Bondline temperature control using carbon fiber embedded resistive heaters

Brandon Phillip Smith

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2017

Reading Committee:

Santosh Devasia, Chair

Mark Tuttle, Chair

Joseph Garbini

Program Authorized to Offer Degree:
Mechanical Engineering

University of Washington

Abstract

Bondline temperature control using carbon fiber
embedded resistive heaters

Brandon Phillip Smith

Co-Chairs of the Supervisory Committee:
Professor Santosh Devasia
Mechanical Engineering

Professor Mark Tuttle
Mechanical Engineering

This research investigates the limits of temperature control and estimation when using embedded resistive heaters for adhesive bonding of carbon fiber-reinforced polymer (CFRP) adherends. With the increasing use of CFRP materials, high quality adhesive bonding is important for reducing costs of manufacturing, and improving repair methods. By using carbon fiber heaters to elevate the temperature and cure adhesive within a bond, the process becomes targeted and highly efficient. To accomplish accurate temperature control without placing thermocouples within the bondline, this research develops a method by which to estimate bondline temperatures using surface measurements. Additionally, it is shown that boundary control on an embedded heater can be used to control heat within a heater. This can be used to compensate for variations in heat flow in the bonded adherends to maintain a uniform temperature. This is demonstrated on a single lap joint with a subsurface heatsink. Finally, the use of carbon fiber embedded heaters will be investigated as a means for manufacturing of space structures and habitats.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to all of his colleagues, friends, and advisers who have supported and helped him through this process. He is especially grateful for his wife Mim for her support and patience throughout the process. Additionally, he would like to acknowledge the financial support he has received from the Boeing Company and the National Science Foundation.

Chapter 1

INTRODUCTION

CFRP materials have a number of advantages over traditional materials, such as aluminum or steel. These include a superior strength-to-weight ratio, resistance to corrosion and fatigue, and the ability to design the direction of strength by varying fiber direction. For these reasons, composite materials have been expanding into markets where they are used for increasingly large structures. The aerospace industry is beginning to see the use of composites for fuselages and wings, and wind turbines are being made with composite blades, particularly for large offshore installations.

Using composites for large primary structures provides better stiffness, and lighter weight, but it also creates challenges. Producing large composite structures currently requires large and expensive autoclaves. As the size of an autoclave increases, it becomes expensive because of the difficulty of maintaining temperature control over a large volume of space, and due to the need to produce a vacuum. Curing times for composites are typically several hours, significantly limiting production volume. Another challenge with using composites for large structures is that repair becomes extremely important. It is typical to replace rather than repair small composite parts. This approach is cost prohibitive for something as large as an aircraft fuselage.

The use of carbon fiber embedded heaters can be a key enabler for both bringing down the cost of manufacturing, and improving repair of composites. Embedded heaters could be used to bond together smaller parts, which would shrink the size of the autoclaves needed. For repair, embedded heaters can provide efficient targeted heating, which would mean a more uniform temperature cure, and could potentially bond geometries and structures that pose

significant difficulties with current out-of-autoclave methods. Using carbon fiber material as the embedded heater has the advantage that when bonding composite adherends, this material can match the carbon fiber in the adherend so that no additional materials are added to the structure. This can avoid internal stresses due to material property mismatches, and using other heater materials such as a steel mesh can add complexities such as the need for surface treatments. Not adding additional materials to the bond is particularly beneficial for applications with rigid certification processes, such as aerospace. For these reasons, carbon fiber is an ideal material for embedded heating for joining CFRP adherends.

1.1 Challenges

One of the challenges of this work is that it should be applicable to a wide variety of materials and configurations. For any given repair, different insulation materials, or different adherends or adhesives could be used. For this reason, this research will focus on specific configurations, and develop empirical approaches for solving the sensing and control problems.

Using this technique in space creates its own unique challenges. It is advantageous that there is a natural vacuum in space, which will help in eliminating voids in the adhesive, except that this vacuum may cause the adhesive to flow out of the bondline, leaving resin starved areas in the bond. Without using vacuum bagging, it will also be necessary to develop a way to apply uniform over-pressure to the bond area.

1.2 Research contributions

This research is separated into three primary directions: sensing, control, and application.

1.2.1 Sensing

For composite repair using adhesive bonding, it is typical to control based on surface temperature measure at the edge of the bond area. The temperature is then controlled higher

than necessary with the knowledge that when heating at the surface, the temperatures will be lower in the bondline.

This research investigated a method for directly estimating temperatures within the bondline based on surface measurements. Adding thermocouples in the bondline is avoided as that would create weak spots within the joint. This is the topic of Chapter 2.

1.2.2 Control

There are many instances when adherends being joined would have non-uniform heat flow. An example of this is when one adherend has substructures with high conductivity, thereby creating a heat sink. In these cases, it is desirable to have a way to introduce non-uniform power to the bondline. This research investigates different techniques for doing this, both analytically and experimentally, which is covered in Chapters 3 and 4.

1.2.3 Application

The use of carbon fiber embedded heaters has several natural application, such as repair of aircraft, or bonding as part of a manufacturing process. For the general application, the single-lap joint is used for analysis and developing the technique. Additionally, one specific application that this research investigates is the feasibility of using this technique in space. There are several advantages to this over other joining techniques. It is efficient, since it introduces heat exactly where it is needed. It also adds very little material, is structurally efficient, and eliminates the need for drilling holes, as is common when using fasteners. In Chapter 5, this research investigates the feasibility of using carbon fiber embedded heaters for modular manufacturing and repair of space structures.

1.3 Research plan and timeline

The research plan timeline can be seen in Table 1.1. This research began investigating a way to estimate temperatures in the bondline using surface measurements. From this work, a paper was published [47] in the spring of 2016.

After sensing, a method for using boundary control was investigated, which lead to a conference paper, which was presented at the Manufacturing Science and Engineering Conference in June of 2017. This paper was purely experimental, and was then expanded with analytical results to produce a journal paper, which is in the process of submission at the time of writing this thesis.

Last, the specific application of using this technique as a method for adhesive bonding in space was investigated. This work will be presented at the 2017 Space Forum.

All of this work combined was used to complete the research goal in light of the various challenges presented.

Table 1.1: Research timeline.

Sensing	Control	Applications
Bondline temperature estimation • "Bondline Temperature Control for Joining Composites with an Embedded Heater", JMSE, 4/2016	Demonstrated multi-zoned heater • "Boundary control on embedded heaters for composite Joining", MSEC 2017, June 4-8 Tabbed heater modeling/design • "Boundary control of embedded heaters for composite joining", submitting to journal	Investigate method for space • "Investigation of embedded resistive heating for high strength adhesive bonding of modular space structures," to be presented at AIAA 2017 Space Forum

Chapter 2

ESTIMATING BONDLINE TEMPERATURE

2.1 Introduction

This chapter applies estimation and control techniques to achieve the desired bondline temperature for out-of-autoclave, adhesive bonding of carbon-fiber composite components. Rather than heat the entire system to cure the adhesive, e.g., in an autoclave, this research uses controlled heating targeted at the bondline with an embedded heater. The main contribution of this work is control using estimates of the bondline temperature without embedded sensors and wiring (of materials not intrinsic to the system) that can lead to reduced bond strength. Sensors placed outside the bond region are used to accurately estimate the bondline temperature to within 2% of the temperature range over which the bondline temperature is changed. Experimental results show that the estimated temperature can be used with feedback to achieve the desired, bondline, time-temperature profile that is needed to cure the adhesive. Additionally, shear lap joint tensile tests show that the resulting joint strength is comparable to specimens bonded using an external heat blanket.

This chapter applies estimation and control techniques to achieve the desired bondline temperature for adhesive-bonding of high-strength, carbon-fiber, composite components using an embedded heater. Carbon-fiber reinforced polymer (CFRP) composites with a thermoset resin such as epoxy can lead to some of the highest strength-to-weight ratio and rigidity. Therefore, such high-strength composites have been used in aerospace, sail boats, top-end bicycles, cars, and motorcycles. In the modern aerospace industry there has been a dramatic increase in the use of composites since the 1980s. This trend is demonstrated most clearly by the Boeing 787 Dreamliner, which is made of more than 50% by weight of

high-strength composites to increase fuel efficiency [42].

The use of autoclaves to cure the thermoset resin implies that these high-strength composites can be expensive to manufacture. In particular, the cost of manufacturing with autoclaves tends to be prohibitive as the size of the part increases. This has a larger impact in applications such as large wind-turbines, since the renewable-energy industry is cost sensitive. An approach to reduce the prohibitive cost of large autoclaves is to fabricate smaller parts and then join them. This approach is taken, for example, in the assembly of the fuselage sections of the Boeing 787 Dreamliner and the Airbus A350XWB. The fuselage is made of high-strength carbon-fiber composite sections that are produced in autoclaves, and then joined together. Nevertheless, at present, such joining operations are expensive, as it requires time and labor intensive drilling of holes for fasteners, e.g., [2, 13, 16, 36].

An alternative approach is to bond the composite parts together using high strength adhesives [7, 8], which would typically require an even larger autoclave to heat the entire system. Rather than heat the entire system to cure the adhesive, it is possible to only heat the area local to the bond. A common method for curing of the adhesive is to use an external surface heater such as a heat blanket or a heat lamp, which is placed on or near the surface of the adherend. With this approach, the outside temperature needs to be at a higher temperature than the bondline since the adherend acts as an insulator between the bondline and the heat source. The increased surface temperature can lead to thermal damage at the surface, and undesirable temperature gradients through the thickness of the adherend [46].

An embedded heater enables heating at the bondline where it is required rather than passing the heat from the surface to the bondline. This can reduce the energy and cost of joining composite components, when compared to the use of external heating methods such as heat blankets, and autoclaves. The proposed embedded heating approach can also be used to improve the adhesive-bonding of repair patches (over a damaged area), and thereby, extend the useful life of relatively large composite structures [45]. The embedded heating

approach has been found to create a uniform temperature distribution in the bondline [5], and the temperature is highest in the bondline where the heat is required for curing. This allows for lower overall part temperatures outside of the bondline, especially at the surface, which reduces the risk of thermal damage to the part and nearby temperature-sensitive equipment.

The current chapter uses resistive heating with a ply of carbon-fiber (e.g., previously proposed to cure thick composite parts [22,38,44,56]) to adhesively bond carbon-fiber-composite adherends [5]. Other materials such a stainless steel mesh can be used as the embedded resistive heater, e.g., used to cure adhesives [41]. Alternatively, epoxy resins can be cured through Joule heating of dispersed carbon nanotubes (CNTs) [29, 48]. Therefore, the epoxy resin could be used as an adhesive for bonding composites. Nonetheless, a substantial reduction in the cost of CNTs is needed for large-scale application of such techniques. Moreover, it is preferable to avoid using materials that are not intrinsic to the composite adherends in the bond-line. Adding additional materials to the composite structure increases the difficulty of predicting failure, and can add to the effort needed for certification in the aerospace industry. To avoid the use of materials (such as steel meshes) that are not intrinsic to the (carbon-fiber composite) adherends, the current work uses a carbon-fiber fabric as the embedded heater in the bondline. A challenge with using an embedded resistive heater for adhesive bonding of carbon-fiber composite adherends is that the carbon-epoxy adherend is electrically conductive. Our recent study [5] shows that the problem with conductive adherends can be overcome by using epoxy structural adhesive film on either side of the embedded heater to electrically insulate the embedded heater from the carbon-fiber composite adherends. However, to enable proper curing of these embedded adhesive films, methods are needed to sense and control the bondline temperature.

The main contribution of this work is control of the bondline temperature without using embedded sensors and wiring (of materials not intrinsic to the system) that can lead to

reduced bond strength. In particular, sensors placed outside the adherends (being bonded) are used to estimate the bondline temperature. Estimation of temperatures at locations where sensors are not available can be done using thermal models, e.g., to achieve controlled curing of relatively thick composite parts [34, 35, 37, 51]. However, such bondline temperature estimation using temperature sensors on the outside of carbon-fiber adherends can be challenging because material properties can be proprietary and difficult to obtain [17]. Therefore, this research uses an empirical modeling approach that is then used to design the estimator. This is accomplished in two parts. In the first dry-run part (i.e., without the adhesive) heat is applied at the bondline with the embedded heater along with a sensor at the bondline. The resulting data is used to empirically model the thermal dynamics and to design the controller and the estimator. Then, in the second part, bonding is achieved with adhesive while controlling the bondline temperature using the estimator and control designed in the first part. It is noted that the proposed bondline temperature estimation (for use with an embedded carbon-fiber heater) can be used to improve adhesive composite bonding with other approaches such as microwave heating [32, 49], induction heating [3, 28], external heaters such as thermal blankets [45], and potentially even with the autoclave-based approach.

2.2 Problem and proposed approach

2.2.1 Problem statement

The goal is to bond two carbon-fiber, composite adherends with structural-film adhesives, which are cured using an embedded heater that is sandwiched between the two adhesive films as shown in Fig. 2.1. For proper curing, the inside temperature T_i at the bondline needs to be controlled along a desired time-temperature profile $T_{i,d}$

$$T_i = T_{i,d}, \tag{2.1}$$

where the desired temperature profile has an initial ramp increase followed by a constant-temperature section as shown in Fig. 2.2, as recommended by the adhesive manufacturer. Although curing can be achieved even with some error in the attained temperature profile, accurate control of the bondline temperature can lead to more predictable bond properties. The problem is to control the inner, bondline temperature T_i without the use of an embedded sensor at the bondline, i.e., without the inner embedded thermocouple in Fig. 2.1.

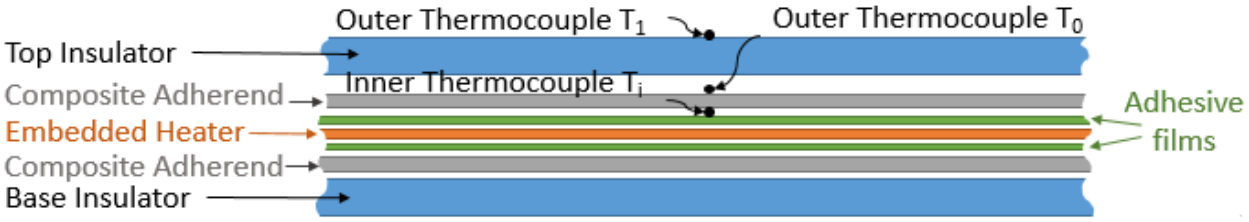


Figure 2.1: Schematic of experimental system during joining. An embedded heater is used to cure the structural-film adhesive, which bonds two carbon-fiber composite adherends. The entire system is insulated and is under vacuum pressure.

2.2.2 Proposed two-part approach

This chapter proposes a method of obtaining a model of the heat transfer across the composite adherend to estimate the inner, bondline temperature T_i (inside the adherend) from measured outer temperatures T_0 and T_1 (outside the adherend). The main challenge is that parameters of the model depend on the conditions under the specific manufacturing setting, and can vary for different adherends. To address this challenge, a two part procedure is used. In the first part, a dry-run is used to estimate the model parameters experimentally, under conditions similar to bonding, but without adhesives, as illustrated in Fig. 2.3. The structural-film adhesive used during joining in Fig. 2.1 is replaced with (non-adhesive) Teflon electrically-insulating sheets to prevent adhesive bonding, and electrical shorting, during experimental estimation of model parameters. During the second bonding part, the change from insulating Teflon sheets to adhesive film does not change the heat transfer model across the adherend.

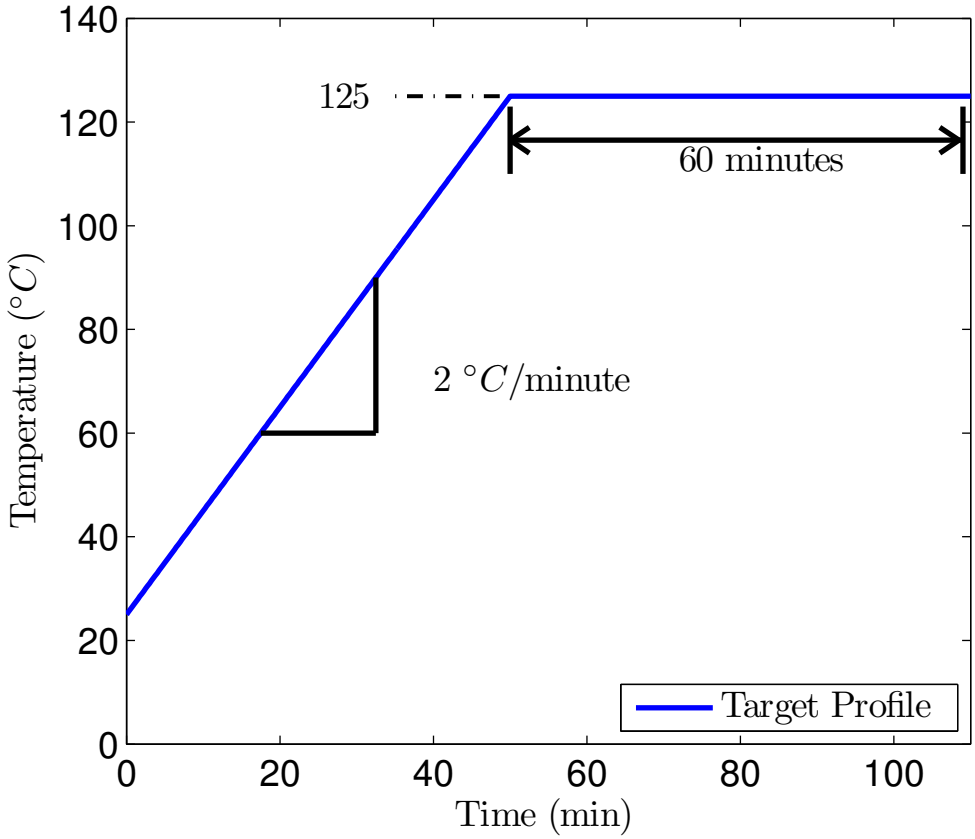


Figure 2.2: The desired temperature profile $T_{i,d}$ at the bondline, as recommended by the adhesive manufacturer. Note that the initial value $T_{i,d}(0)$ at time $t = 0$ is room temperature, which depends on the experimental conditions.

Therefore, the thermal properties of the adherend and the model should not change during the second part, when bonding is accomplished with structural-film adhesive. Hence the estimator and controller designed in the first part without adhesive can be used to achieve the desired bondline temperature while bonding with adhesive.

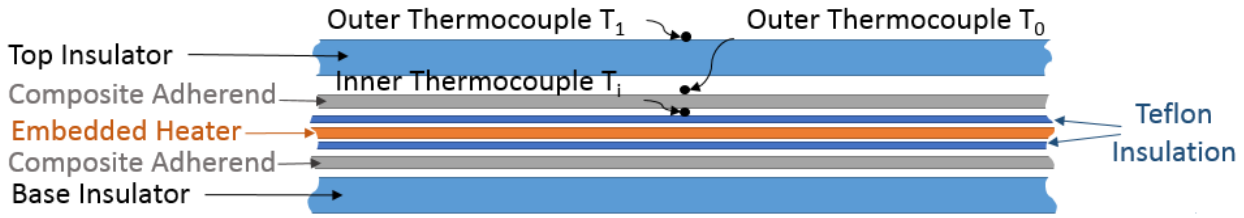


Figure 2.3: Schematic of experimental system (without bonding) for controller and estimator design. The structural-film adhesive in Fig. 2.1 is replaced with Teflon insulating sheets to prevent adhesive bonding during experimental estimation of model parameters. The entire system is still under vacuum pressure as in the joining case.

2.3 Controller and estimator design

The controller and estimator are experimentally developed and evaluated in this first dry-run part — without the adhesive. The structural-film adhesives in Fig. 2.1 are replaced with Teflon insulation sheets, as shown in Fig. 2.3 to avoid bonding. The process consists of the following four steps: (1) measure and model the thermal dynamics; (2) design the controller using the model in step 1; (3) use the data from step 1 to design an estimator to predict the bondline temperature for the controller designed in step 2; and (4) evaluate the controller and estimator.

2.3.1 Measure and model thermal dynamics

A step input (a constant-amplitude AC voltage) was applied across the embedded heater with the configuration shown in Fig. 2.3, and temperatures outside the adherend (T_0, and T_1), and inside the adherend at the bondline (T_i) were measured. The measured response of the bondline temperature T_i is shown in Fig. 2.4, which is modeled as a first order system $G_{T,i}$ of the form

$$G_{T,i}(s) = \frac{T_i(s)}{u(s)} = \frac{K}{\tau s + 1}. \tag{2.2}$$

Since the heat generated by the embedded heater is proportional to the square of the input voltage, the input u to the system is considered to be

$$u = \alpha\, V^2, \tag{2.3}$$

$$\text{where } \; 0 \leq \alpha \leq 1. \tag{2.4}$$

Here V is voltage applied to the embedded heater, and the effective power is adjusted by changing the factor α through pulse width modulation (PWM). For the experiments in the current chapter, the voltage V was chosen to be $19.2(V)$, which enables a bondline temperature of approximately $125\ ^{\circ}C$ when $\alpha = 0.5$.

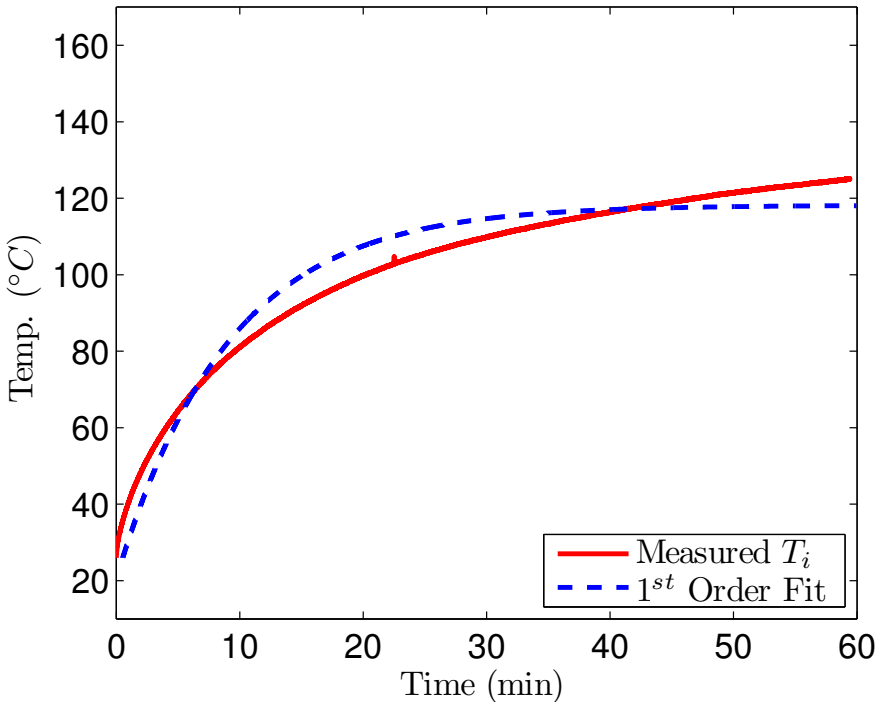


Figure 2.4: Open-loop step response T_i of the thermal system: (dotted line) response of fitted model $G_{T,i}$ in Eq. 2.2; and (solid line) experimentally measured.

