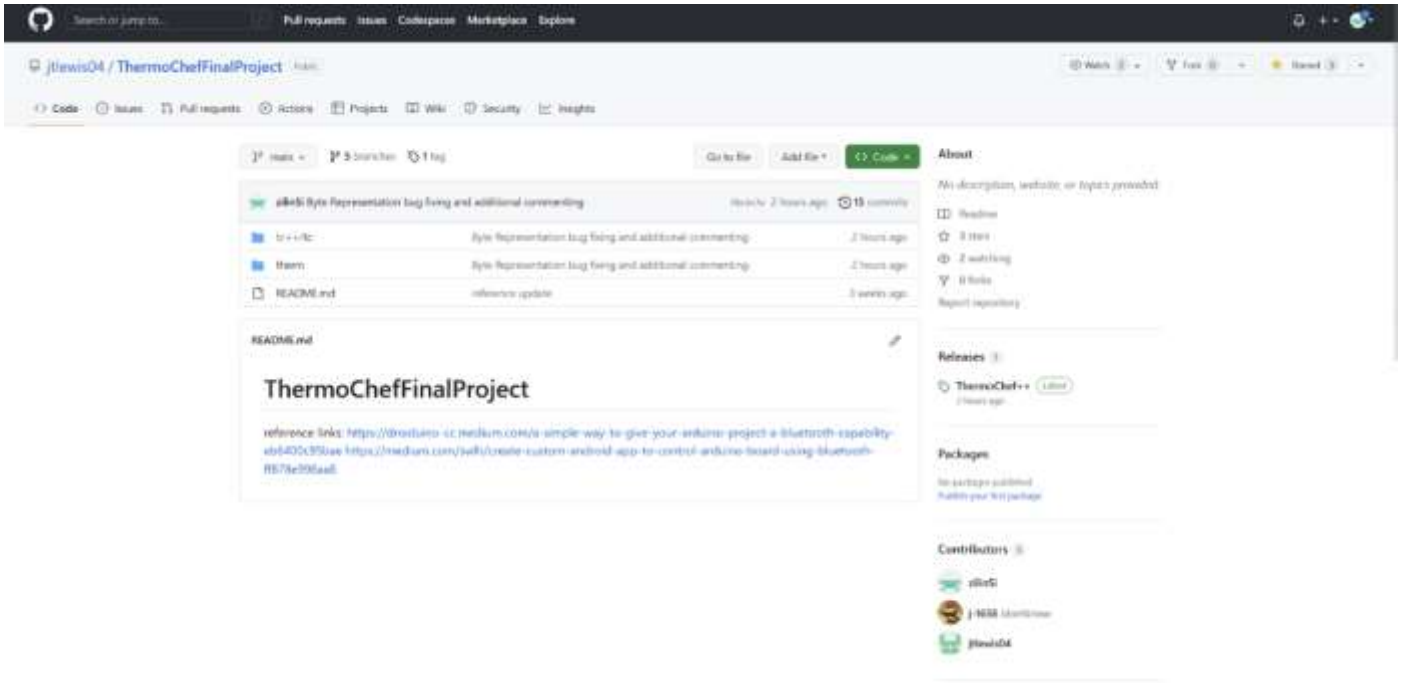
```
        JPanel panel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g)
            {
                super.paintComponent(g);
                Graphics2D g2d = (Graphics2D)(g);
                g2d.setColor(Color.BLACK);
                Font font = new Font(Font.DIALOG, Font.PLAIN, 30);
                g2d.setFont(font);
                g2d.drawString("Time (s)", 850, 650);
                g2d.translate(35,500);
                g2d.rotate(-Math.PI/2);
                g2d.drawString("Temperature (C)", 0, 0);
                g2d.rotate(Math.PI/2);
                g2d.translate(-35,-800);
                g2d.drawLine(0,X_AXIS,1800,X_AXIS);
                g2d.drawLine(Y_AXIS,0,Y_AXIS,3400);
                g2d.setColor(Color.RED);
                boolean gb = true;
                boolean cb = true;
        for (int x = 0; x < 7200; x++) {
            g2d.drawLine(x+Y_AXIS, X_AXIS-(int) tempFun.value(x), x + 1+Y_AXIS,
X_AXIS-(int) tempFun.value(x + 1));
        }
        g2d.setColor(Color.BLACK);
        for(int i = 0; i<10000; i++)
        {
            if((int)tempFun.value(i) == 63 && cb &&
(int)tempFun.value(i)!=Integer.MAX_VALUE && (int)tempFun.value(i)!=Integer.MIN_VALUE)
            {
                g2d.drawLine((int)(Y_AXIS +
tempFun.value(i)),0,(int)(Y_AXIS+tempFun.value(i)),3400);
                if(i<60)
                {
                    g2d.drawString("Red Meat "+i+"s",500,X_AXIS + 25);
                }
                else{
                    g2d.drawString("Red Meat "+i/60+"min " + i%60+"s",500,X_AXIS + 25);
                }
                System.out.println("Time: " + i + " " + "Predicted Temp: " +
(int)tempFun.value(i));
                cb = false;
            }
            if((int)tempFun.value(i) == 71 && gb &&
(int)tempFun.value(i)!=Integer.MAX_VALUE && (int)tempFun.value(i)!=Integer.MIN_VALUE)
            {
                g2d.drawLine((int)(Y_AXIS +
tempFun.value(i)),0,(int)(Y_AXIS+tempFun.value(i)),3400);
                if(i<60)
```

```
                {
                    g2d.drawString("Ground Beef "+i+"s",500,X_AXIS + 50);
                }
                else{
                    g2d.drawString("Ground Beef "+i/60+"min " + i%60+"s",500,X_AXIS +
50);
                }
                System.out.println("Time: " + i + " " + "Predicted Temp: " +
(int)tempFun.value(i));
                gb =false;
            }
            if((int)tempFun.value(i) == 74 && (int)tempFun.value(i)!=Integer.MAX_VALUE &&
(int)tempFun.value(i)!=Integer.MIN_VALUE)
            {
                if(i<60)
                {
                    g2d.drawString("Poultry "+i+"s",500,X_AXIS + 75);
                }
                else{
                    g2d.drawString("Poultry "+i/60+"min " + i%60+"s",500,X_AXIS + 75);
                }
                System.out.println("Time: " + i + " " + "Predicted Temp: " +
(int)tempFun.value(i));
                g2d.drawLine((int)(Y_AXIS +
tempFun.value(i)),0,(int)(Y_AXIS+tempFun.value(i)),3400);
                break;
            }
        }
            }
        };
        panel.setSize(2500, 2500);
        panel.setVisible(true);
        frame.add(panel);

        }

}
```

GitHub Page



NOMENCLATURE

λ_p = Thermal conductivity of the product
∇^2 = Laplacian of the temperature
T = Internal temperature of food
P_p = density of the product
C_p = Specific heat capacity
t = Elapsed time
Q_{evp} = Heat sink corresponding to water evaporation
x_l = Mass fraction of liquid
x_v = Mass fraction of vapor
∇ = Gradient
D_l = Diffusion coefficient
D_v = Diffusion coefficients
R_{evp} = rate of water evaporations
x: Space along the x-axis
z: Space along the z-axis
k: Thermal conductivity constant