The model parameters in Eq. 2.2 were calculated to be

$$\tau = 541.5(s), \quad K = 0.185(^{\circ}C/V^2) \tag{2.5}$$

by using a least-squares fit (with MATLAB) of the experimental data to minimize the difference between the measured inner temperature and predicted inner temperature,

$$\sum_n \left| T_i(t_n) - \begin{bmatrix} u(t_n) & -\dot{T}_i(t_n) \end{bmatrix} \begin{bmatrix} K \\ \tau \end{bmatrix} \right|^2, \tag{2.6}$$

where $t_n = nT_s$ are the sampling instants and $T_s = 0.25(s)$ is the sampling period. The open-loop step response of the fitted model $G_{T,i}$ in Eq. 2.2 is compared to the experimentally measured response in Fig. 2.4. The first order model $G_{T,i}$ captures the major portion of the response, although errors are present in the rise time, and the model does not capture slow drift in the temperature in the experimental response. It is shown in the subsequent sections that the controller can account for these modeling errors, especially since the desired target profile in Fig. 2.2 is varying slowly relative to the time constant τ of the open-loop system in Eq. 2.5.

2.3.2 Controller design

A proportional-integral (PI) controller $C(s)$ of the form

$$C(s) = \frac{K_p s + K_i}{s}. \tag{2.7}$$

is used to control of the inner, bondline temperature T_i, with the control schematic in Fig. 2.5, which assumes that the inner temperature measurement is available to the controller.

With the PI controller in Eq. 2.7 with the open-loop system $G_{T,i}(s)$ in Eq. 2.2 , the

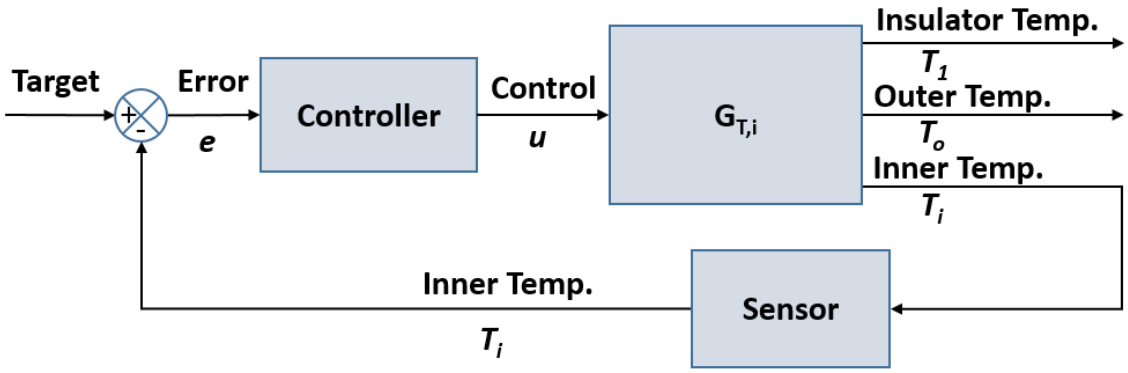


Figure 2.5: Block diagram of control scheme when a sensor is available to measure the inner, bondline temperature T_i.

closed-loop model $G_{T,i,CL}(s)$ takes the form

$$G_{T,i,CL}(s) = \frac{T_i(s)}{u(s)} = \frac{K(K_p s + K_i)}{\tau s^2 + (1 + KK_p)s + KK_i} \tag{2.8}$$

$$= \frac{as + b}{s^2 + 2\zeta\omega_n s + \omega_n^2}. \tag{2.9}$$

The poles of the system were chosen to be critically damped and a time constant that was 4 times smaller than the open-loop time constant τ, i.e.,

$$\zeta = 1, \tag{2.10}$$

$$\frac{1}{\zeta\omega_n} = \frac{1}{\omega_n} = \frac{\tau}{4}, \tag{2.11}$$

which results in the following proportional K_p and integral K_i controller gains

$$K_p = \frac{2\zeta\omega_n\tau - 1}{K} = 37.9361, \tag{2.12}$$

$$K_i = \frac{\tau\omega_n^2}{K} = 0.1601. \tag{2.13}$$

A discrete-time version $C(z)$ of the controller $C(s)$ was obtained using a zero-order hold [33], so that

$$C(z) \;=\; (1 - z^{-1})\mathcal{Z}\left\{\mathcal{L}^{-1}\left[\frac{C(s)}{s}\right]\right\}, \tag{2.14}$$

where $\mathcal{L}^{-1}$ is the inverse Laplace transform, and $\mathcal{Z}$ is the z-transform. The resulting discrete-time controller was found to be

$$C(z) = (1 - z^{-1})\mathcal{Z}\left\{\mathcal{L}^{-1}\left[\frac{K_p}{s} + \frac{K_i}{s^2}\right]\right\} \tag{2.15}$$

$$= (1 - z^{-1})\left(\frac{K_p}{1 - z^{-1}} + \frac{K_i T_s z^{-1}}{(1 - z^{-1})^2}\right) \tag{2.16}$$

$$= \frac{K_p - (K_p - K_i T_s)z^{-1}}{1 - z^{-1}} \tag{2.17}$$

$$= \frac{K_{p,d} - K_{i,d}z^{-1}}{1 - z^{-1}}, \tag{2.18}$$

where

$$K_{p,d} = K_p \tag{2.19}$$

$$K_{i,d} = (K_p - K_i T_s) \tag{2.20}$$

and T_s is the sampling period. The discrete-time controller $C(z)$ was implemented as shown in Fig. 2.6 with a saturation block that limits the power applied to the embedded heater as described by Eq. 2.4.

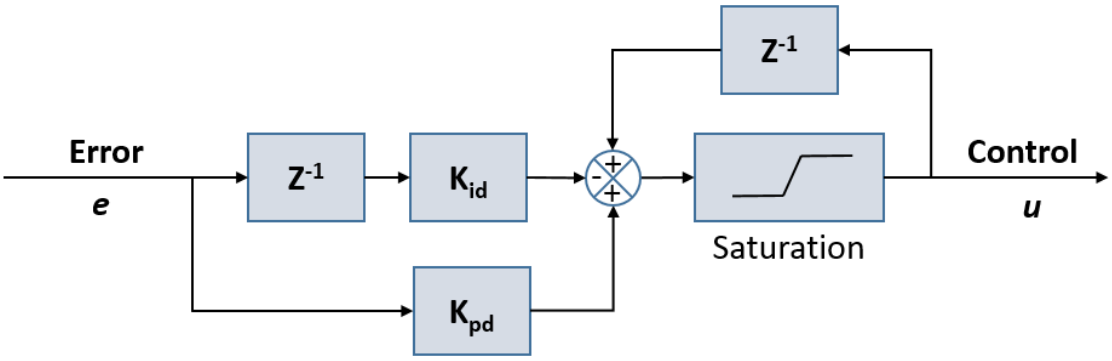


Figure 2.6: Schematic of the discrete-time implementation of the controller ($C(z)$ in Eq. 2.18) with a saturation block.

2.3.3 Bondline temperature estimator design

A relationship between the inner bondline temperature T_i, and the outer temperatures T_o (outside the composite adherend in Fig. 2.3), and T_1 (outside the first layer of insulation) is obtained using a one dimensional, lumped-capacitance, heat transfer model [10]. The change in outer temperature T_o is related to the heat entering and leaving a small region around the thermocouple as

$$(\rho L)c_p\frac{dT_o}{dt} = \frac{k_a}{L_a}(T_i - T_o) - \frac{k_1}{L_1}(T_o - T_1), \tag{2.21}$$

where k_a is the thermal conductivity of the adherend, L_a is the thickness of the adherend, ρ is the mass density of the adherend, L is the effective thickness of the layer around the thermocouple and c_p is the associated specific heat capacity, k_1 is the thermal conductivity of the insulation layer, and L_1 is the thickness of the insulation layer as in Fig. 2.7. The first term on the right hand side of Eq. 2.21 represents heat flow from the embedded heater towards the outer temperature T_o thermocouple through the adherend, and the second term represents heat flow away from the outer temperature T_o thermocouple through insulator 1 in Fig. 2.7.

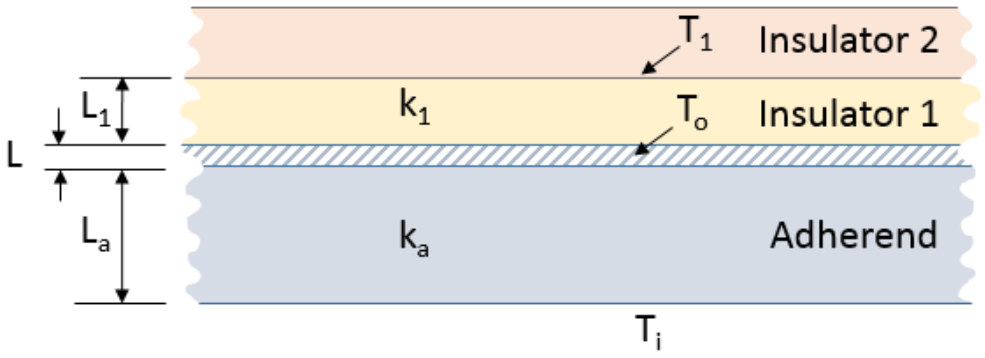


Figure 2.7: Schematic of temperature model through adherend.

The estimate $T_{i,e}$ of the inner, bondline temperature T_i can be written in terms of the outer temperatures T_o and T_1 from Eq. 2.21 as

$$T_{i,e} = c_1 T_o + c_2 \dot{T}_o + c_3 T_1, \tag{2.22}$$

where

$$c_1 = \frac{k_1 L_a}{k_a L_1} + 1, \tag{2.23}$$

$$c_2 = \frac{(\rho L) c_p L_a}{k_a}, \tag{2.24}$$

$$c_3 = -\frac{k_1 L_a}{k_a L_1}. \tag{2.25}$$

2.3.4 Estimator-parameter determination

The parameters of the estimator c_1, c_2, c_3 were found experimentally, using the measured inner temperature T_i, outer temperature T_o, as well as the temperature outside the first insulator T_1. This data is available from the step response of the system shown in Fig. 2.4. In particular, a least squares fit (with MATLAB) of the experimental data to minimize the

difference between the predicted inner temperature $T_{i,e}$ from the model in Eq. 2.22, and the measured bondline temperature T_i

$$\sum_n \left| T_i(t_n) - c_1 T_o(t_n) - c_2 \dot{T}_o(t_n) - c_3 T_1(t_n) \right|^2, \tag{2.26}$$

where t_n are the sampling instants, yields

$$c_1 = 1.2831, \quad c_2 = 0.1930\ s, \quad c_3 = -0.1937. \tag{2.27}$$

The result of the least squares fit, i.e., the estimated inner, bondline temperature $T_{i,e}$, is close to the inner, bondline temperature T_i as shown in Fig. 2.8a. The error between the measured bondline temperature T_i and the estimated temperature $T_{i,e}$,

$$e_E = T_i - T_{i,e}, \tag{2.28}$$

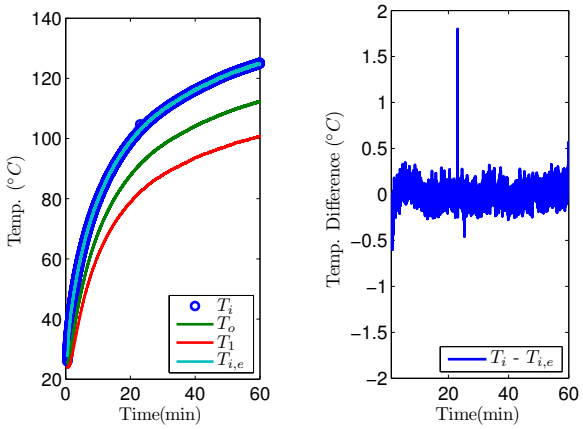
is shown in Fig. 2.8b, and this error is bounded within $\pm 0.3\ ^\circ C$, after initial transients, with a root-mean-square (rms) error of $0.132\ ^\circ C$.

2.3.5 Sensitivity of estimator parameters

The error e_E in the estimated temperature $T_{i,e}$ due to parameter errors in the sensing scheme cannot be corrected using feedback control. Therefore, the proposed experimental determination of the estimator parameters c_1, c_2, c_3 (using a dry-run) is particularly suitable for composite bonding because thermal and material property dependent parameters such as

$$p_1 = k_1, \quad p_2 = k_a, \quad p_3 = \rho\ L\ c_p \tag{2.29}$$

can vary substantially. For example, the thermal conductivity in a composite is $12 - 15$ times higher in the fiber direction than perpendicular to the fiber direction [54]. Thermal



(a) (b)

Figure 2.8: Results of step response as in Fig. 2.4 for (a) the measured temperatures (inner, bondline temperature T_i, and outer temperatures T_o, T_1), and the estimated temperature $T_{i,e}$ calculated from Eq. 2.22, and (b) the error e_E in the fit from Eq. 2.28.

conductivity is also dependent on processing conditions [15], which can vary between samples. This makes it challenging to determine parameter values p_1, p_2, p_3 a-priori.

A sensitivity analysis of the error e_E in the estimated temperature $T_{i,e}$ was performed by modifying the estimator parameters c_1, c_2, and c_3 as they would change from the fitted values in Eq. 2.27 assuming they were defined by material properties as in Eqs. 2.23, 2.24 and 2.25. In particular, when the parameter $p_1 = k_1$ changed by $\delta p_1\%$, the fitted estimator parameters were changed as

$$c_1 = (1.2831 - 1)\left(1 + \frac{\delta p_1}{100}\right) + 1 \tag{2.30}$$

$$c_2 = 0.1930, \tag{2.31}$$

$$c_3 = -0.1937\left(1 + \frac{\delta p_1}{100}\right). \tag{2.32}$$

When the parameter $p_2 = k_a$ changed by $\delta p_2\%$, the fitted estimator parameters were changed as

$$c_1 = \frac{1.2831 - 1}{(1 + \delta p_2/100)} + 1 \tag{2.33}$$

$$c_2 = \frac{0.1930}{(1 + \delta p_2/100)}, \tag{2.34}$$

$$c_3 = \frac{-0.1937}{(1 + \delta p_2/100)}, \tag{2.35}$$

and when the parameter $p_3 = \rho\ L\ c_p$ changed by $\delta p_3\%$, the the fitted estimator parameters were changed as

$$c_1 = 1.2831, \quad c_2 = 0.1930\left(1 + \frac{\delta p_3}{100}\right), \quad c_3 = -0.1937. \tag{2.36}$$

The resulting rms value of the error e_E in the estimated temperature $T_{i,e}$ due to potential parameter variations are shown in Fig. 2.9. The simulation results show that the rms value of the error e_E in the estimated temperature $T_{i,e}$ can increase by an order of magnitude when there is 10% error in the estimated material properties, such as the thermal conductivity k_1 of the insulation layer. The results indicate that the proposed experimental determination of the estimator parameters for the specific repair conditions is important to ensure accurate estimation (and therefore accurate control) of the bondline temperature T_i.

2.3.6 Controller and estimator evaluation

Evaluation of the controller and estimator was done in the dry-run configuration described in Fig. 2.3. The bondline temperature T_i was controlled to track the cure profile described in Fig. 2.2 using the control design in Eq. 2.18, the estimated temperature $T_{i,e}$ with the parameters from Eq. 2.27, and the control scheme shown in Fig. 2.10. The results of this experiments are shown in Fig. 2.11.

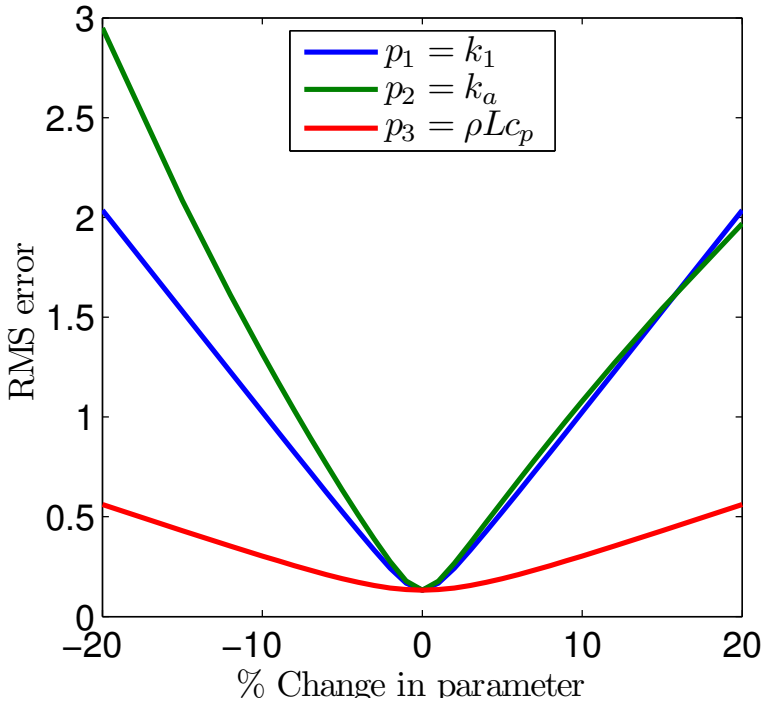


Figure 2.9: Root mean square value of the error e_E in the estimated temperature $T_{i,e}$ due to variations in the parameters $p_1 = k_1$, $p_2 = k_a$, and $p_3 = \rho\ L\ c_p$.

The estimated error e_E is between $\pm 2\ ^\circ C$ as seen in Fig. 2.11b, with rms error 1.134 during the ramp portion of the cure profile, and $\pm\ 1^\circ C$ with rms error 0.249 while at steady state. Note that most of the error arises from high-frequency noise. This represents an estimate error $(e_E/T_r \times 100)$ that is less than 2% of the temperature range T_r over which the desired temperature is varied, i.e.,

$$T_r \quad = \max_t T_{i,d}(t) - T_{room} \approx 125^\circ C - 25^\circ C = 100^\circ C \tag{2.37}$$

where T_{room} is the room temperature. Similarly, the percent tracking error $(e_d/T_r \times 100)$

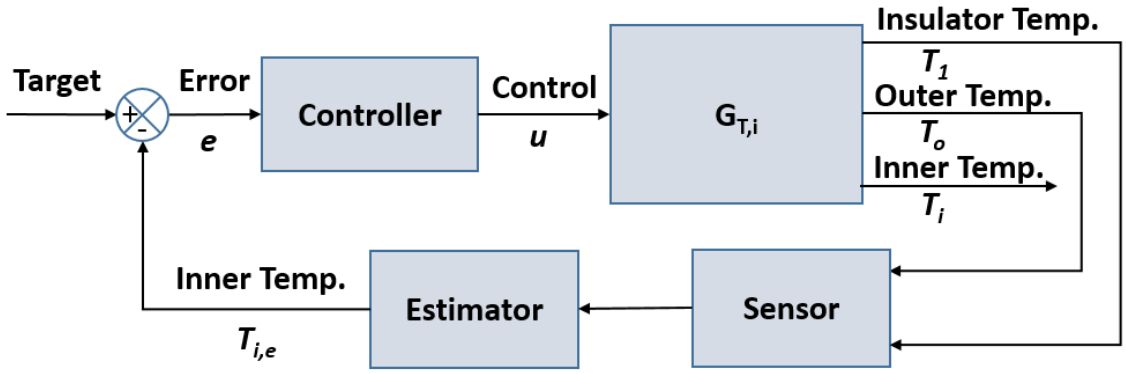
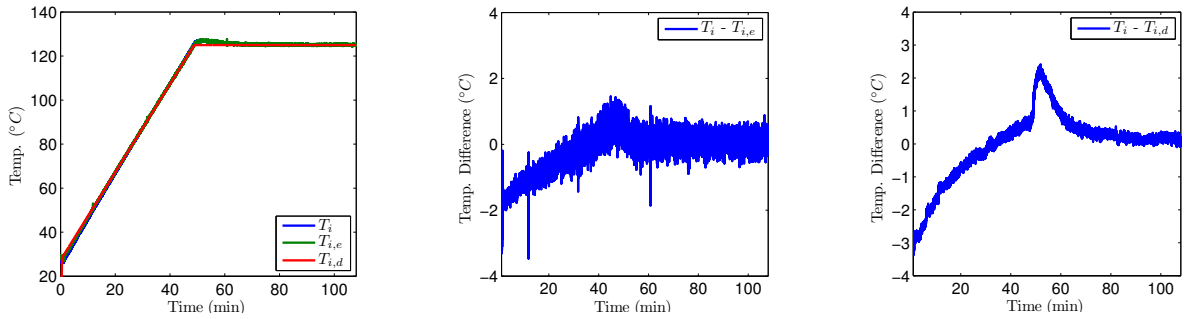


Figure 2.10: Block diagram of control system during joining. The inner, bondline temperature T_i is controlled using the estimated inner, bondline temperature $T_{i,e}$.



(a) (b) (c)

Figure 2.11: Experimental results with cure profile without adhesive with (a) measured inner bondline temperature T_i, estimated inner temperature $T_{i,e}$, and desired inner temperature $T_{i,d}$ with the control as in Fig. 2.10, (b) estimator error e_E as in Eq. 2.28, and (c) tracking error e_d as in Eq. 2.38.

between the desired and achieved temperature

$$e_d = T_i - T_{i,d}, \tag{2.38}$$

is below 3% of the tracking range T_r, as seen in Fig. 2.11c.

2.4 Experimental bonding results

In this second part of experiments, the Teflon insulation sheets shown in Fig. 2.3 were replaced with structural-film adhesive as shown in Fig. 2.1 and the adherends were bonded. Shear lap joint tensile tests were performed to compare the resulting joint strength to that of specimens bonded using a commonly-used, external heat blanket.

2.4.1 Experiment materials and methods for bonding

The adherends used for these experiments were produced with HexPly 155, woven prepreg, with a cure temperature of $127°C$. Each adherend was made of 16 layers, with a $[0]_{16}$ stacking sequence. The thickness after curing was measured at 5.81 mm.

During bonding experiments, two layers of structural-film adhesive (each 0.0635 mm thick, Scotch-Weld AF 163-2U) were used for each joint. For the embedded resistive heater, unidirectional carbon-fiber fabric (Fiber-glast #2596, 12K, 0.152 mm thick) was used. The layers of adhesive and embedded fabric were placed as shown in Fig. 2.1.

A 1.27 cm thick silicon rubber material was used to thermally insulate the experimental system from the bottom. Covering each experiment was one layer of 3.175 mm of silicon rubber insulation, and one layer of insulating polyester breather fabric. All of this was covered with a vacuum bag. Each experiment was run under 82.7 kPa (0.817 atm) of vacuum pressure, as was recommended by the adhesive manufacturer. A picture of this setup during heating with the embedded heater can be seen in Fig. 2.12.

To power the embedded resistive heater, a 120 Volt, 15 Amp Variac power supply was connected in series with a solid state relay (SSR), and controlled by an Arduino Mega microcontroller board.

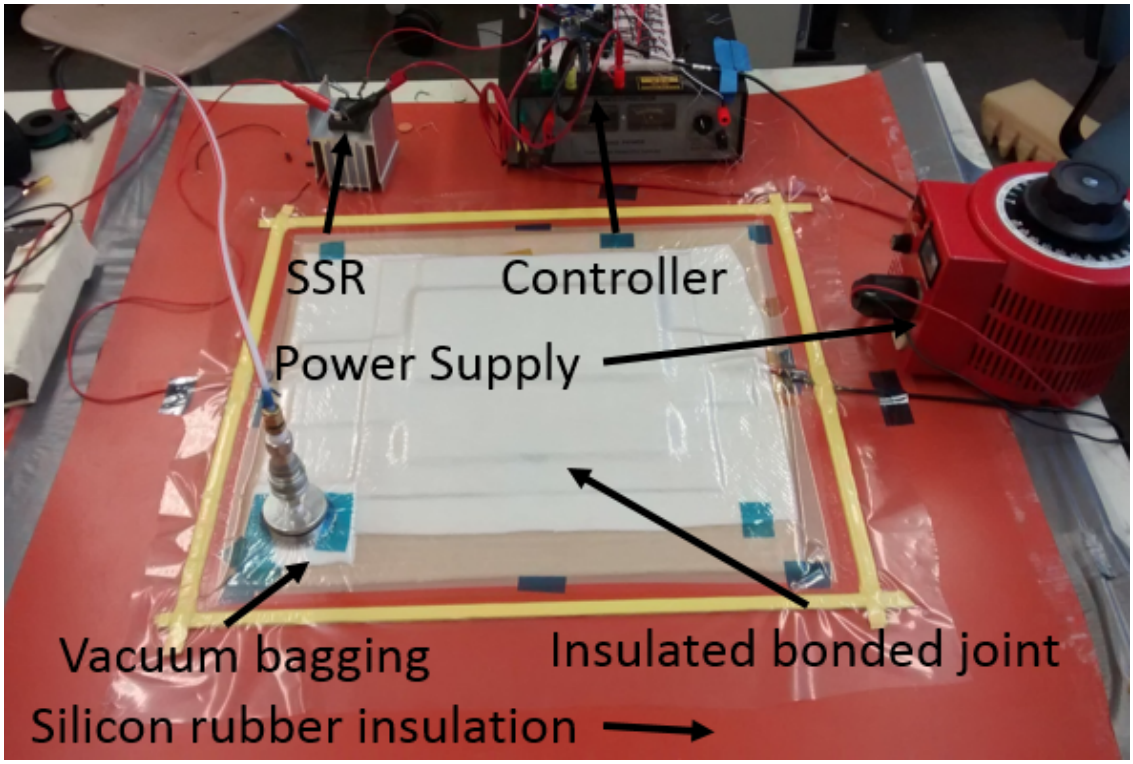
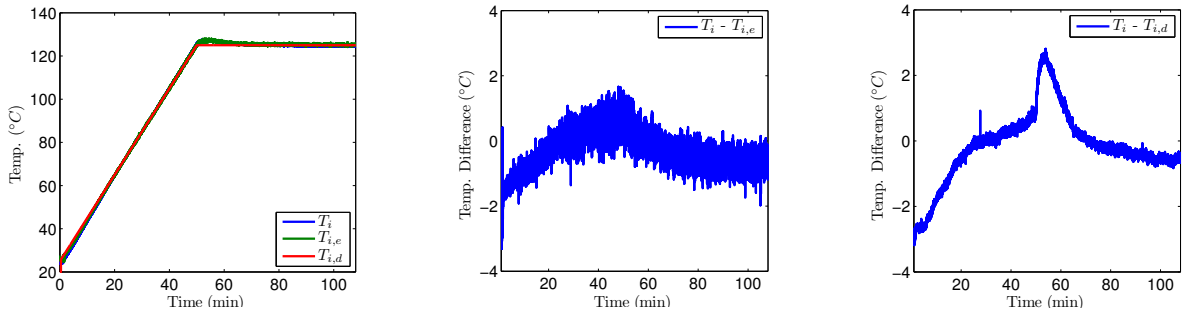


Figure 2.12: Photo of the experimental setup during bonding with the embedded resistive heater.

2.4.2 Bondline temperature estimation and control

The estimator and controller designed in Section 2.3 were used to track the cure profile in Fig. 2.2 while bonding a single lap joint. In these bonding experiments a pristine (new) piece of embedded heater fabric was placed between structural-film adhesives as shown in Fig. 2.1. A thermocouple was left inside of the bond to measure the estimation error. The resulting temperature profile achieved during this bonding experiment can be seen in Fig. 2.13a. The error e_E in the temperature estimate, when comparing to the actual measured internal temperature can be seen in Fig. 2.13b.

During these bonding experiments, the error is similar to the dry-run case without ad-



(a) (b) (c)

Figure 2.13: Experimental results with cure profile during bonding with adhesive with (a) measured inner bondline temperature T_i, estimated inner temperature $T_{i,e}$, and desired inner temperature $T_{i,d}$, and (b) estimator error e_E as in Eq. 2.28, and (c) tracking error e_d as in Eq. 2.38.

hesive. Fig. 2.13b shows that the estimated error e_E is between $\pm 2\ ^\circ C$ with rms error 1.375 during the ramp portion of the cure profile, and $\pm\ 0.5\ ^\circ C$ with rms error 0.588 while at steady state. The estimated error e_E then is below 2% of the temperature range T_r. Similarly, the error e_d between the desired and achieved temperature is below 3% of the temperature range T_r, which can be seen in Fig. 2.13c.

An advantage of the embedded heater method is that the heat is being applied in the bondline where it is needed. This leads to highest temperatures within the bond, and lower temperatures elsewhere. The extent of the difference in temperature at the bondline T_i compared to the temperature outside of the adherend T_0, and outside of the first layer of insulation T_1, can be seen in Fig. 2.14a. Using the embedded heating method, the temperature outside the adherend is about $14^\circ C$ cooler, while with external heating, the temperature outside the adherend is up to $12^\circ C$ hotter as seen in Figs. 2.14b and 2.14c.

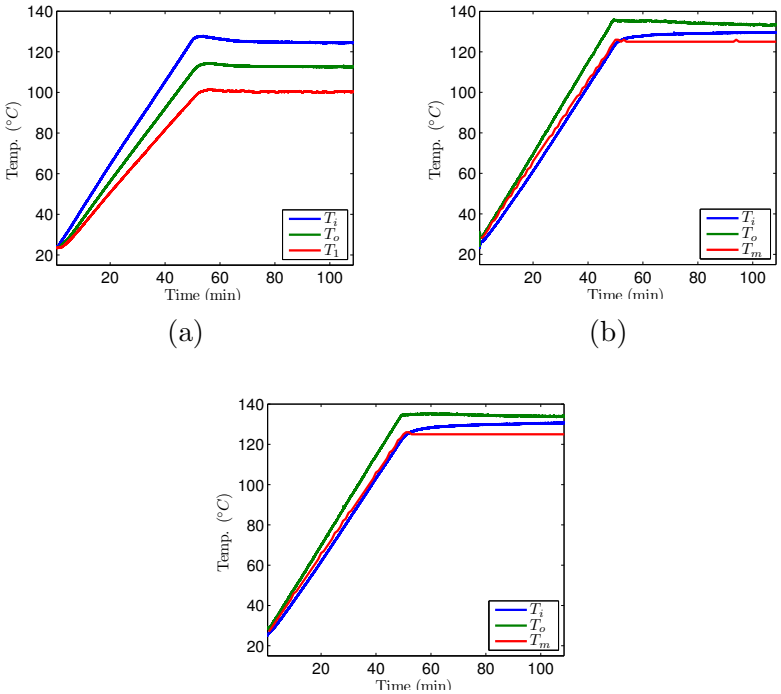


Figure 2.14: Temperature profiles for bonding with (a) an embedded heater, (b) an external heat blanket with fabric in the bondline and, (c) an external heat blanket without fabric in the bondline.

2.4.3 Adhesive bonding using external heating

The proposed method is compared to a common industrial bonding technique that uses external heating. The heat blanket used was 30.5 cm square, rated for 120 V and 720 W, produced by Heatcon, Inc. For temperature control, a Heatcon composite repair system (model HCS9000B) was used, also known as a hot bonder. There were three control thermocouples placed along the edge of the bond and averaged, which is represented in Fig. 2.15 as T_m. Thermocouple measurements for the bondline temperature T_i and the surface temperature T_0 were also monitored for comparison with the embedded heating method.

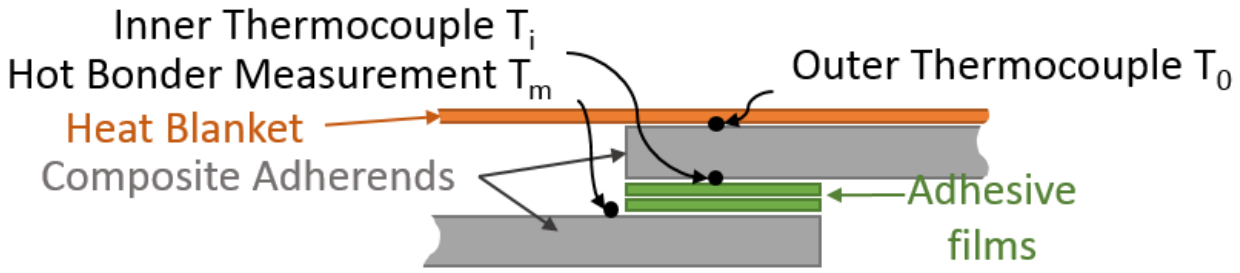


Figure 2.15: Schematic of bonding with Heatcon hot bonder with external heater.

Embedding a carbon-fiber fabric in the bondline, increases thickness and stiffness (though not necessarily the strength) of the bondline. To study the potential effect of increased thickness on the shear lap strength of the bond, two single lap joints were made. One was made with the configuration shown in Fig. 2.15 without embedded fabric between the two layers of adhesive. The second single lap joint was made with a single layer of carbon-fiber composite fabric (same fabric type used in the embedded heating method) between the layers of adhesive in the joint. This was done to compare the strength of the bond with the joint made using the embedded resistive heating method, and to determine if any strength difference occurred due to the presence of the fabric in the bondline.

The temperature profiles for both of these experiments can be seen in Figs. 2.14b and 2.14c. Since heating is from the outside, the temperature at the surface of the adherend T_0 is higher than inside the bond T_i, as would be expected. In this experiment, the outer temperature T_0 was up to $12°C$ hotter than the inner temperature T_i. Also, the control temperature T_m, which was measured at the edge of the bond is also different than the actual temperature inside the bond T_i, differing by over $5°C$.

2.4.4 Tensile testing of single lap joints

Tensile testing of single lap joints (ASTM D5868) was done in a 100 kN capacity Instron test frame. The displacement rate was set to 1.27 mm/min, and all specimens were loaded until failure. Tabs were added at the end of the specimens as seen in Fig. 2.16 to reduce bending forces. Seven single lap joints with 2.54 cm width were cut from the bonded panels. A schematic of the tensile specimens as cut from the single lap joints can be seen in Fig. 2.16, with the load applied by the Instron load frame as indicated in the diagram. The section in the center where the thermocouple is placed within the bondline was not tested, as the thermocouple could have an effect on the bond strength.

To determine the effect of the extra layer of carbon-fiber within the bondline, samples were made using external heating with and without fabric embedded in the bondline. The results of the tensile testing on the single lap joints are displayed in Table 2.1, and in a bar graph displaying the mean data in Fig. 2.17. The added fabric between the structural adhesive films can reduce the bond strength with similar curing conditions. For example, with the external heat-blanket, the mean load at failure of the lap joint reduced from 14.02 kN without embedded fabric to 13.35 kN with embedded fabric, as seen in Table 2.1, which is consistent with previous results [5]. The reduction in the tensile strength of the lap-joint with the addition of fabric in the bondline could arise from reduced flow of the resin between the adherends, or due to the change in the overall thickness of the bond. While the strength reduction due to an increase in the bond thickness (with the addition of the embedded fabric) would be similar for both (embedded and external) heating methods, the bondline heating with the embedded heater could lead to increased resin flow at the bondline when compared to the external heater.

The results show that the bond strength achieved with the embedded resistive heater method is at-least comparable (if not better) to that achieved with the external hot-bonder method. The variance in the tensile testing data for the embedded heater method was higher

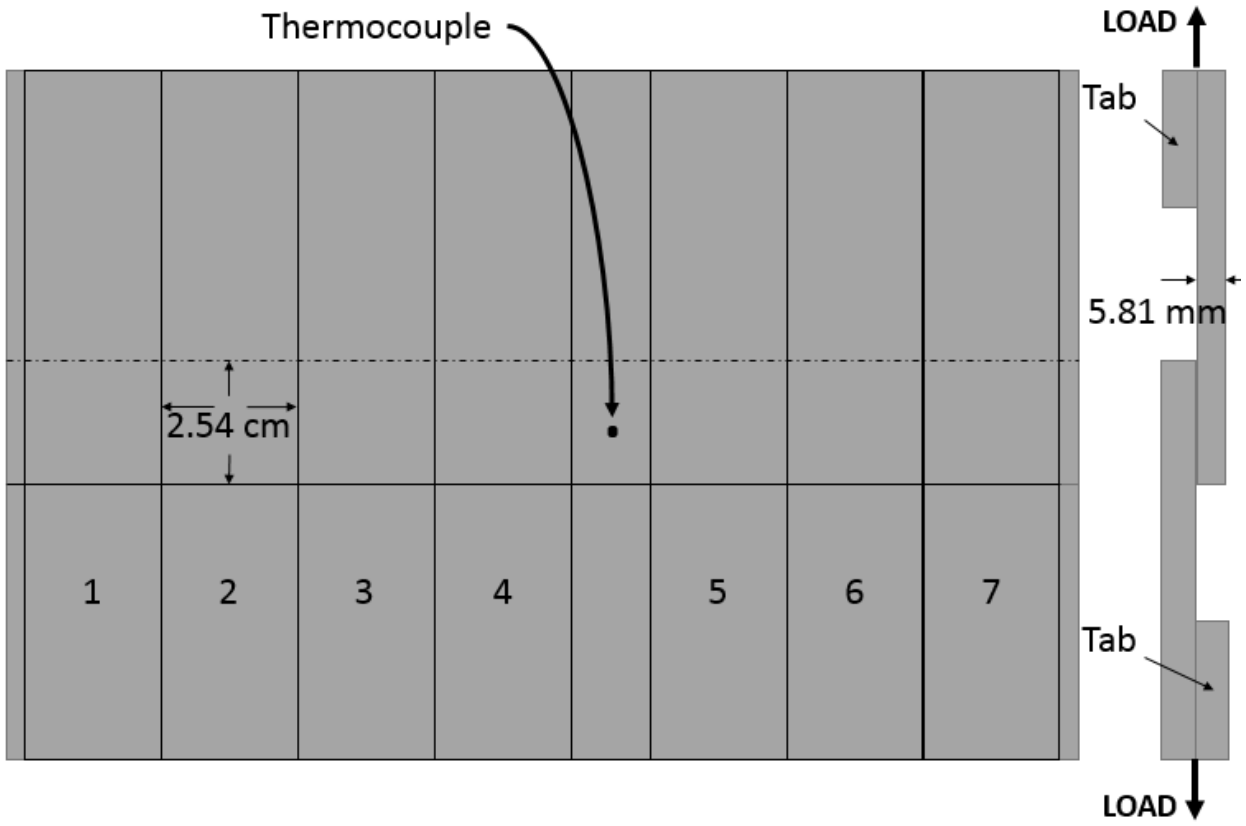


Figure 2.16: Schematic of specimens cut from single lap joint cut for tensile testing with top-view (left), and side-view after adding tabs (right).

than results with the hot bonder experiments, which could be indicative of a small sample size. Nevertheless, the mean load at failure with the embedded heater (14.71 kN) is similar to the mean load at failure (14.01 kN) with the external heater without embedded fabric in the bondline. Moreover, the mean load at failure with the embedded heater (14.71 kN) is statistically better (double-sided Welch t-test [53] with 95% confidence) than the mean load at failure (13.35 kN) with the external heater and with the embedded fabric in the bondline.

Table 2.1: Load (kN) at failure for single lap joint tensile test specimens produced with an embedded resistive heater (ERH), with a hot bonder (HB) without embedded fabric, and a hot bonder with embedded fabric.

Sample	ERH	HB w/o Fabric	HB w/ Fabric
1	17.09	13.21	12.59
2	15.63	14.57	13.92
3	12.41	12.52	13.87
4	14.49	14.91	12.30
5	14.38	14.28	13.73
6	14.62	14.75	13.06
7	14.38	13.86	13.97
Mean	14.71	14.01	13.35
Std. Dev.	1.42	0.88	0.69

2.5 Conclusions

This chapter developed a method to use external sensors for estimating and controlling inner, bondline temperatures when joining composite materials with an embedded resistive heater. Tensile testing results showed that the bond strength achieved with the embedded resistive heater method is comparable (if not better) to that achieved with the external heat-blanket method. However, the proposed method applies heat directly at the bondline where it is needed rather than transfer heat from the outside, which requires less energy. Moreover, the highest temperatures tend to be at the bondline with the embedded heater method, which avoids high temperatures at the outside, as is the case with external heaters, which can lead to thermal damage at the surface. Ongoing efforts in this area include investigating effects of different adherend properties along the bonded region, and developing methods for more detailed modeling, estimation and control that might be needed for regions with varying thermal properties.

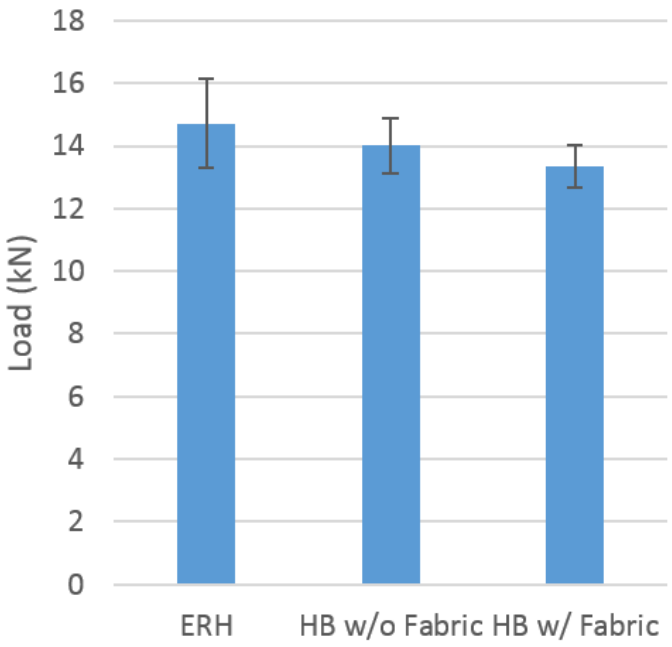


Figure 2.17: Mean tensile test load at failure for single lap joints made with an embedded resistive heater (ERH), hot bonder (HB) without embedded fabric, and hot bonder with embedded fabric, all with one standard deviation error bars.

Chapter 3

NON-UNIFORM POWER GENERATION USING EMBEDDED HEATERS

3.1 Introduction

This chapter investigates the use of boundary control of embedded resistive heaters as actuators for precision and targeted temperature control when curing high strength adhesives when joining composite adherends. Targeted temperature control is particularly useful for out-of-autoclave (OoA) applications where a uniform heating technique will lead to non-uniform temperatures in the bondline while curing, for example, in the case of non-uniform heat loss when heatsinks are present. Non-uniform temperatures in the bondline can cause inconsistent and uneven curing and a weakened joint. This work devises an empirical method (that can be applied when material parameters and models are not readily available) to model and predict the desired power generation for a specific joining application, and uses this model to design embedded heaters to minimize temperature variation in the bondline.

This chapter looks at two approaches for achieving non-uniform power generation. The first is through the use of multiple heaters, each connected to an independent controller. The second approach is through boundary control on a single embedded heater, with multiple tabs along the edges. By applying different voltages to different tabs, a non-uniform power is generated within the heater, which can be targeted to produce power in such a way to produce a more uniform temperature in the bond area of a joint. A schematic of a tabbed heater can be seen in Fig. 3.1.

This chapter will first develop an approach to determine the desired power within the bondline for a specific experimental configuration. Then the two types of heaters describes

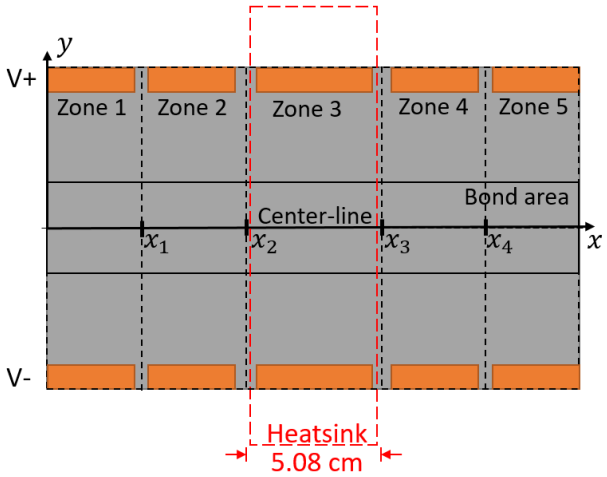


Figure 3.1: A heater made with copper tabs on two edges to generate non-uniform power.

above will be investigated analytically to determine how well they are able to produce the desired power.

3.2 Problem Description

This chapter uses the well-studied single-lap joint, e.g., [20, 40] as an example system to evaluate the proposed boundary control approach. A schematic of this system can be seen in Fig. 3.2, where the bond area is marked with cross-hatching, and a heatsink is placed under the lap joint so that it is parallel to the y-dimension of the bond area.

When heating a joint with an embedded heater that generates uniform power, such as in Fig. 3.3, a significant drop in temperature is seen towards the center of the bond area. This was done experimentally, and can be seen in Fig. 3.4, where the measured steady state temperatures in the bond area $T_{m,u}$ are shown as red circles.

The analysis and results in this and the next chapter focus on achieving uniformity at the target temperature $T_{d,ss}$, which is the steady state temperature of a ramp and hold profile $T_d(t)$, as shown in Fig. 3.5, because this is where a majority of the curing occurs. The target

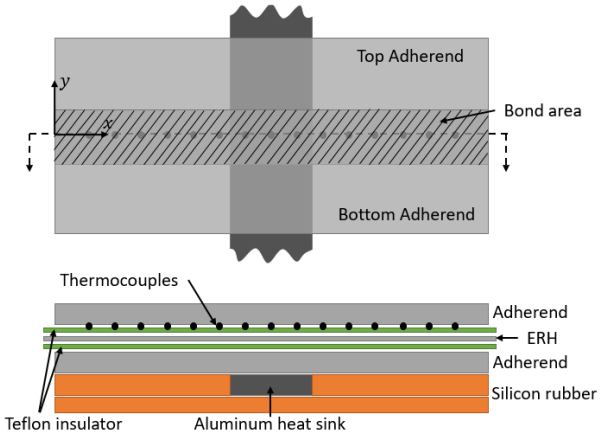


Figure 3.2: Schematic of a single-lap joint with an embedded resistive heater within the bondline for heating and curing adhesive. A heatsink is placed under the center of the bond area, while the rest of the structure is insulated from below.

temperature $T_{d,ss}$ is also where it is most difficult to obtain a uniform temperature in the bondline because more heat is lost to the heatsink at this higher temperature. The goal will be to maintain temperatures throughout the bond area to within $\pm 6°C$ of the steady state target temperature $T_{d,ss}$. This is the manufacturers recommended range for the carbon fiber/epoxy prepreg system used for the adherends in this study. The problem will be solved in two parts and then verified experimentally.

1. *Desired power generation* - Find the desired power generation profile $P_d(x, y)$ in the bond area that will produce a uniform temperature $T_{d,ss}$ in the bond area. This will be done empirically since using material properties can be proprietary and difficult to obtain, and therefore an analytical approach is often not practical for real-world applications.

2. *Multi-zone heater design* - Design an embedded heater through simulation that will

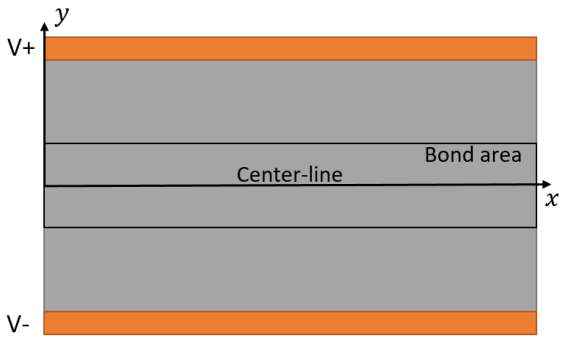


Figure 3.3: A heater of uniform power generation with bond area and center-line shown. Copper tape is used to put tabs on the top and bottom to distribute voltage.

minimize the maximum temperature error in the bondline,

$$\min(|T_{d,ss} - T(t,x,y)|_\infty). \tag{3.1}$$

3.3 Determining desired power within bondline

3.3.1 Model

The initial step in designing a heater and control scheme is to find the power generation profile $P_d(x,y)$ that will produce uniform temperatures in the bond area at the steady state temperature $T_{d,ss}$. To determine the desired power profile, an experiment was run that is described later in Chapter 4, where a known amount of power was applied with a heater of uniform power generation. With this uniform input at the bondline, the temperature was measured at points along the length of the bond center-line. This experimental data can be used to fit a model at the measurement locations, which is then extrapolated to predict temperatures at more points along the bondline. The resulting model is then inverted to find the desired power profile $P_d(x,y)$ within the bond area. This desired profile, which is found

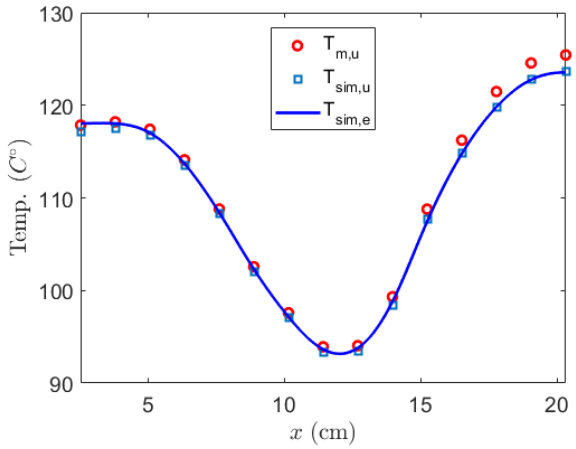


Figure 3.4: Temperature profile along the center-line of the bond area with heatsink under center, at the end of a step response with a uniform embedded resistive heater as in Fig. 3.3. The figure includes measured temperatures $T_{m,u}$, simulated temperatures $T_{sim,u}$, and the simulated temperatures extrapolated through the entire bonding area $T_{sim,e}$.

along the center-line of the bond area where temperatures are measured, is then extended to the rest of the bond area (along the y-direction). This is visualized in Fig. 3.3, with a line through the center of the embedded heater and the bonded region.

A model is developed in this section to describe the transient temperatures within the bond area of a single-lap joint as shown in Fig. 3.2. This will be represented with a 2-dimensional lumped heat transfer equation, where there is some mass (node x_k) that is being heated with energy $\dot{q}_{in}$, and losing heat through conduction. It is assumed that the conduction coefficients in the vertical (k_z) and horizontal (k_x and k_y) directions are independent, and that there is no heat flowing in the y-direction. By subtracting the ambient temperature T_∞ from the measurements, the system can be described in terms of the relative

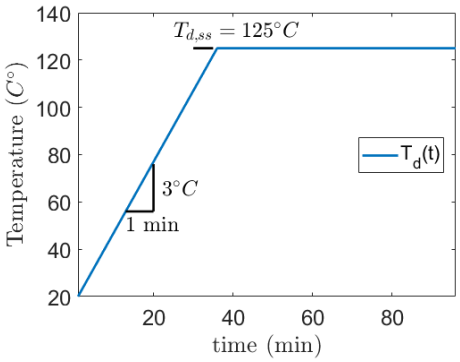


Figure 3.5: Ramp and hold temperature profile $T_d(t)$ recommended for the adhesive used in this study. The ramp rate is 3 ° C per minute, and the hold temperature $T_{d,ss}$ is 125 ° C.

temperature $T(t,x,0) = T^*(t,x,0) - T_\infty$ as

$$\begin{aligned}\rho C p V_{xyz}\frac{dT(t,x_k,0)}{dt} &= -2k_z\frac{A_{xy}}{L_z}T(t,x_k,0)\\ &- k_x\frac{A_{yz}}{L_x}(T(t,x_k,0) - T(t,x_{k+1},0))\\ &- k_x\frac{A_{yz}}{L_x}(T(t,x_k,0) - T(t,x_{k-1},0)) + A_{xy}\dot{q}_{in},\end{aligned} \tag{3.2}$$

where ρ is the material density, C_p is the specific heat capacity, V_{xyz} is the element volume, A indicates the area of one side of the node, and L is the length between nodes. The diagram in Fig. 3.6 helps to illustrate Eq. 3.2. This schematic shows that heat is lost through conduction to the left and right with conduction coefficient k_x to temperatures $T(x_{k+1},0)$ and $T(x_{k-1},0)$. Conduction to the top and bottom (z) goes to T_∞ with conduction coefficient K_z. Heat is added from the heater as $\dot{q}_{in}$. This system is meant to represent the thermal system of adherends heated with an embedded heater, where heat is lost above and below to the environment, and is also transferred horizontally near the bondline.

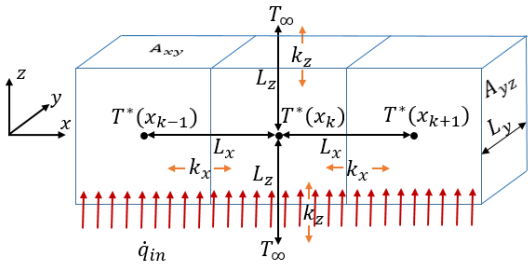


Figure 3.6: Lumped capacitance model diagram for heat flow along the center-line of the bond area in a single-lap joint as shown in Fig. 3.2. A typical element (x_k) is shown with heat conducted to neighboring elements (x_{k+1} and x_{k-1}), to the surrounding structure (at T_∞), and with heat added by the heater ($\dot{q}_{in}$). The temperatures $T^*(t, x, y)$ are denoted here as $T^*(x)$ for simplicity.

The model in Eq. 3.2 can be simplified to general parameters as

$$\begin{aligned}\gamma\dot{T}(t, x_k, 0) = &-\beta_k T(t, x_k, 0) - \alpha(T(t, x_k, 0) - T(t, x_{k+1}, 0)) \\ &- \alpha(T(t, x_k, 0) - T(t, x_{k-1}, 0)) + \dot{Q}_{in},\end{aligned} \tag{3.3}$$

with parameters γ, β, α, and the heat added $\dot{Q}_{in}$ used in place of $\dot{q}_{in}$ ($A_{xy}\ V^2/(\Omega\ \cdot\ m^2)$), with units of V^2/m^2. The area A_{xy} and resistance Ω of the heater are divided through the rest of the equation and represented in the other parameters.

3.3.2 Solving for parameters α and β

A solution will be found by putting the model equations into the form

$$\dot{Q}_{in} = \gamma\dot{T}_i + \beta_i T_i + \alpha(T_i - T_{i-1}) + \alpha(T_i - T_{i+1}). \tag{3.4}$$

Putting this into matrix form for a 5 node system we get

$$\underbrace{\begin{bmatrix} T_1 & 0 & 0 & 0 & 0 & T_1 - T_2 & \dot{T}_1 \\ 0 & T_2 & 0 & 0 & 0 & 2T_2 - T_1 - T_3 & \dot{T}_2 \\ 0 & 0 & T_3 & 0 & 0 & 2T_3 - T_2 - T_4 & \dot{T}_3 \\ 0 & 0 & 0 & T_4 & 0 & 2T_4 - T_3 - T_5 & \dot{T}_4 \\ 0 & 0 & 0 & 0 & T_5 & T_5 - T_4 & \dot{T}_5 \end{bmatrix}}_{\text{A}} \underbrace{\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \alpha \\ \gamma \end{bmatrix}}_{\text{x}} = \underbrace{\begin{bmatrix} \dot{Q}_{in} \\ \dot{Q}_{in} \\ \dot{Q}_{in} \\ \dot{Q}_{in} \\ \dot{Q}_{in} \end{bmatrix}}_{\text{b}}. \tag{3.5}$$

We will call this $Ax = b$ for simplicity. Now, to solve for the parameters, we will try to minimize

$$\min |W \odot (b - Ax)|_2, \quad \text{where } \forall x > 0. \tag{3.6}$$

The W vector will be used for weighting the data so that temperatures closer to the end target temperature of $125°C$ will have more weight. The $\odot$ symbol in the equation above represents element-wise multiplication. This weighting is done to minimize the impact of non-linearities in the system.

3.3.3 Choosing a weighting matrix

The choice of a weighting matrix will determine how well the model fits within a temperature range. Having a higher weighting at temperatures closer to the steady state temperature of 125 ° will mean the model will fit better at steady state. As a majority of adhesive curing happens near and at steady state, this is the temperature range that will be targeted by weighting.

This weighting matrix will be chosen to be

$$W = \begin{bmatrix} W_m \\ W_m \\ W_m \\ W_m \\ W_m \end{bmatrix} \tag{3.7}$$

where W_m is linear so that at ambient temperature, W_m is zero and at the maximum temperature T_{max}, W_m is 1. That is

$$W_m = \frac{t}{t_{final} - t_{initial}}, \tag{3.8}$$

where t represents time at a sample instant.

3.3.4 Going from parameters back to model

After an experiment has been run and parameters from Eq. 3.5 have been fitted, it is useful to reform the equation as a system of first order equations. For the case where γ and α are constant, the system looks like

$$\begin{bmatrix} \dot{T}_1 \\ \dot{T}_2 \\ \dot{T}_3 \\ \dot{T}_4 \\ \dot{T}_5 \end{bmatrix} = \begin{bmatrix} \frac{-\beta_1-\alpha}{\gamma} & \frac{\alpha}{\gamma} & 0 & 0 & 0 \\ \frac{\alpha}{\gamma} & \frac{-\beta_2-2\alpha}{\gamma} & \frac{\alpha}{\gamma} & 0 & 0 \\ 0 & \frac{\alpha}{\gamma} & \frac{-\beta_3-2\alpha}{\gamma} & \frac{\alpha}{\gamma} & 0 \\ 0 & 0 & \frac{\alpha}{\gamma} & \frac{-\beta_4-2\alpha}{\gamma} & \frac{\alpha}{\gamma} \\ 0 & 0 & 0 & \frac{\alpha}{\gamma} & \frac{-\beta_5-\alpha}{\gamma} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} + \frac{1}{\gamma} \begin{bmatrix} \dot{Q}_{in} \\ \dot{Q}_{in} \\ \dot{Q}_{in} \\ \dot{Q}_{in} \\ \dot{Q}_{in} \end{bmatrix}. \tag{3.9}$$

3.3.5 A temperature varying β

It is possible that the model can be improved by making β function of temperature, such as

$$f_\beta(T_i) = \zeta T_i + \eta. \tag{3.10}$$

This then makes the matrix equation

$$\underbrace{\begin{bmatrix} T_1^2 & T_1 & 0 & 0 & \cdots & 0 & T_1 - T_2 & \dot{T}_1 \\ 0 & 0 & T_2^2 & T_2 & \cdots & 0 & 2T_2 - T_1 - T_3 & \dot{T}_2 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 2T_3 - T_2 - T_4 & \dot{T}_3 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 2T_4 - T_3 - T_5 & \dot{T}_4 \\ 0 & 0 & 0 & 0 & \cdots & T_5 & T_5 - T_4 & \dot{T}_5 \end{bmatrix}}_{\mathrm{A}} \underbrace{\begin{bmatrix} \zeta_1 \\ \eta_1 \\ \zeta_2 \\ \eta_2 \\ \vdots \\ \eta_5 \\ \alpha \\ \gamma \end{bmatrix}}_{\mathrm{x}} = \underbrace{\begin{bmatrix} \dot{Q}_{in} \\ \dot{Q}_{in} \\ \dot{Q}_{in} \\ \dot{Q}_{in} \\ \dot{Q}_{in} \end{bmatrix}}_{\mathrm{b}}. \tag{3.11}$$

For the system in question, a single-lap joint with a heat sink under the center region, it was found that adding these extra parameters cause the system to be overdetermined. For this work work, the best model was determined to be Eq. 3.3.

3.3.6 Refining the mesh

The parameters for the model described in Eq. 3.3 are found with the least squares method using a limited number of thermocouples. Using this information, the model is then refined so that it can predict any point along the bond center-line. This is done by taking the nodal Eq. 3.3 with node spacing L_x and adjusting the parameters to match a smaller node spacing $L_{\bar{x}}$ in the x-direction as shown in Fig. 3.7. This can be described with nodes at locations $\bar{x}_k$

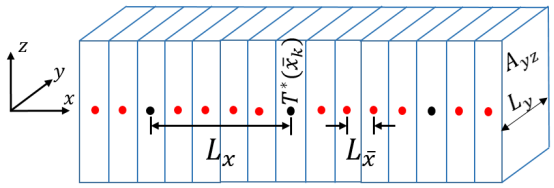


Figure 3.7: Lumped capacitance model diagram with reduced node spacing $L_{\bar{x}}$ as opposed to L_x as in Fig. 3.6.

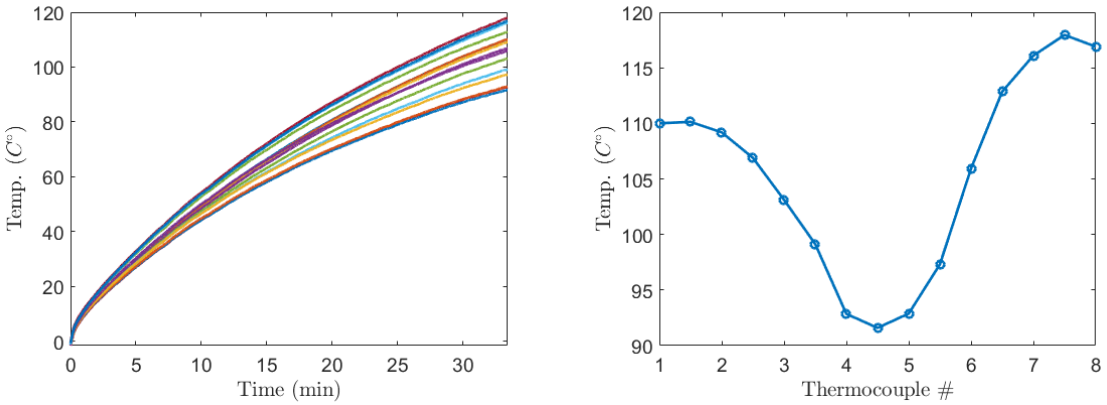
as

$$\begin{aligned}\bar{\gamma}\dot{T}(t,\bar{x}_k,0) = &-\bar{\beta}T(t,\bar{x}_k,0)\\ &-\bar{\alpha}(T(t,\bar{x}_k,0)-T(t,\bar{x}_{k+1},0))\\ &-\bar{\alpha}(T(t,\bar{x}_k,0)-T(t,\bar{x}_{k-1},0))+\bar{\dot{Q}}_{in}.\end{aligned} \tag{3.12}$$

This equation was used to predict the temperature at nodes in-between the measured temperatures as shown in Fig. 3.4 (see solid line $T_{sim,e}$). The scaling for each parameter is summarized in Table 3.1.

Table 3.1: Summary of parameter scaling to get a mesh refined model.

Parameter	Physical meaning	Scaling to get refined parameter
γ	$\rho C_p V_{xyz}$	$\bar{\gamma} = \gamma(L_{x,r}/L_x)$
α	$k_x \frac{A_{yz}}{L_x}$	$\bar{\alpha} = \alpha(L_x/L_{x,r})$
β	$k_y \frac{A_{xy}}{L_y}$	$\bar{\beta} = \beta(L_{x,r}/L_x)$
$\dot{Q}_{in}$	$A_{xy}\dot{q}_{in}$	$\bar{\dot{Q}}_{in} = \dot{Q}_{in}(L_{x,r}/L_x)$



(a) Full temperature data for experiment with a uniform heater and heatsink.

(b) Experimental data for uniform heater with heatsink. This plot shows the temperature at each thermocouple at the end of the experiment at time 33.4 minutes.

Figure 3.8: Experiment run with uniform embedded heater and heatsink under middle to steady-state of $125°C$.

3.3.7 Example system

The proposed model will be used to fit a system of 15 thermocouples, with a heatsink under the center 3 thermocouples and heated with an embedded heater generating uniform power. The temperatures for all 15 thermocouples for the entire experiment can be seen in Fig. 3.8a. The profile of the 15 thermocouples at the end of the experiment can be seen in Fig. 3.8b.

Using the experimental data, and model described where α and γ are constant, the parameters were calculated. The value of the β parameters can be seen in Fig.3.9. From the shape of these parameters, we can use to intuition to see some the model is picking up some key elements from the experiment. Towards the center of the experiment, β increases. We would expect this as there is more heat loss in this area due to the heat sink.

To verify the experiment, the parameters were plugged back into a simulation, and run using the same input as the experiment and over the same length of time. The results of this

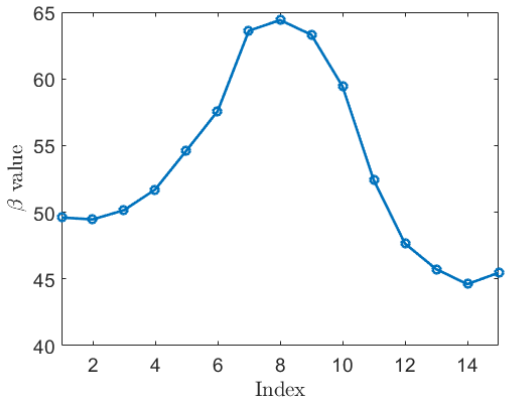
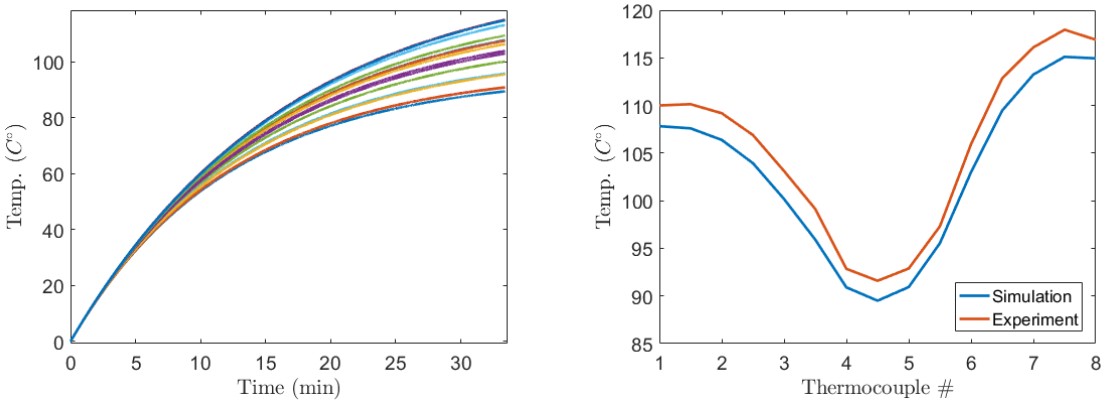


Figure 3.9: Parameter fit for β values.



(a) Full temperature data for a simulation with a uniform heater and heatsink.

(b) Comparison of steady-state temperature between experiment and simulation.

Figure 3.10: Simulation run to emulate a uniform embedded heater and heatsink under middle to steady-state of $127°C$.

can be seen in Fig. 3.10. These results are for the final steady-state values across the center of the bondline. The simulation values are slightly lower than the experimental values. This is due to the nonlinear nature of the thermal system through the range of temperatures.

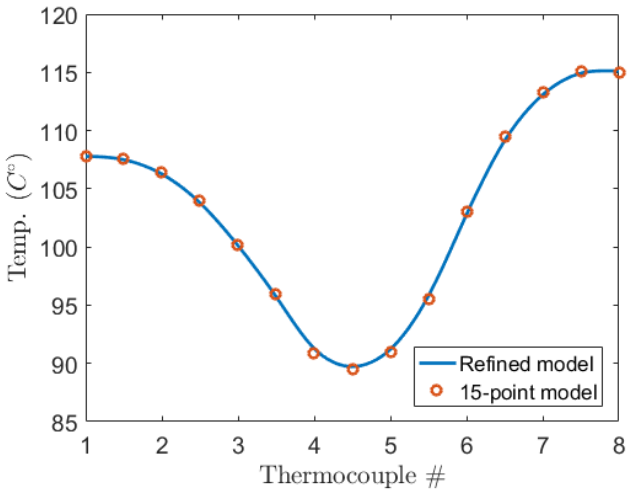


Figure 3.11: Comparison at time 33.4 minutes between original 15 point simulation and refined 100 point simulation.

3.3.8 Example system refined

Using the same example model and the parameter modifications described in Section 3.3.6, a refined model of 100 points was generated. The results of this can be compared with the original 15 node model by overlaying the temperature at each thermocouple at a specific time instant. This was done at the end point in time at 33.4 minutes. The results of this can be seen in Fig. 3.11. As expected, the refined model goes through the 15 point model.

3.3.9 Number of thermocouples for fit

Preliminary experiments were done using 15 thermocouples evenly spaced within the bond centerline. It is possible that less or more thermocouples are needed to produce an accurate model. An accurate model is demonstrated when adding more thermocouples has a negligible effect. To analyze this, both measured data and manufactured data will be examined for validation.

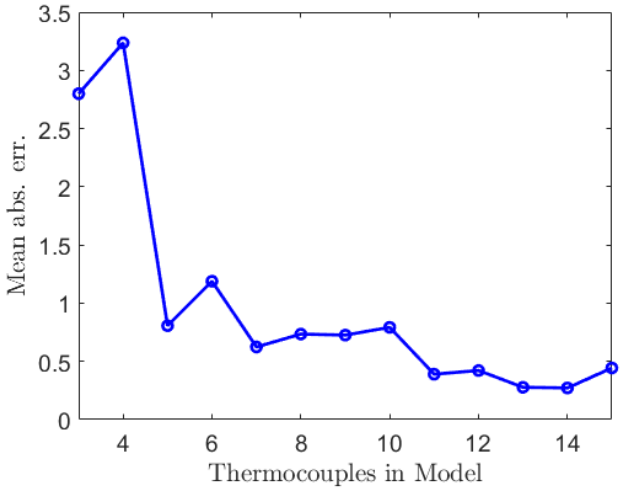


Figure 3.12: Mean absolute error between refined model and 15 measured points relative to the number of thermocouples used in the model.

Measured data

To see how the model improves as more thermocouples are added, models were created using 3 to 15 thermocouples. For the models with 3, 4, 5, 8, and 15 thermocouples, actual thermocouple measurements were used. For the other integers where measurements were not taken at even spacing, data was manufactured by interpolating measured temperatures. These models were refined to 500 points. Then each model was compared with measured data at the 15 points where thermocouple data was taken. These results can be seen in 3.12. The mean error drops significantly as more thermocouples are added to the model up until 8 thermcouples, then only a marginal gain is found beyond that. This implies that using 15 thermocouples is sufficient to accurately portray the temperatures in the bond area.

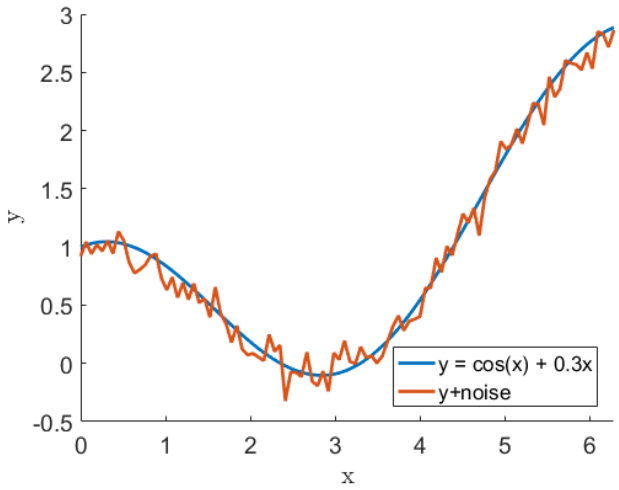


Figure 3.13: Manufactured data to be used to validate the number of points to get a good model.

Manufactured data

To verify the previous findings that the number of thermocouples to get an accurate model in Fig. 3.12 is reasonable, a similar process was done on manufactured data. The manufactured data comes from the equation

$$y = cos(x) + 0.3x, \tag{3.13}$$

which is similar in form to the shape of the measured data as shown in Fig. 3.11. Added to this data was added white Gaussian noise, and this manufactured data can be seen in Fig. 3.13.

A model was fit to data using equally space points, always including the end points. The fit was obtained using MATLABS machine learning function FITRGP. This was done 100 times, and the average RMS error can be seen in Fig. 3.14. From this plot we see that a fairly good fit can be found after about 5 data points. This matches what was seen with experimentally measured data.

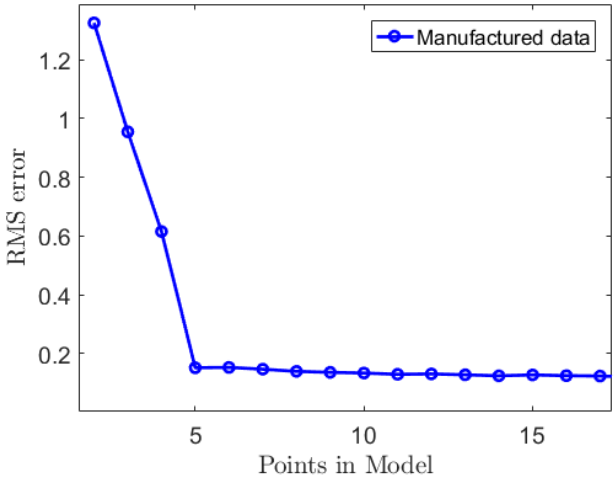


Figure 3.14: RMS error between manufactured data and model based on a subset of points.

3.4 Inverting model to get optimal input

Finding the optimal input at each point along the bond centerline will inform heater design. As the model changes as the system heats up and is held at steady state, the primary focus of design will be at steady-state. This is where most of the adhesive curing happens. There are two desired input profiles we can construct. One is to do a simple inversion of the input and output, assuming a simple gain system. The second is to invert the model that was generated in section 3.3.1.

3.4.1 Input from inverting simple gain system

To determine the desired average power profile, data can be taken from an experiment where a known amount of power is applied with a single-zone heater. A single zone heater will theoretically generate the same amount of power everywhere on the heater. The system can described with a simple gain in x as

$$T(x) = G(x)P(x), \tag{3.14}$$

where T is the measured output in temperature, P is the input of power, and G is the overall system mapping. This system mapping G is found by measuring T and P at points along x on the center-line of the bond area, and then approximating this with a polynomial function. Using this method then the desired input can be found as

$$P_{d,s}(x) = G^{-1}(x)T(x). \tag{3.15}$$

Here the label $P_{d,s}$ implied the desired input, with an s to indicate the simplied approach.

3.4.2 Input from inverting model from section 3.3.1

The desired power profile $P_d(x, 0)$ at the bond center-line (shown in Fig. 3.2 as the x-axis), needed to achieve the steady state target temperature $T_{d,ss}$, can be found by first re-framing the model in Eq. 3.3 into a system of first order equations,

$$\dot{T}(t, x, 0) = AT(t, x, 0) + BP(t, x, 0), \tag{3.16}$$

$$\hat{T}(t, x, 0) = CT(t, x, 0), \tag{3.17}$$

where A, B, and C come from empirical model parameters, $T(t, x, 0)$ is the predicted temperatures, $\hat{T}(t, x, 0)$ is the output temperatures, and $P(t, x, 0)$ is the power added into the bond center-line. The temperatures $\hat{T}(t, x)$ and $T(t, x)$ are assumed to be the optimal target trajectory $T_d(t)$, which is a uniform temperature throughout the bond area during a ramp and hold profile, as shown in Fig. 3.5. Since the steady state temperature $T_{d,ss}$ will be used for heater design, the steady state desired profile can be found independent of time t. By taking the derivative of Eq. 3.17 and combining with Eq. 3.16, the desired profile can be

solved as

$$P_d(t, x, 0) = (CB)^{-1}(\dot{T}_d(t) - CAT_d(t)), \tag{3.18}$$

where $T_d(t) = T_{d,ss}$, and $\dot{T}_d(t) = 0$, so that the desired profile at steady state is

$$P_d(x, 0) = (CB)^{-1}(-CAT_{d,ss}). \tag{3.19}$$

This desired profile is extended through the rest of the bond area, and is constant in the y-dimension.

3.4.3 Comparing the two approaches

The solutions to achieve temperature uniformity in the bondline using the two approaches described can be seen in Fig. 3.15. There are a few differences that are worth remarking on. There is a mean offset between the two solutions, the modeled inverse has a larger "hump", and the simple inverse is "smoother". It is speculation to say one approach is better than the other, but as the modeled inverse accounts for some flows along the bondline in the lateral direction (x), it seems likely that it is more accurate.

3.4.4 Effect of β on modeled inverse solution

It can be seen that the shape of the modeled inverse solution is identical to the shape of β, which is shown in Fig. 3.16. The reason for this is due to the fact that when the temperature is uniform along the bond centerline, there is no heat transfer along the direction of x, so the model simplifies to

$$\gamma\dot{T}_i = -\beta_i T_i + \dot{Q}_{in}. \tag{3.20}$$

In this case it would be expected that the inverse input shape would match β.

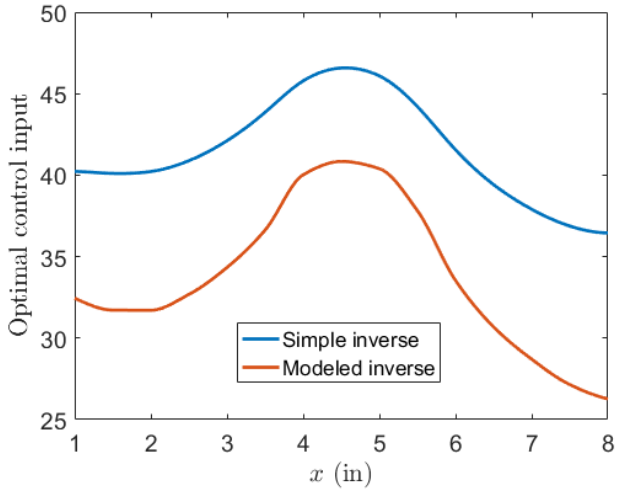


Figure 3.15: Optimal inputs to achieve uniform temperature at bond centerline using a simple inverse and modeled inverse.

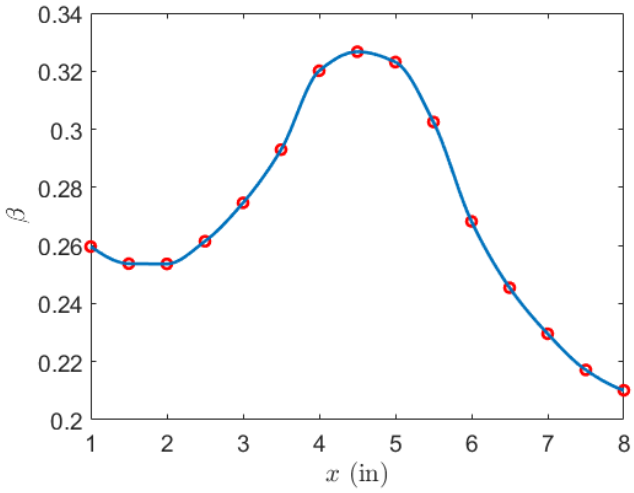


Figure 3.16: Value of β along centerline when refined from 15 to 500 points.

3.5 FDM model of a tabbed heater

A finite difference method (FDM) model will be used to calculate the power generated in a 2-dimensional resistive heater. This problem is defined by the Laplace equation (3.21), where voltage (V) is a function of displacements in the x and y directions. From this solution, the power can be found from the gradient of the voltage squared.

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0 \tag{3.21}$$

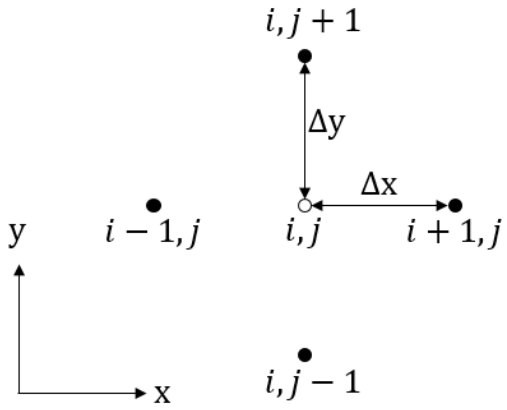


Figure 3.17: Square grid mesh to be used in FDM analysis.

A grid of square elements will be used to define the mesh, which can be seen in Fig. 3.17. The grid nodes are labeled such that

$$i + 1 = i + \Delta x \tag{3.22}$$

$$j + 1 = j + \Delta j, \tag{3.23}$$

where it is also chosen that $\Delta x = \Delta y$. Using this grid, the potential at points $i + 1$ and

$i-1$ is defined as

$$V_{i-1,j} = u_{i,j} - V_x \Delta x + \frac{1}{2} V_{xx} \Delta x^2, \tag{3.24}$$

$$V_{i+1,j} = u_{i,j} + V_x \Delta x + \frac{1}{2} V_{xx} \Delta x^2. \tag{3.25}$$

Subtracting these equations and solving for the first derivative V_x gives

$$V_x = \frac{V_{i+1,j} - V_{i-1,j}}{2\Delta x}. \tag{3.26}$$

Adding Eqs. 3.24 and 3.25 and solving for the second derivative V_{xx} gives

$$V_{xx} = \frac{V_{i+1,j} + V_{i-1,j} - 2V_{i,j}}{\Delta x^2}. \tag{3.27}$$

This equation can now be plugged into the Laplace equation 3.21 (assuming a similar solution in y).

This gives the solution for each point in the grid as

$$V_{i,j} = \frac{u_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1}}{4}. \tag{3.28}$$

This essentially means that each grid point is the average of the points around it.

This solution 3.28 does not work however at the boundaries where there are no nodes on one or more sides. This requires boundary condition definitions. For the case of resistive heaters, the boundary assumption is that there is no loss of potential through the edge, so at the edges

$$V_x = 0. \tag{3.29}$$

Using this and Eq. 3.26, gives

$$\frac{V_{i+1,j} - V_{i-1,j}}{2\Delta x} = 0 \tag{3.30}$$

$$V_{i+1,j} = V_{i-1,j}. \tag{3.31}$$

For the case then where $V_{i+1,j}$ is off the edge of the heater and not a valid point

$$V_{i,j} = \frac{2V_{i-1,j} + V_{i,j+1} + V_{i,j-1}}{4}. \tag{3.32}$$

A similar argument works for the other edges.

One other boundary condition that needs to be addressed is the boundary where copper tabs are located on the heater. At these locations, the copper is used to transmit voltage to the heater evenly across the tab. As the copper has a significantly lower resistivity than the carbon-fiber heater, it will be assumed that at these edge points, the potential V is equal in the region of the tab. This approach is visualized in Fig. 3.18.

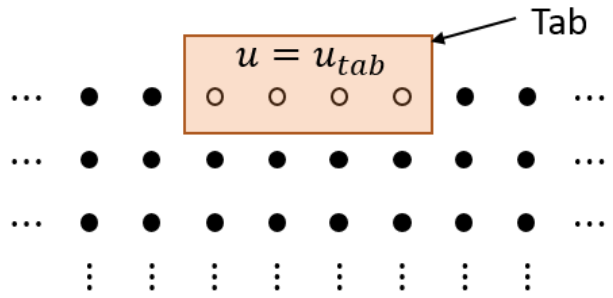


Figure 3.18: Grid where tab is present has equal potential along tab.

3.6 FDM vs. Analytical model in MATLAB

The current model has been created in MATLAB using the finite difference method with square elements. To validate this model, an analytical solution to the Laplace equation (3.21) can be found with the boundary conditions

$$V(0, y) = g(y)$$
$$V(x, 0) = 0$$
$$V(L, y) = 0$$
$$V(x, H) = 0.$$

These boundary conditions are illustrated in Fig. 3.19.

To simplify the solution, which will be shown later, $g(y)$ is chosen as

$$g(y) = sin(2\pi/H). \tag{3.33}$$

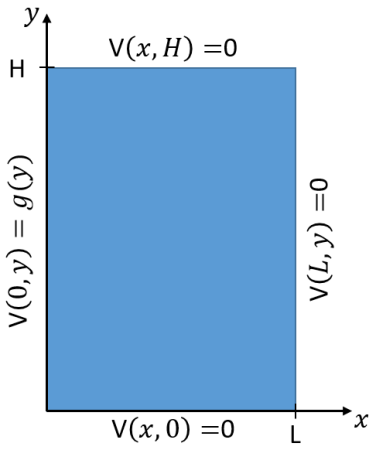


Figure 3.19: Boundary conditions for analytical and finite difference method solutions.

The analytical solution can be found in many mathematics textbooks, and specifically in [24, pages 558–560]. The solution to this PDE is found by first assuming the solution is a combination of separable functions,

$$V(x,y) = h(x)q(y). \tag{3.34}$$

Plugging this back into 3.21, and simplifying, this becomes

$$\frac{d^2h}{dx^2} - \lambda h = 0, \quad and \quad \frac{d^2q}{dy^2} + \lambda q = 0. \tag{3.35}$$

Now solutions for these two differential equations can be found independently, then combined. Given two boundary conditions of zero in y, this is a good place to start, and solving for $q(y)$ gives a solution of the form

$$q(y) = sin(ky), \tag{3.36}$$

where when plugging this into 3.35 it is found that

$$k = \frac{n\pi}{H}, \quad and \quad \lambda = \left(\frac{n\pi}{H}\right)^2. \tag{3.37}$$

Now looking at $h(x)$, the solution can be assumed to have the form of

$$h(x) = c_1 cosh\left(\frac{n\pi(x-L)}{H}\right) + c_2 sinh\left(\frac{n\pi(x-L)}{H}\right). \tag{3.38}$$

Using the boundary condition of $h(L) = 0$, it is found that $c_1 = 0$, which means

$$h(x) = c_2 sinh\left(\frac{n\pi(x-L)}{H}\right). \tag{3.39}$$

Putting these solutions together then

$$V(x,y) = c_2\ sinh\left(\frac{n\pi(x-L)}{H}\right) sin\left(\frac{n\pi y}{H}\right). \tag{3.40}$$

Now bringing in the final boundary condition, $u(0,y) = g(y)$, gives

$$V(0,y) = g(y) = c_2\ sinh\left(\frac{n\pi(-L)}{H}\right) sin\left(\frac{n\pi y}{H}\right). \tag{3.41}$$

Since the Fourier series is an odd function, the solution is

$$V_n = B_n sinh\left(\frac{n\pi(x-L)}{H}\right) sin\left(\frac{n\pi y}{H}\right), \tag{3.42}$$

where

$$B_n = \frac{2}{H sinh\left(\frac{n\pi(-L)}{H}\right)} \int_0^H g(y) sin\left(\frac{n\pi y}{H}\right) dy. \tag{3.43}$$

When $g(y)$ is plugged in, all terms go to zero except for $n = 2$, so that

$$V(x,y) = \frac{1}{sinh\left(\frac{2\pi(-L)}{H}\right)} sinh\left(\frac{2\pi(x-L)}{H}\right) sin\left(\frac{2\pi y}{H}\right). \tag{3.44}$$

Using this solution, and taking $L = 50$ and $H = 100$, the solution can be found in MATLAB and seen in Fig. 3.20. This choice of L and H was chosen to match closely the ratio used in experiments for bonding single lap joints.

This MATLAB solution can then be used to validate the finite difference method model. The advantage of the finite difference model is that it can calculate a solution for any boundary conditions without a new analytical solution. To compare the two solutions (analytical and finite difference method in MATLAB), the FDM model was found within an error tolerance of $1e-9$, and the solutions were compared with increasing grid size.

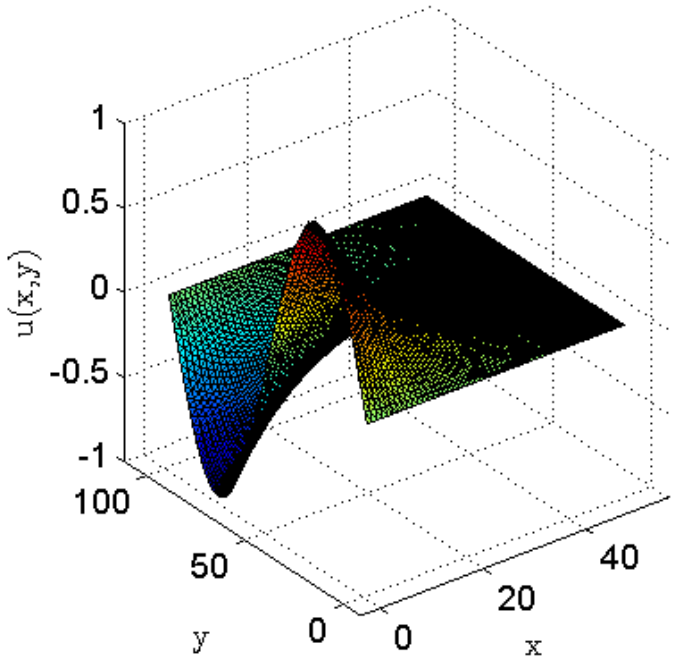


Figure 3.20: Analytical solution to Laplace Equation found in MATLAB.

The results of the comparison between analytical and finite difference methods in MATLAB can be seen in Fig. 3.21. In this figure, the maximum error between the FDM and analytical solutions is decreasing as the grid size increases, and at 20,000 nodes, the error is less than .01 percent of the RMS of the solution.

3.7 FDM vs. PDE tool solution in MATLAB

MATLAB has a very easy to use PDE tool for simple geometries and boundary conditions. Using this tool, the solution in Fig. 3.22a was found. To see if the FDM solution converges to this solution, the RMS of the difference between the solutions can be examined by looking at how this quantify changes as the iterations in the FDM solution are increased.

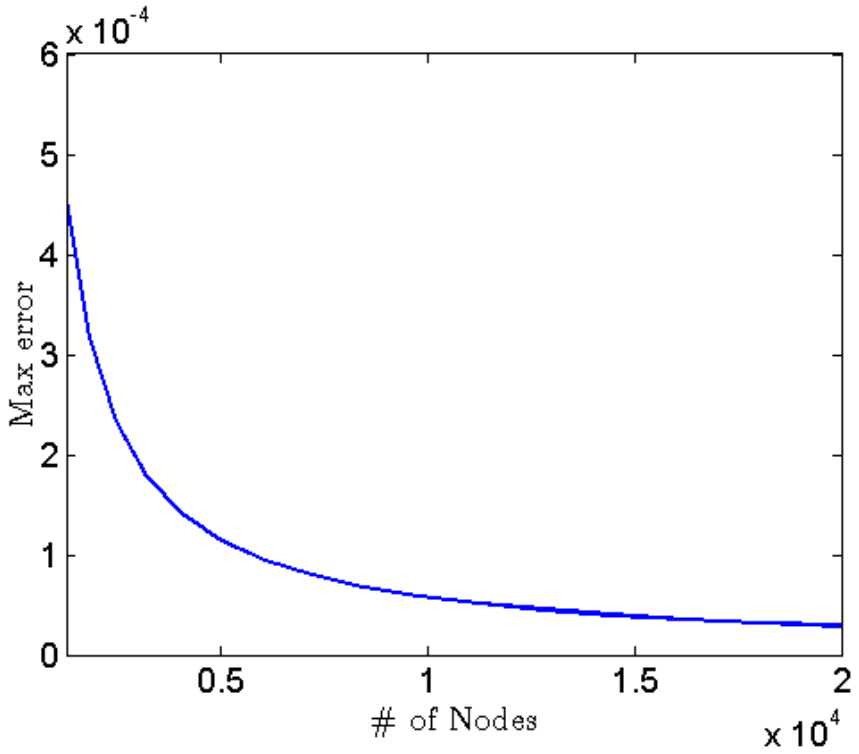


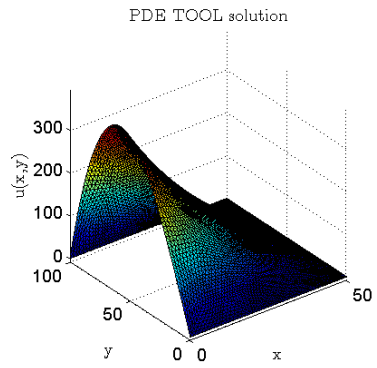
Figure 3.21: Analytical solution to Laplace Equation found in MATLAB.

Mathematically, this looks like

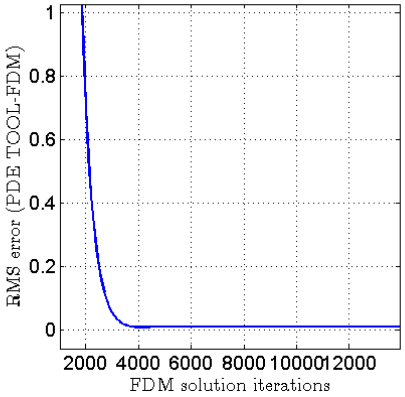
$$e_{rms} = \sqrt{\frac{1}{n}\left(e_1^2 + e_2^2 + \cdots + e_n^2\right)}, \tag{3.45}$$

where e_i is the error at point i between the PDE tool solution and the FDM solution, and n is the total number of points in the solutions. Since the PDE tool does not use a simple square mesh, the FDM solution was interpolated onto the same grid as the PDE tool. By this metric, it can be seen that the two solutions are converging with increasing FDM iterations in Fig. 3.22b.

Fig. 3.23 shows the difference between these two solutions plotted in 3D, and the error is small (about 4 orders of magnitude smaller than the solution), with error that looks more



(a) Solution from PDE tool in MATLAB.



(b) Convergence in RMS between PDE tool and FDM solutions.

Figure 3.22: PDE tool solution and convergence of FDM solution to PDE tool solution.

like noise than a continuous function. Therefore, it would seem that the FDM model is accurate and can be used to approximate a solution for our embedded heaters.

3.8 Analytical solutions to multiple and tabbed heater approaches

This section will use the model developed here in order to determine how well the multiple heater and tabbed heater approaches are able to approximate the desired power profile defined in this section.

3.8.1 Multiple heater solution

The power generated in a heater with a single tab on each side is uniform throughout the heater. This is because power generated is proportional to the voltage gradient squared, and the voltage gradient in a single tabbed heater is uniform across is length. With this in mind, the cost function to minimize is

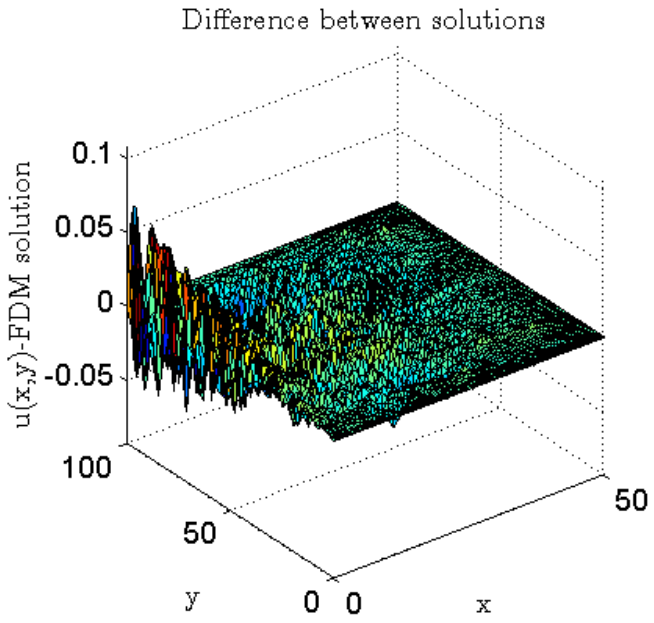


Figure 3.23: Difference between PDE tool and FDM solutions.

$$J(\theta) = \|P_d - P_\theta\|_2, \tag{3.46}$$

where P_d is the desired power profile, and P_θ is the optimal solution given a set of design parameters θ. These design parameters are the widths of the individual heaters x_1, x_2, x_3, etc. as shown in Fig. 3.24. The total length of the heater is prescribed, so there is one less parameter than the number of heaters.

This cost function was minimized with the MATLAB function FMINCON. The results for 5 heaters is shown in Fig. 3.25. As would be expected, in the regions where the desired profile has a sharper gradients, heater are more narrow, and the opposite is true for flatter regions of desired power.

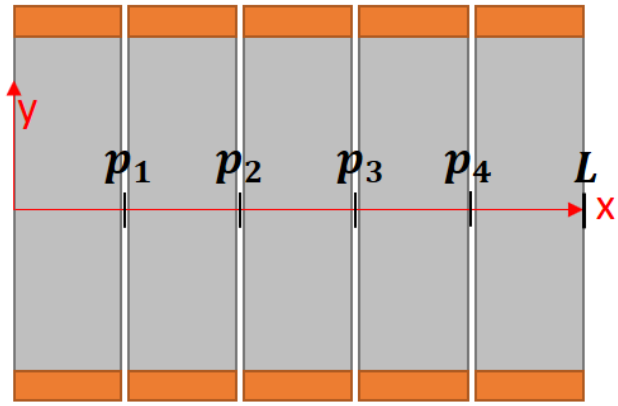


Figure 3.24: Schematic of a multiple heater configuration and design parameters.

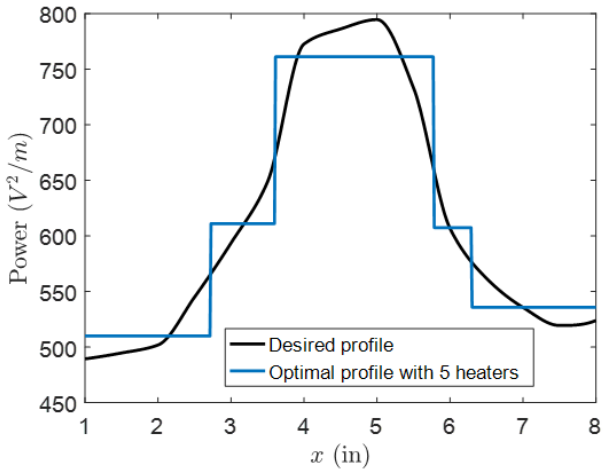


Figure 3.25: Solution to Eq. 3.46 for 5 heaters.

3.8.2 Applying solution to tabbed heater

The model for a tabbed heater was developed in section 3.5. Using this model, it is possible to find an optimal solution for the cost equation using Eq. 3.46, except h_θ is the result of the model. This approach, while possible, is computationally expensive. Instead, an alternate approach is tries here. The solutions found using the multiple heater approach will also

be applied to the tabbed heater solutions. The results of this for a 6 tabbed heater can be seen in Fig. 3.26. One clear advantage of this approach is that it removed the sharp discontinuities present in the multiple heater approach, as evident in Fig. 3.25.

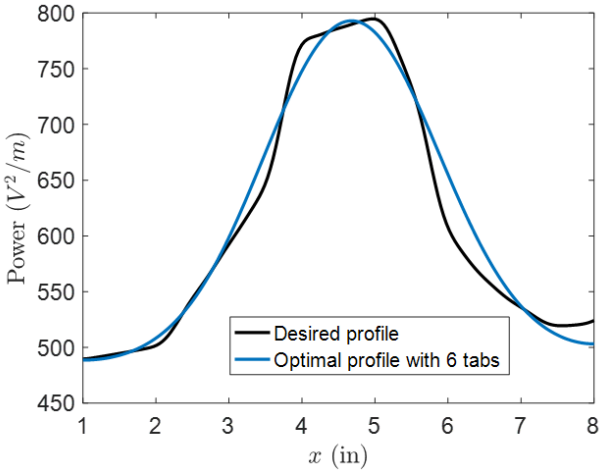


Figure 3.26: Solution to Eq. 3.46 for 6 heaters.

3.8.3 Comparing same solution applied to both multiple and tabbed heaters

To compare the analytical results between the multiple heater and tabbed heater approaches, it is helpful to compare how well they are able to reduce the maximum error in temperature according to the model derived in section 3.3.1. A plot comparing the two approaches can be seen in Fig. 3.27. This plot show the results for an increasing number of tabs.

This approach does seem to work for this specific example, but there is no general guarantee for this approach. As such, a method for finding an optimal heater design for a tabbed heater will also be examined.

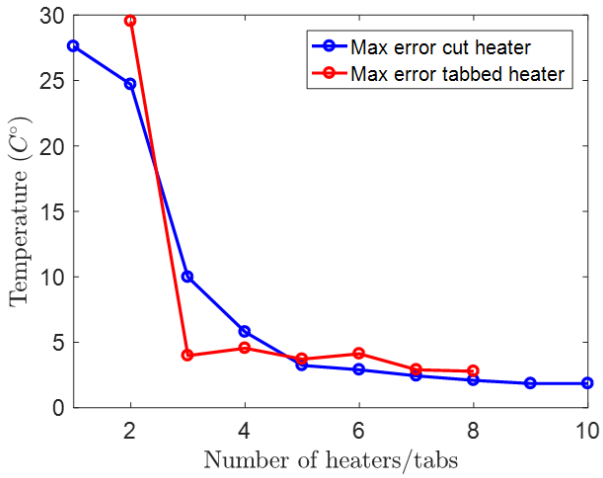


Figure 3.27: Comparison of temperature error between multiple heater and tabbed heaters with increasing number of tabs. Heater and tab locations are in the same locations in each line.

3.8.4 Tabbed heater solution

The goal of designing a heater with multiple tabs is to minimize the temperature variation in the bond area. To do this, a model that can predict the power generation of a heater with a specific tab placement is needed. A 2-dimensional steady state voltage solution for a tabbed heater was developed using a finite difference method (FDM) model [39] to solve the Laplace Equation

$$\Delta V = \frac{\partial V^2}{\partial^2 x} + \frac{\partial V^2}{\partial^2 y} = 0. \tag{3.47}$$

The inputs to this model can be seen in Table 3.2, where the heater width and length were determined by the geometry of the single-lap joint experimental setup. It was found experimentally that smaller gaps between tabs can cause an electrical short, while large gap sizes in simulation reduced the effectiveness of a heater to target power in the bond area, so an effective gap size was chosen to be 0.635 cm.

A FDM simulation calculates the power generated $P(\theta_j, c_i)$ for a set of spatial parameters

Table 3.2: FDM model design parameters for heater with multiple tabs as in Fig. 3.1

Parameter	Value(s)
Heater width in x (cm)	17.78
Heater length in y (cm)	12.7
Tab gap width (cm)	0.635
Grid size (mm)	1.27
Gap locations θ (cm)	x_1, x_2, x_3, $\cdots$
Tabs activated (bin)	c_i

$\theta = x_1, x_2, \cdots, x_{n-1}$, where n is the number of tabs as indicated in Fig. 3.1, and a tab activation configuration c, which is a binary number where each digit indicates whether a tab is "on" (attached to a voltage), or "off" (floating voltage). For example, one tab activation configuration is $c = [1\ 0\ 0\ 0\ 0]$, which indicates a 5 tabbed heater with the first tab on, and the rest off. A multiple tabbed heater power profile P_{sim} is found as a linear combination of simulations of c_i configurations, so that

$$\begin{aligned} P_{sim}(\zeta, \theta_j) = \zeta_1 P(\theta_j, c_1) + \zeta_2 P(\theta_j, c_2) \\ + \zeta_3 P(\theta_j, c_3) + \cdots + \zeta_m P(\theta_j, c_m). \end{aligned} \tag{3.48}$$

Here $m = 2^n$ is the total number of configurations c_i, and ζ_i indicates the amount of time configuration c_i is activated, and is constrained to a normalized time period of length 1, so that the sum of all configurations should add to 1, and no individual time ζ_i is less than 0 or greater than 1,

$$\sum_{i=1}^{m} \zeta_i = 1, \quad \text{and} \quad 0 \leq \zeta_i \leq 1. \tag{3.49}$$

Using the FDM simulation, the optimal power profile P_{sim} and times ζ for given heater

parameters θ_j are found by minimizing the cost function

$$J_P(\zeta, \theta_j) = \|P_d(x, y) - P_{sim}(x, y, \zeta, \theta_j)\|_2 \tag{3.50}$$

with respect to ζ using a discrete proportional-integral (PI) controller to adjust the error in each control zone n (area within the region of a tab, as in Fig. 3.1). For zone n, the parameter η, which indicates the amount of time the tab in zone n is activated, is calculated at step k as

$$\eta_n(k) = \eta_n(k-1) + K_{pd}e_n(k) - K_{id}e_n(k-1), \tag{3.51}$$

where the K_{pd} is a discrete proportional gain, K_{id} is a discrete integral gain, and the error in each zone n is calculated as

$$e_n(k) = P_{d,n}(x, y) - P_{sim,n}(x, y, \zeta(k-1), \theta_j). \tag{3.52}$$

The values of η_n were found by running Eq. 3.51 to convergence, giving the optimal values of η_n, and is represented as the times ζ_{opt} to activate tab configurations c_i. The code for this minimization can be found in Appx. C.10. This minimization gives the optimal ζ,

$$\zeta_{opt} = \arg\min_{\zeta} \; J_p(\zeta, \theta_j). \tag{3.53}$$

An example of an optimized power profile $P_{sim}(x, y_{avg}, \zeta_{opt}, \theta_j)$ for a heater with 5 tabs can be seen in Fig. 3.28, along with the desired profile $P_d(x, 0)$. The power profiile $P_{sim}(x, y_{avg}, \zeta_{opt}, \theta_j)$ is averaged over the y-dimension of the bond area between the y-bounds of the bond area y_1 and y_2,

$$P_{sim}(x, y_{avg}, \zeta_{opt}, \theta_j) = \frac{1}{y_2 - y_1} \int_{y_1}^{y_2} P_{sim}(x, y, \zeta_{opt}, \theta_j) dy. \tag{3.54}$$

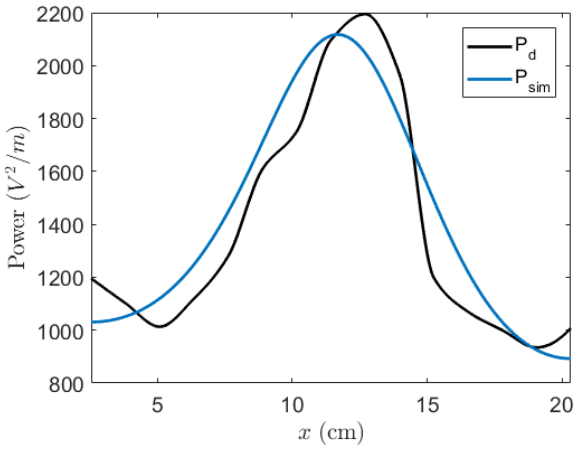


Figure 3.28: The desired power profile $P_d(x, 0)$, and the solution to Eq. 3.50 for a 5 tab heater, $P_{sim}(x, y_{avg})$. Both profiles are averaged over the y-dimension (as in Eq. 3.54) in this plot for a clearer comparison.

The final step to finding the optimal heater design parameters θ is to minimize the temperature error between the simulation and desired steady state temperature $T_{d,ss}$ in the bond area. This is expressed in the cost function $J_T(\theta)$,

$$J_T(\theta) = \|T_{d,ss} - T(\zeta_{opt}, \theta, x)\|_\infty, \tag{3.55}$$

where the simulated temperature along the bond center-line $T(\zeta_{opt}, \theta, x)$ is found using the optimal configuration times ζ_{opt} from Eq. 3.53, and putting this back into the model in Eq. 3.3. The cost is then minimized with respect to the heater design parameters θ to find the optimal set θ_{opt},

$$\theta_{opt} = \arg\min_{\theta} \; J_T(\theta), \tag{3.56}$$

by initializing the design with equal sized tabs, and systematically re-sizing the tabs until the minimum solution for $J_T(\theta_{opt})$ is found. This method varies the location of each tab gap on a grid by step size S_x to the left and right, and stores the results for all tab locations. The

result with the lowest error is kept, then this process is repeated until convergence. Then the step size S_x is decreased by half, and the process is repeated until S_x converges at the smallest grid step. A flow chart of this algorithm can be seen in Fig. 3.29. The results of these simulations are shown for heaters designed with 2 to 7 tabs in Fig. 3.30. This plot shows how the maximum temperature error (as in Eq. 3.1) decreases as more tabs are added, and with 3 or more tabs, there is less than 3 degrees of predicted temperature variation in the bondline.

3.8.5 Effect of heater length on tabbed heater design

A tabbed heater has one major drawback when compared to using multiple heaters. This is that there are limitations on the power differential that can be delivered to the bond area. As a tabbed heater gets longer, and the tabs get further from the bond area, the gradient is reduced in the bond area. This is demonstrated in Fig. 3.31. Heaters of 4, 5, and 6 inches in length are compared. For all three, the tabs are of equal length, and voltage is only applied to the center tab pair. If multiple heaters were used instead of tabs in this case, power would only be delivered in the area of that one heater. For a tabbed heater, power gets spread through the entire bond area, with the effect becoming greater for longer heaters. Despite this shortcoming, the advantages of a tabbed heater can outweigh this downside.

3.8.6 Comparing multiple heaters vs. tabbed heater designs

The optimal heater design solutions will be compared here similar to how the tabbed and multiple heater solutions were compared in 3.8.3. Instead of the same heater designs being applied to both the tabbed and cut heaters, the tabbed heaters were designed directly to minimize temperature error. A plot showing the results can be seen in Fig. 3.32.

The two design approaches presented here both have weaknesses. Using multiple heaters creates sharp gradients at the boundaries between heaters. Having multiple heaters in the

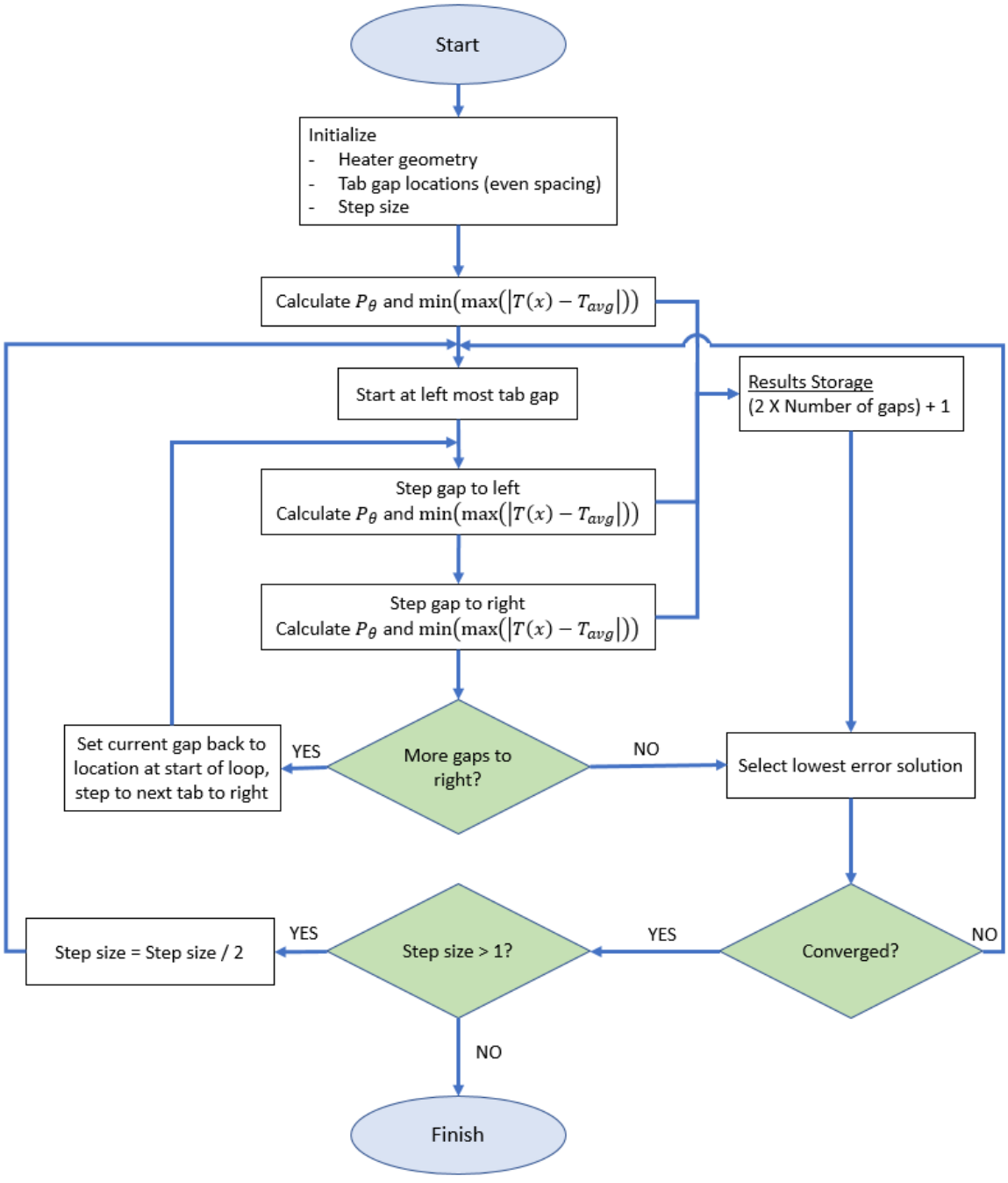


Figure 3.29: Flow chart of the algorithm used to solve for the optimal tab locations for a multi-tabbed heater.

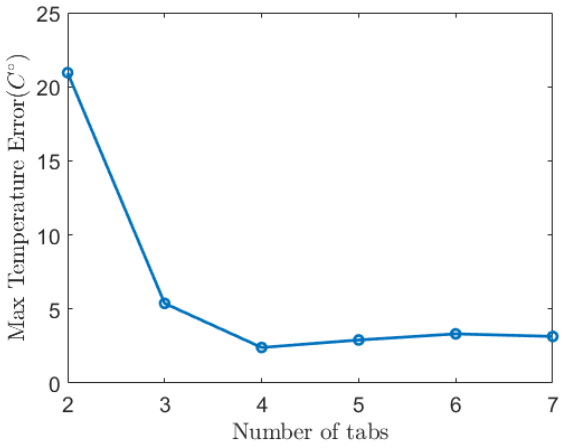


Figure 3.30: Predicted maximum temperature error as in Eq. 3.1 for a 12.7 cm long heater with different numbers of tabs.

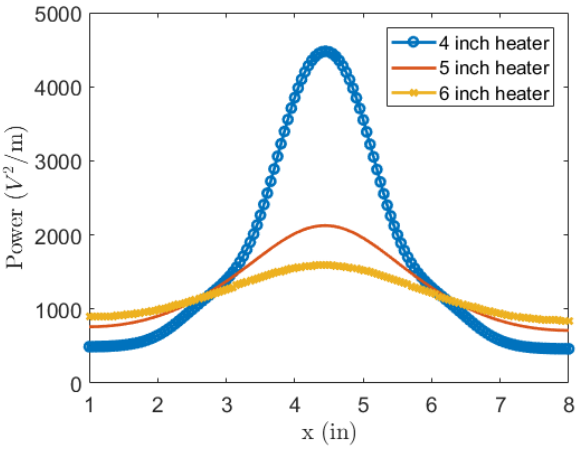


Figure 3.31: Power generated at the center-line of the bond area for a 5 tabbed heater with equally sized tabs, and only the center tabs on. Results shown for 4, 5, and 6 inch length heaters.

bondline also runs the risk of shorting between heaters, causing hot spots, burning, and unpredictable power generation. Additionally, gaps between heaters creates unknown bond properties and potential weaknesses. The tabbed heater approach on the other hand elimi-

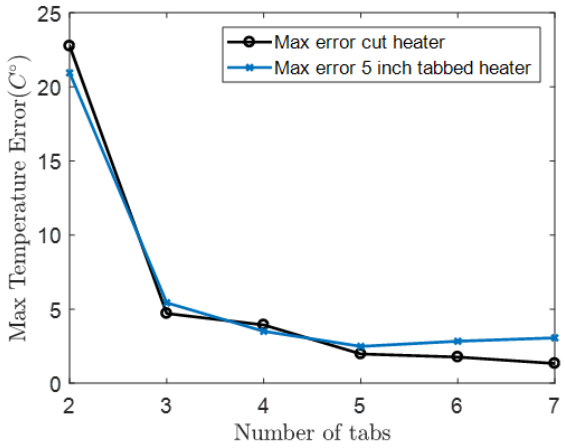


Figure 3.32: Comparison of temperature error between multiple heater and tabbed heaters with increasing number of tabs.

nates those problems, and even appears to give a better control for low numbers of tabs for this desired power profile, but for larger numbers of tabs, and with longer heaters (longer distance between tabs and bonding area) the performance begins to the lag the multiple heater approach. As such the choice of design depends on the application. For this application of a single lap joint the tabbed heater is the better design.

Chapter 4

EXPERIMENTAL RESULTS OF BOUNDARY CONTROL ON AN EMBEDDED HEATER

4.1 Introduction

This chapter demonstrates a method of precisely controlling bondline temperatures for curing adhesive in a bonded joint using an embedded heater with boundary control for actuation. The intended application is an OoA process for bonding composite adherends in the presence of substructures, which act as heatsinks. Uneven heat loss can cause a non-uniform temperature distribution in the bond area, which can lead to uneven curing, improper air removal causing defect formation [12], and a weakened joint. The method proposed here solves the temperature control problem by designing the boundary heating pattern to compensate for heatsinks that cause uneven heat loss during adhesive bonding. This is done by generating more heat in areas where there is more heat loss to increase temperature uniformity throughout the bondline.

Joining composite materials comes with a set of challenges that are uniquely different from traditional materials such as steel or aluminum. These traditional materials are often joined using fasteners or bolts, which usually necessites drilling holes. In composite materials, drilling holes creates stress concentrations by cutting fibers [8, 27], which can cause delamination near joints [11]. In response to these challenges, a common way of joining composites is through adhesive bonding. Compared to bolted or fastened joints, adhesive bonding has higher joint stiffness, superior fatigue performance, and higher structural efficiency [7, 16].

High strength adhesives used to join composites require high curing temperatures (upwards of 100 $^{\circ}C$) for faster curing time and better overall performance [14]. For an optimal

and even cure, the temperature should be uniform throughout the bonding region. A common way to accomplish this is to bond within an autoclave, where the entire part can be heated evenly, and both vacuum and pressure are added to remove air bubbles from the bondline that might become points of weakness in a cured joint [9]. Large parts require large autoclaves, which leads to high manufacturing costs [25, 52]. Even for smaller parts, the autoclave or oven approaches are not possible in the presence of components that are sensitive to elevated temperatures. For these cases, OoA methods for curing adhesive are used. These methods include heat blankets [23, 34], UV curing [18], and microwave heating [32, 50, 55]. External heating methods such as blankets heat the outside surface of the adherends, which can limit part thickness and geometry because carbon fiber composites materials have poor through-thickness heat conductivity. Similarly, UV can be used with photo-curable adhesives but are not suited to carbon fibers, which are not UV transparent. The use of microwaves is also limited to specialized applications since the presence of graphite fibers in composite materials can lead to arcing and local hot spots [26]. These limitations motivate research on new methods of curing adhesive bonds for composites.

The focus of this chapter is on joining using an embedded resistive heater (ERH) made of carbon fiber to heat high strength adhesives. To accomplish this, a single layer of carbon fiber, either dry or pre-impregnated with resin, is sandwiched between two layers of electrically isolating film adhesive, and placed within the bondline. Voltage is applied to the edges of this single layer of carbon fiber, which acts as a resistive heater. This technique has the advantage of adding heat directly at the bondline, minimizing the area of elevated temperature to near the bond, and thereby reducing energy usage. Previous work has been done using the ERH method for curing and bonding composite materials. The technique has been demonstrated to cure composite adherends using carbon mats [22, 38, 56] or carbon nanotube (CNT) buckypaper [31]. Bonded joints have been produced using embedded resistive heaters with a metal mesh [28, 41], CNT buckypaper [48] , and carbon fiber [5, 47]. The advantage of

using carbon fiber is that the heater could be of the same material as the adherends, so the bonding process would not add any extrinsic materials besides the adhesive in the bondline.

In an application where there is good insulation and uniform heat loss into the surrounding structure, using a carbon fiber embedded heater for adhesive bonding has been shown to produce comparable bond strength with joining performed in an autoclave or oven [5,47]. In practice, there can be non-uniform heat loss from the embedded heater into the adherends being joined so that there is a variation of temperature at the bondline. An example of this would be adhesively bonding a carbon fiber repair patch to a surface with substructures that act as heat sinks. This is illustrated in Fig. 4.1, where the repair patch is to be bonded on the outer surface of an aircraft with a supporting stringer underneath on the inner surface. Uniform heating of the bond area would produce a drop in temperature near the supporting stringer. This is the case for any heater, whether using an embedded heater, or other OoA methods. Therefore, there is a need to develop approaches to generate local variations in the heating profiles to achieve temperature uniformity.

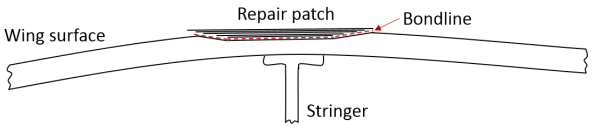


Figure 4.1: Schematic of a repair on an aircraft wing with substructure.

This chapter uses the technique proposed in Chapter 3 where spatially varying voltage is applied to the boundary of an embedded heater to produce non-uniform power generation within the heater. This is done through adding copper tabs along the edges of the carbon fiber embedded heater. By adding many such tabs, and connecting them to voltage independently in a controlled manner, it is possible to adjust how current flows through the heater, and therefore creating a power distribution within the heater that can be used to maintain a uniform temperature distribution in the bondline. The desired temperature T_d is a ramp

and hold trajectory that is specified by the adhesive manufacturer for optimal curing. After the adhesive is cured and the bonding complete, the copper tabs and excess material are removed, leaving only the heater that is embedded in the bondline. A schematic of a heater with a single copper tab on top and bottom is shown in Fig. 3.3. With AC voltage (V) applied on each tab, these tabs distribute the applied voltage so that current flows evenly across the heater, creating a uniform power distribution. Chapter 3 focused on designing a heater with multiple tabs for generating non-uniform power as shown in Fig. 3.1. To demonstrate this technique, bonding of a single-lap joint will be used as the target application, which is a benchmark setup for investigating adhesive bonding, e.g., [20, 40]. A heatsink will be placed under the center of the bond area to evaluate the effect of non-uniform heat loss, and the ability of the boundary control approach to correct for this.

4.2 Experimental validation

Three experiments were run to evaluate the technique described in Chapter 3. The first experiment was run with a uniform power heater to demonstrate the temperature variation in the bond area, and to fit a thermal model of the system. The second experiment is a step response run with a heater designed with multiple tabs in order to tune a controller for each tab. The third experiment was run with the same multiple tabbed heater and controlled to compensate for the temperature variation in the bond area.

Description of experimental system

The experimental setup can be seen in Fig. 3.2, with a more detailed layup shown in Fig. 4.2. The adherends for the single-lap joint were produced with HEXCEL HexPly 155 (BMS8-168), plain weave prepreg, with a cure temperature of 125 $^{\circ}C$. The adherends are 16 layers, with a $[0]_{16}$ stacking sequence, with a cured thickness measured at 5.81mm.

The first two experiments run for this chapter were run without adhesive. In the bondline

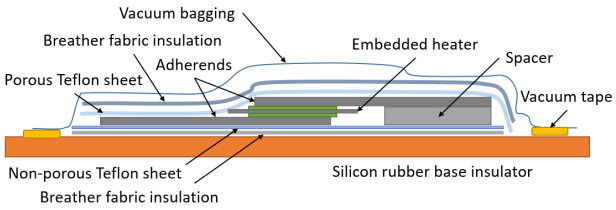


Figure 4.2: Sideview schematic of the experimental setup for bonding a single-lap joint. Not shown here is an aluminum heatink that goes beneath the lower adherend for the length of the setup left to right, as in Fig. 3.2.

on either side of the embedded heater a non-porous teflon sheet was used to electrically isolate the heater from the adherends. The final bonding experiment was done with supported structural adhesive film (Scotch-Weld AF 163-2U). While bonding, this adhesive provides electrical isolation between the embedded heater and adherends, as demonstrated in [5, 47]. The heaters used were made of dry carbon fiber fabric of plain weave, 24 X 24 thread count, 0.178 mm thick , and 1k tow (item number CF141 at The Composite Store). Along the center-line of the bond area, 15 thermocouples were placed 1.27 cm apart to collect temperature.

The entire experiment was run over a 1.27 cm sheet of silicon rubber for insulation. An aluminum (6061) bar with a cross-section of 0.635 cm by 5.08 cm was used as the heatsink under the center of the single-lap joint. At the ends of the aluminum bar, where it leaves the bonding area, bags of ice were placed to cool the bar during the experiments. Over the top of the adherends was placed a single layer of 3.175 mm thick silicon rubber for insulation, as well as a layer of breather fabric. The entire experiment was vacuum bagged, with a vacuum of 40.6 kPa (0.40 atm). A picture of this experimental setup can be seen in Fig. 4.3.

To control the voltage to the embedded heater two Arduino Megas were used. One to collect temperature and calculate control, and the other to control the voltage switching. The voltage was applied with a Variac power supply, and modulated with Omron G3NE-220TL

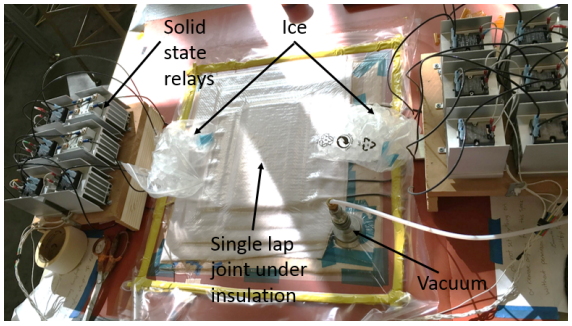


Figure 4.3: Picture of experiment run under vacuum, with labels pointing out ice placed on ends of the heatsink, solid state relays, vacuum port, and the single-lap joint covered by insulation.

solid state relays.

Step response with uniform heater

An embedded heater with uniform power generation as shown in Fig. 3.3, was used to heat a bond area with a heatsink placed underneath the bottom adherend as shown in Fig. 3.2. Instead of adhesive, non-porous teflon sheets were used to electrically isolate the heater and simulate adhesive. This experiment was done with a step input of 20 Volts at 50% duty cycle until the maximum temperature reached the target temperature of 125°C, which took approximately 32 minutes. The temperatures across the bond area at the end of this experiment $T_{m,u}$ are shown in Fig. 3.4. Under the center of the bonding area where the heatsink was placed there is a significant dip in temperature of approximately 30 °C. The temperature also is higher on one side of the bond area. This is consistent with other experiments run with a similar vacuum bagging approach, and appears to be skewed due to the placement of the through-bag vacuum connector. The percent variation (PV) of

temperature through the center-line of the bond area is 25.2 %, as calculated by

$$PV = \frac{T_{max} - T_{min}}{T_{max}} \times 100, \tag{4.1}$$

where T_{max} is the maximum measured temperature, and T_{min} is the minimum measured temperature. These results were used to generate the desired power profile shown in Fig. 3.28, which was calculated by fitting Eq. 3.3 and solving for Eq. 3.19.

Step response with tabbed heater

A second experiment was run using a tabbed heater designed by the minimization in Eq. 3.56. The purpose of this second experiment was to tune controller parameters for each tab, so that for a heater with 5 tabs on each edge, there would be 5 individually tuned controllers. This experiment was also run using non-porous teflon in place of adhesive.

A controller was designed by first fitting a step response of the system to each zone corresponding the area within each tab as in Fig. 3.1. The results of this were then used to fit a first order system for each zone

$$G_n = \frac{K_n}{\tau_n s + 1}. \tag{4.2}$$

The model parameters K_n and τ_n were found for each zone using a least squares approximation of the temperature response, i.e.,

$$\sum_k \left| T_n(t_k) - \begin{bmatrix} P_n & -\dot{T}_n(t_k) \end{bmatrix} \begin{bmatrix} K_n \\ \tau_n \end{bmatrix} \right|^2, \tag{4.3}$$

where T_n indicates the measured temperature in each zone n at time instants t_k, and P_n is the power input to the system, which is proportional to the square of the voltage applied to the heater.

Using the model in Eq. 4.2, a proportional and integral (PI) controller

$$C(s) = K_p + \frac{K_i}{s}; \tag{4.4}$$

was designed. The proportaional gain K_p and integral gain K_i were found that meet the chosen damping ratio $\zeta = 1$ and natural frequency $\omega_n = 10/\tau_n$ for the closed loop system. The full procedure for selection of the proportional and integral gains for a single heater has been described in [47] and is omitted here for brevity.

Using the results of a step response to a tabbed heater, the model parameters K_n and τ_n were determined. From these model parameters, control coefficients K_p and K_i were found and are displayed in Table 4.1.

Table 4.1: Model parameters K_n and τ_n, and calculated control constants K_p and K_i for each zone on a tabbed heater as shown in Fig. 3.1.

Zone #	K_n ($^\circ C/V^2$)	τ_n (s)	K_p ($V^2/^\circ C$)	K_i ($V^2/^\circ C$)
1	0.17	378.9	114.9663	1.5969
2	0.20	608.4	94.5808	0.8182
3	0.22	753.3	88.2111	0.6163
4	0.24	749.1	80.2145	0.5636
5	0.22	617.3	85.3578	0.7278

Bonding with tabbed heater

Using an embedded heater with 5 tabs as shown in Fig. 3.1, and control coefficients from Table 4.1, a single-lap joint was bonded with a heatsink under the center of the bond area. The tabs of this heater were designed and placed as described in section 3.8.4. The goal of this experiment was to demonstrate the ability to improve uniformity of temperature in the bond area when there is a heatsink causing heat loss in the bondline.

The results of this experiment can be seen in Fig. 4.4. In this plot, the temperatures measured with the tabbed heater $T_{m,t}$ are averaged over the first 15 minutes after reaching the target steady state temperature $T_{d,ss}$ (125°C), which is the time frame when most of the curing takes place [35]. The percent variation (PV) of temperature in the center-line of the bond area was measured to be 4.7%, and was predicted to be 4.5% in simulation. All of the thermocouple data for the first and third experiments is shown in Table 4.2. The thermocouples are numbered such that thermocouple 1 is located on the left side of Fig. 3.2. The temperature variation in the bondline while using the multiple tabbed heater is $\pm 3\ ^\circ C$, which is within the desired range of $\pm\ 6\ ^\circ C$, and matches well with simulation.

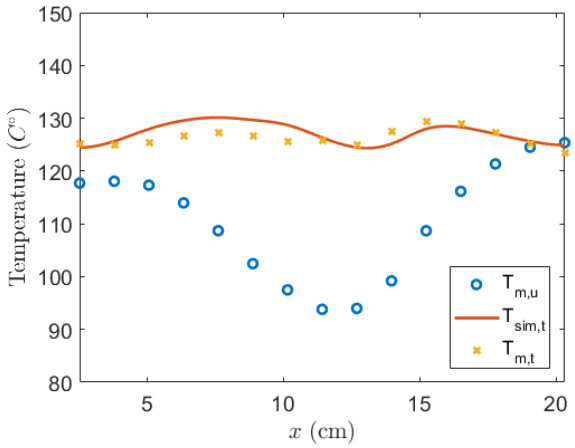


Figure 4.4: Temperature profiles along the center-line of the bond area with heatsink under center. Shown for a uniform heater experiment $T_{m,u}$, a tabbed heater simulation $T_{sim,t}$ when controlling to the desired temperature $T_{d,ss}$, and a tabbed heater experiment $T_{m,t}$ averaged over the first 15 minutes after reaching $T_{d,ss}$. Both tabbed simulation and experiment were run with an optimized heater with 5 tabs as in Fig. 3.1.

Table 4.2: Temperature data for each thermocouple (TC) along the bond area center-line. Data is shown for the single tab heater experiment (uniform power) $T_{m,u}$, and the multiple tab heater experiment $T_{m,t}$, which is averaged over the first 15 minutes after reaching the steady state temperature $T_{d,ss} = 125\ ^\circ\ C$.

TC# (n)	Uniform heating $T_{m,u}$ ($^\circ C$)	Boundary control $T_{m,t}$ ($^\circ C$)
1	117.7	125.1
2	118.1	124.9
3	117.3	125.5
4	114.0	126.7
5	108.7	127.2
6	102.5	126.6
7	97.5	125.5
8	93.8	125.7
9	94.0	125.0
10	99.2	127.6
11	108.7	129.4
12	116.2	129.0
13	121.3	127.3
14	124.5	125.2
15	125.4	123.4

4.3 Discussion and conclusions

This chapter examines the use of boundary control of an embedded resistive heater. The specific application studied is a single-lap joint with a heatsink under a section of the bond area. A model, in combination with experimental data, is developed to predict the desired power profile $P_d(x, y)$ needed to maintain the steady state uniform temperature $T_{d,ss}$ within the bond area. The power produced by a tabbed heater is simulated with an FDM model, which is used with the desired power profile to design a tabbed heater to minimize temperature variation in the bond area. Using a heater designed with 5 tabs, experimental results show a more than 5 times reduction in the maximum temperature variation when compared

to a uniform power (single tab) embedded heater, from 31.6 $^\circ$ C to 6.0 $^\circ$ C. This improves the temperature variation to $\pm$ 3 $^\circ C$, which is well within the goal of reaching better than $\pm$ 6 $^\circ C$ in the bond area. These results match well with simulation, with the expected percent variation (PV) being 4.5%, and a measured PV of 4.7%.

Chapter 3 demonstrated an analytical approach to designing an embedded heater with multiple tabs, which is computationally efficient, and is able to predict the amount of temperature variation. While this chapter focuses on rectangular heaters and bond areas, this approach is general and could be extended to other geometries and different heatsink patterns.

Chapter 5

INVESTIGATION OF EMBEDDED RESISTIVE HEATING FOR HIGH STRENGTH ADHESIVE BONDING ON RIGID SPACE STRUCTURES

5.1 Introduction

This chapter examines high strength adhesive bonding using carbon fiber embedded heaters to fabricate large, rigid structures in space from smaller, easy-to-transport modules. The use of carbon fiber embedded resistive heaters can be used to produce high quality adhesive bonds under vacuum bagging. This study shows that this technique has similar results when done in higher vacuum conditions with a method of applying external pressure to the adherends. The advantages of this bonding method, as opposed to fastening, are the minimal use of extra materials as the embedded heater becomes part of the composite structure, and energy efficiency due to the targeted heating at the bondline. Such embedded-heater-based joining could be a key enabler for modular manufacturing and repair of space structures.

One of the primary difficulties and costs associated with space exploration is the limitations imposed by the current necessity of sourcing materials, supplies, and structures from Earth. Payload fairings are small by necessity, and range from about 3 to 10 meters in diameter [21]. For example, the record setting launch of 104 satellites by the Indian Space Research Organization (ISRO) on February 15th, 2017, was done with a 3.2 meter diameter payload fairing. Limitations on fairing size has encouraged the design of space structures to include a variety of solutions so that rigid structures can be transported with a small footprint, and then expanded when deployed. Some examples of this include inflatable structures [19, 43], and origami-inspired expanding booms [30, 57]. Considering the cost of putting a payload

into space, and size limitations, it is important to have high strength, low weight, and cost effective ways of joining rigid materials in space after deployment.

Techniques for joining are important for initial construction, as well as for repair. This research examines one such technology, which is high strength adhesive bonding using carbon fiber embedded heaters. There are a number of advantages of bonding as a method of joining. Minimal material is added to the structure when compared to fastening, and the bond becomes part of the structure, which equates to less weight to bring to space. It does not require drilling into materials, which for composites can cause damage and stress concentrations in the adherends, requires heavy tooling, and drill heads that need replacement [6,8]. However, there are downsides to adhesive bonding. Some high strength adhesives require cold storage before use, elevated and controlled temperatures for curing, and skilled technicians for application.

The use of carbon fiber embedded heaters for bonding has been shown to produce bonds of comparable strength to more traditional methods, such as heat blankets or ovens [4,5,47]. Embedded heaters for bonding would be particularly well suited for space applications due to its minimal need of large specialized equipment, and it is extremely efficient by targeting heating to the bondline. The adhesives and carbon fiber used for embedded heating can also be chosen to match the materials in the adherends, thereby reducing stress concentrations.

This chapter investigates the feasibility of adhesive bonding in space using carbon fiber embedded heaters. In normal Earth operating conditions, a vacuum and over-pressure are applied by vacuum bagging. The combination of vacuum and pressure is necessary to minimize voids within the bond [9,14]. This process requires extra equipment such as a vacuum pump, and materials used once and discarded. The advantage of attempting this in space-like conditions is that there is a natural vacuum. However, using the natural vacuum of space introduces new problems to be investigated, two of which this study will seek to address. The first is whether the high vacuum of space would cause improper curing and a weakened

joint due to outgassing. The second is that it becomes necessary to apply an external source of over-pressure to the joint. This study will look at whether space-like vacuum with the addition of over-pressure can produce a bond of similar quality to bonding with vacuum bagging, and bonding done in an autoclave.

5.2 Methods

This experimental study examines the ability of embedded heaters to form a high quality adhesive bond in a vacuum chamber with an apparatus applying external pressure. The quality of the bond will be compared to specimens created using an embedded heater with the standard vacuum bagging technique, as well as specimens bonded in an autoclave. The cure profile for all experiments was a ramp up rate of 3 °C per minute until reaching 177 °C, hold at this temperature for 2 hours, then ramp down at a rate of 3 °C per minute. A single-lap joint configuration was used for this study, with specimens created to the ASTM standard D5868 [1].

The heater configuration which was used for the vacuum chamber and vacuum bag experiments can be seen in Fig. 5.1. Extra fabric was added on all sides of the bond area to minimize edge effects. After curing, this extra fabric can be removed. Voltage was applied to copper tabs at either end of the heater, with current flowing in the long direction of the heater. An Arduino based controller with solid state relays (SSRs) modulated voltage to the embedded heater, with power provided by a Variac power supply. Full details of this controller can be found in previous work [47].

The rest of this section will describe the three different methods used to create specimens, bonding in i) a vacuum chamber, ii) under a vacuum bag, and iii) in an autoclave.

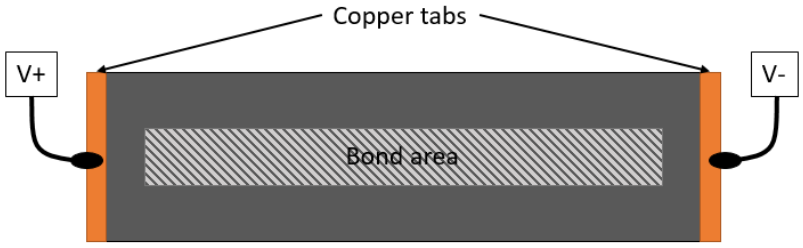


Figure 5.1: Schematic of an embedded heater, with AC voltage (V) applied at both ends. The bond area, which is smaller than the heater, is marked with cross-hatching.

5.2.1 Bonding experiment in a vacuum chamber

A bonding experiment was done in a vacuum chamber, which was brought to a vacuum pressure of 1% of atmospheric pressure. In addition to this vacuum, over-pressure of 33.86 kPa was applied using a fixture of plates and screws, which can be seen in Fig. 5.2. The joint was placed between the silicon rubber spacers. Using pressure sensors (FlexiForce A401-25) under the pressure plate, the screws were adjusted to get uniform pressure across the bond area.

The layup of the single-lap joint to be bonded in the vacuum chamber can be seen in Fig. 5.3. This consisted of an embedded heater, surrounded by film adhesive on both sides, all placed inside a bottom and top adherend. Non-porous teflon sheet was placed between the embedded heater and adherends to avoid electrical shorting.

5.2.2 Bonding experiment under vacuum bag

A bonding experiment was done using an embedded heater under a vacuum bag. The layup was similar to the experiment done in the vacuum chamber. The layers for this bonding can be seen in Fig. 5.4. Vacuum pressure of 33.86 kPa (approximately 66% or atm) was applied

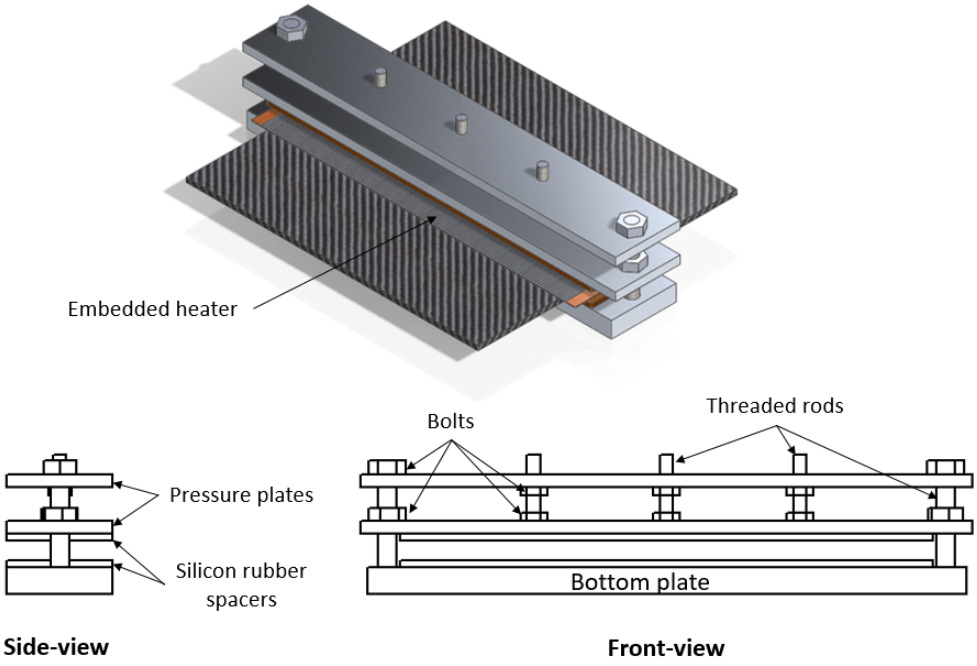


Figure 5.2: Schematic of fixture to apply over-pressure to single-lap joint during bonding in a vacuum chamber.

using a vacuum pump. This pressure is the same pressure applied via the pressure apparatus in the vacuum chamber.

5.2.3 Bonding experiment in autoclave

A specimen was joined in an autoclave using the same configuration shown in Fig. 5.4, except that the silicon rubber base insulator was replaced with a sheet of aluminum. Additionally, no voltage was applied to the embedded heater, as the temperature was controlled using the autoclave. The adhesive manufacturer did not have a pressure recommendation for bonding, so an over-pressure of 344.74 kPa was applied to match previous studies done comparing autoclave to embedded heater adhesive bonding [5]. This pressure was substantially higher

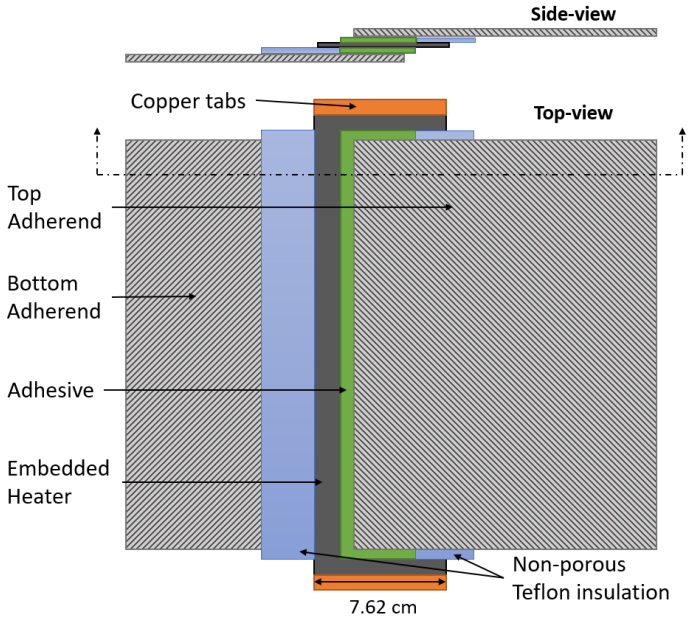


Figure 5.3: Side and top views of the embedded heating layup with adherends. This layup was used in the vacuum chamber and for vacuum bagging.

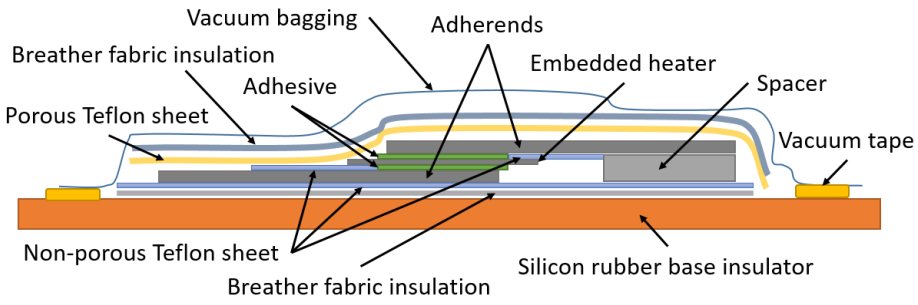


Figure 5.4: Side view schematic of bonding with embedded heater under vacuum.

than what was achieved in the vacuum chamber or under vacuum bagging.

5.3 Materials

The single-lap joints produced in this study are comprised of two carbon fiber composite adherends, bonded together with a single layer of carbon fiber prepreg, sandwiched between two layers of film adhesive. Both the adherends and embedded heater were made with graphite/epoxy plain weave prepreg produced by HEXCEL (BMS8-168), and the adherends were 8 layers with a stacking sequence of $[0, 90]_{2s}$.

The film adhesive was the Master Bond polymer system FLM36-LO. This adhesive did not have a carrier film, but carrier film is recommended when using embedded heaters to reduce the chance of shorting between the heater and adherends. A noteworthy characteristic of the chosen adhesive is its low outgassing conformity to the NASA ASTM E595 specification. It is electrically insulating, and can be stored at room temperature. The recommended cure temperature is 1-2 hours at 177 °C.

5.4 Results

The specimens created in the vacuum chamber, under a vacuum bag, and in an autoclave were compared by first examining specimens under a microscope, and then through tensile testing. Before testing, the specimens were cut out of the 22.86 cm specimens into approximately 2.54 cm samples, as shown in Fig. 5.5.

5.4.1 Bondline under the microscope

After each set of adherends were cut into 8 tensile specimens, 2 samples from each were examined under a microscope. The samples examined are numbered 2 and 4 as shown in Fig. 5.5. After cutting, the edges of these samples were sanded consecutively with 240, 800, and 1200 grit sandpaper. They were then wiped with paper towels and acetone for cleaning. On the two cut edges of these samples, the center 1.27 cm of the bondline was scanned with a microscope to look for void content.

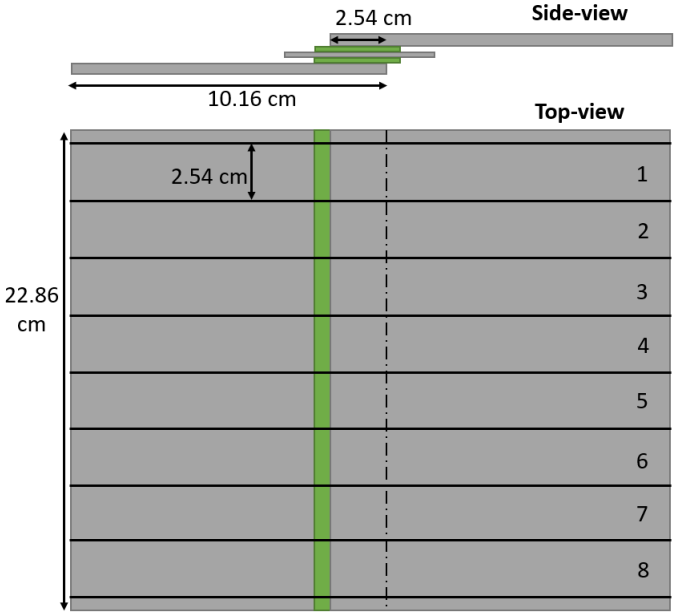
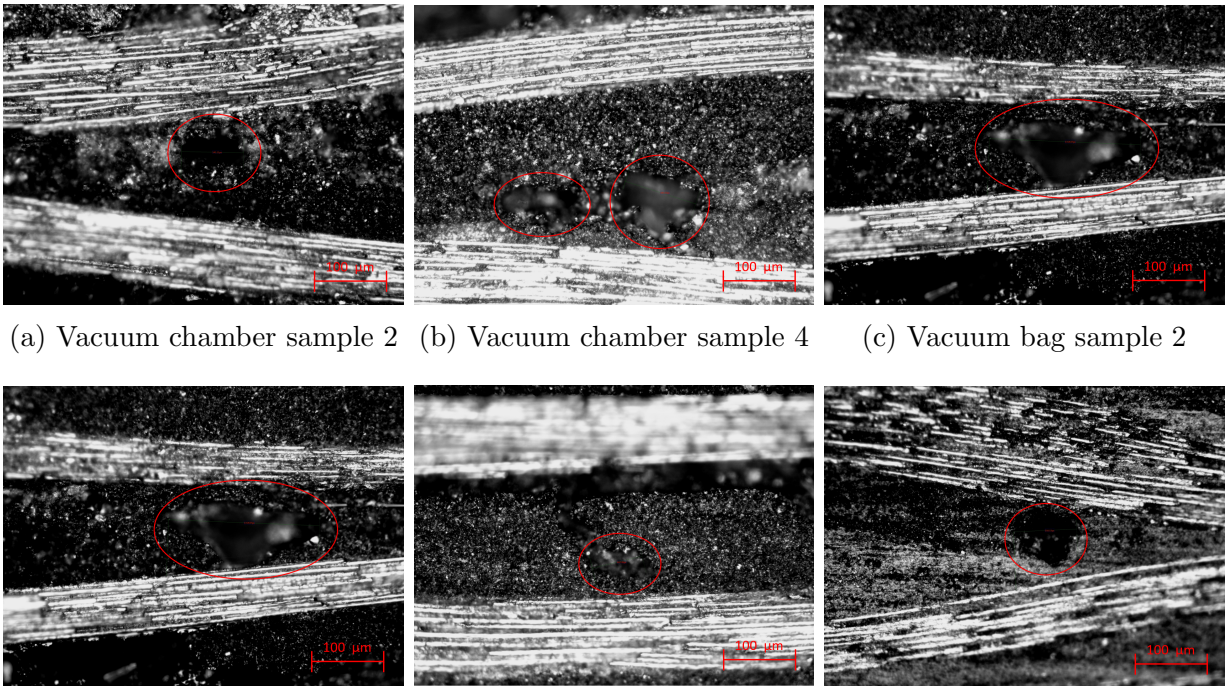


Figure 5.5: Schematic of a bonded single lap joint, to be cut into 2.54 cm wide specimens for tensile testing.

Quantitatively, both the vacuum chamber and vacuum bagging specimens appeared to have comparable void content, though these voids did not make up a significant fraction of the bondline. The samples produced in the autoclave did have noticeably less void content. A snapshot of the bondline for each sample can be seen below in Fig. 5.6. Photos were selected where voids were found, which are highlighted by red circles in the photos.

5.4.2 Tensile testing

Using the prepared samples produced from bonding in a vacuum chamber, under vacuum bagging, and in an autoclave, tensile tests were performed in a 100 kN capacity Instron test frame. The raw results can be seen in Fig. 5.7, with the maximum loads listed in Table 5.1. Comparing just the vacuum chamber and vacuum bag results, the plots in Fig. 5.7 show that the vacuum chamber produced samples with noticeably less scatter, and on average



(a) Vacuum chamber sample 2 (b) Vacuum chamber sample 4 (c) Vacuum bag sample 2

(d) Vacuum bag sample 4 (e) Autoclave sample 4 (f) Autoclave sample 4

Figure 5.6: Photos of the bondline from samples of each bonding method. Photos were chosen where voids were found, which are circled in red.

had less displacement at failure. However, the mean load at failure was about 5% lower for the samples produced in the vacuum chamber. Taking a two sample t-test on this data, it cannot be said that these samples are statistically different, with $p = 0.14$. This indicates that the results show that the tensile strengths of the two methods are similar. After the samples were broken, the fracture surfaces were examined, and adhesive was found on both sides of every fracture surface. This implies good adhesion and cohesive failure. The results from the autoclave had substantially lower tensile strength than both other methods

Table 5.1: Table of tensile test results for each method, with maximum tensile load at failure, mean maximum load, and standard deviations.

Sample (#)	Vacuum Chamber Max Load (N)	Vacuum Bag Max Load (N)	Autoclave Max Load (N)
1	7489.4	7608.5	2995.3
2	7459.8	7217.2	4038.9
3	7550.8	8445.3	4890.8
4	7194.4	8339.1	4738.9
5	8414.3	7872.1	5096.3
6	7595.2	7615.6	5277.4
7	7372.7	7686.8	4480.7
8	7719.9	8924.2	4594.3
Mean	7599.6	7963.6	4514.1
Std. Dev.	364.1	558.7	722.5

5.5 Conclusions

This study examined the feasibility of using embedded resistive heaters made of carbon fiber for high strength adhesive bonding in space. This was done by bonding a test specimen in a vacuum chamber at approximately 1% of atmospheric pressure, which was compared to bonding under a vacuum bag at 66% of atmospheric pressure. Samples were also made in an autoclave. A pressure apparatus was used in the vacuum chamber so that equal over-pressure was applied when compared with vacuum bagging. No significant difference in void content was found by looking at the bonds under a microscope, or by tensile testing. While it did appear that the samples bonded at a higher level of vacuum did have lower variation in bond strength, and a slightly lower average load at failure, these difference were not statistically significant. An additional comparison was made with samples made in an autoclave. The autoclave samples had significantly lower tensile strengths. It is possible that the additional pressure in the autoclave caused a resin starved bondline, resulting in lower strength. This work used relatively few samples, and could benefit from a larger sample size. Future work

could look at higher levels of vacuum, as well as other common high strength adhesives.

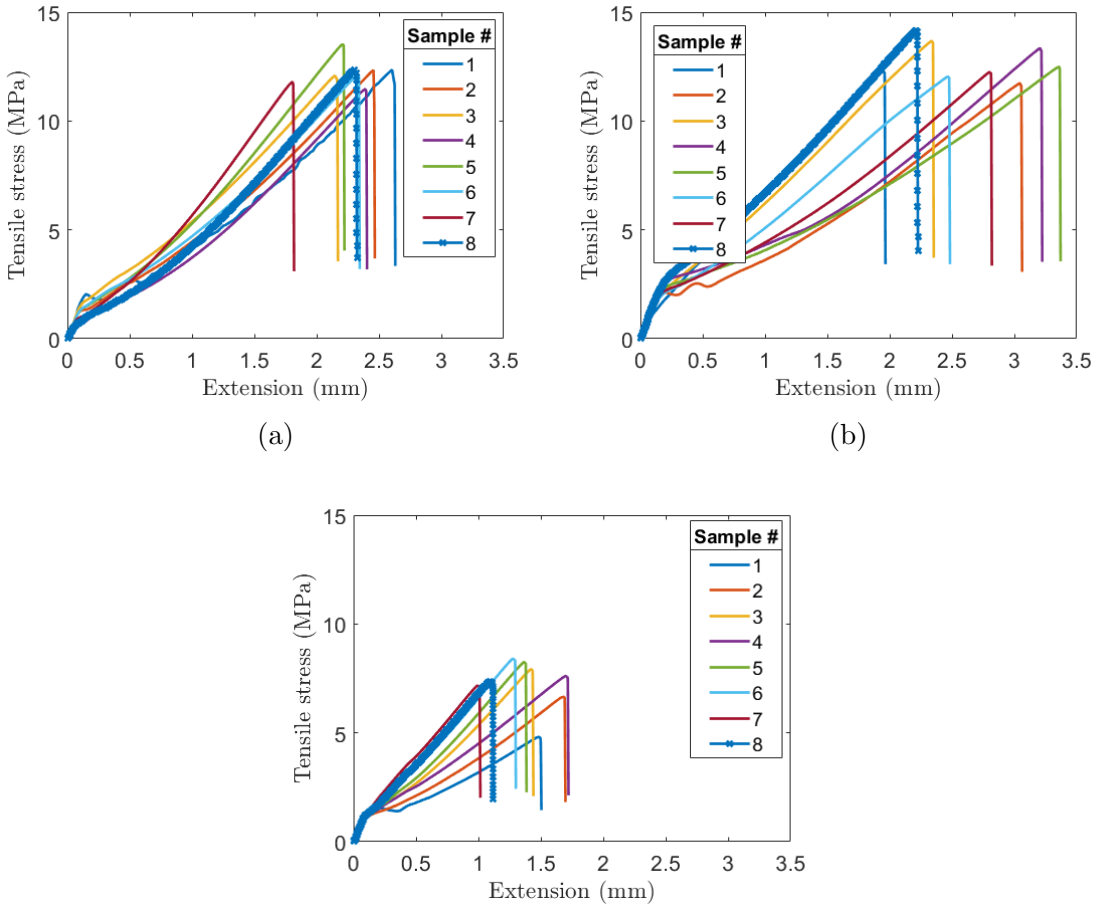


Figure 5.7: Tensile testing results for a) vacuum chamber, b) vacuum bagging, and c) autoclave.

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

This research investigated the limits of temperature estimation and control when using carbon fiber embedded resistive heaters for adhesive bonding. A single-lap joint configuration was used as a benchmark example. This work first focused on sensing and estimating internal bondline temperatures, then looked at methods for precisely controlling temperatures in a bond area with non-uniform heat loss, and last looked at whether this method could be useful for joining and manufacturing in space applications.

6.1.1 Sensing bondline temperature

In Chapter 2 a method was developed to use external sensors for estimating and controlling inner bondline temperatures when joining composite materials with an embedded resistive heater. Two external thermocouples were used, along with a single thermocouple internal to the bondline during a dry-run to fit a model during a step response. During bonding, the internal thermocouple can be removed, and the external measurements used with the developed model to precisely control bondline temperatures.

The estimation method developed was applied on a single-lap joint with thick adherends (5.81 mm), where there was a temperature difference of approximately 20 $^{\circ}C$ between the bondline and the outer suface of the adherend. Despite this large temperature difference, the method developed in this research was able to accurately estimate and control the bondline temperature from external measurements to within $\pm$ 2 $^{\circ}C$ of the target after initial

transients.

6.1.2 Boundary control on an embedded heater

In Chapter 3 two methods for generating non-uniform power with embedded resistive heaters were investigated: using multiple heaters, and using a single heater with multiple tabs. Producing non-uniform power is useful for situations when heating a system with non-uniform heat loss, such as when a heatsink is present. A model was first developed to predict the desired power within the bondline of a single-lap joint. This was done by fitting a lumped heat method model with experimental data. A finite difference method model was then developed to calculate the amount of power produced by a heater with multiple tabs on each side. These models were used to design heater configurations to minimize the temperature variation in the bond area for a single-lap joint that had a significant heat loss under the center region. Experimentally, a 30 $^{\circ}C$ temperature dip was seen in the center of the bond area where an aluminum heat sink was placed. A tabbed heater was designed and used for experimental validation of the developed model in Chapter 4. The experimental validation showed good agreement with simulations, producing more than a 5 times reduction in the maximum temperature error in the bond area. The initial design goal was to produce better than $\pm$ 6 $^{\circ}C$ of temperature variation within the bondline, and using a tabbed heater, $\pm$ 3 $^{\circ}C$ was demonstrated.

6.1.3 Embedded resistive heaters for joining space structures

A study was done to examine the feasibility of using embedded resistive heaters for adhesive bonding in space. This was done by comparing bonding a single-lap joint under three configurations: i) in a vacuum chamber at 1% atmospheric pressure (atm) with a clamping pressure apparatus, ii) under a vacuum bag at 66% atm, and iii) in an autoclave with 345 kPa over-pressure. The clamping pressure applied in the vacuum chamber matched the pres-

sure that the vacuum bagging applied to the bonded area. It was found that the vacuum chamber and vacuum bagging produced tensile specimens with similar strengths and void content despite the difference in vacuum. The tensile results for the specimens produced in the autoclave were substantially lower, which may have been due to excessive resin flow and starvation in the bond due to the high pressure. Overall, the results from this study are encouraging for the use of embedded heaters for bonding in the high vacuum of space.

6.2 Future work

There are some areas within this work that warrant future research. Regarding estimating bondline temperatures, this research focused on estimating single locations within the bond. While this is adequate for simple geometries with relatively uniform thermal properties, a more advanced model should be developed to predict the temperatures throughout the entire bond area, even in the presence of heatsinks or other thermal abnormalities. For example, in a scarf repair, the bonded adhered is of varying thickness. It would be useful to produce a model that would predict how this affects temperature estimates.

To extend the work done with using boundary control on embedded heaters, work could be done with other geometries of bonds and heatsink patterns. An example of this would be a round heater, as shown in Fig. 6.1, which would be a common configuration for a standard composite scarf repair. Additionally, an assumption was made that tabs were activated in pairs, to act like control zones. With a rectangular embedded heater, activating tabs in pairs limits the ability to control temperatures in the bond to a single dimension. It would be of interest to activate each tab on a heater boundary individually, which would give the ability to target power in two dimensions.

To further prove the usefulness of embedded heaters for adhesive bonding on space structures, more experiments should be done that more closely reflect the harsh environment of space, which includes radiation, atomic oxygen, and a vacuum of greater then $1e-5$ Torr.

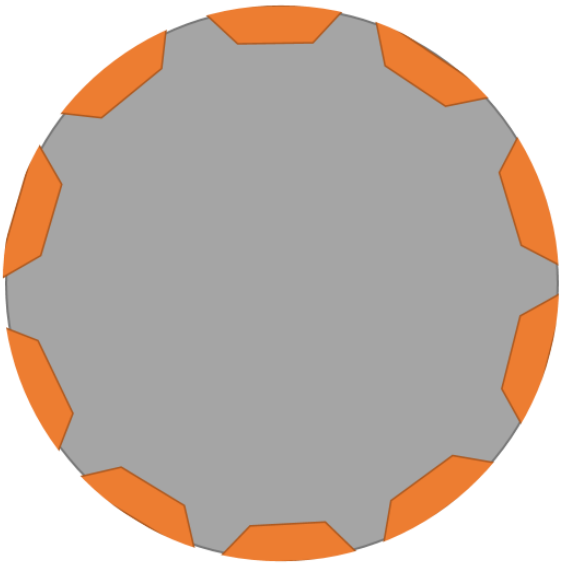


Figure 6.1: Example of a round heater geometry with tabs.

This research also used a simple mechanical clamping mechanism for applying pressure in vacuum. A method to apply more uniform pressure should be investigated, potentially using inflatable bladders. Finally, a comparative cost analysis between adhesive bonding and other forms of joining specific to space would be valuable for demonstrating usefulness.

BIBLIOGRAPHY

[1] ASTM D5868 Standard Test Method for Lap Shear Adhesion for Fiber Reinforced Plastic (FRP), 2001.

[2] Gasser F. Abdelal, Georgia Georgiou, Jonathan Cooper, Antony Robotham, Andrew Levers, and Peter Lunt. Numerical and Experimental Investigation of Aircraft Panel Deformations During Riveting Process. *Journal of Manufacturing Science and Engineering*, 137(1):011009, 2015.

[3] T J Ahmed, D Stavrov, H E N Bersee, and A Beukers. Induction welding of thermoplastic composites - An overview. *Composites Part A: Applied Science and Manufacturing-Applied Science and Manufacturing*, 37(10):1638–1651, 2005.

[4] M. Ashrafi, B. P. Smith, S. Devasia, and M. E. Tuttle. Embedded resistive heating in composite scarf repairs. *Journal of Composite Materials*, page 0021998316673706, 2016.

[5] Mahdi Ashrafi, Santosh Devasia, and Mark E. Tuttle. Resistive embedded heating for homogeneous curing of adhesively bonded joints. *International Journal of Adhesion and Adhesives*, 57:34–39, 2015.

[6] A.A. Baker. Fibre composite repair of cracked metallic aircraft components practical and basic aspects. *Composites*, 18(4):293–308, 1987.

[7] A. Baldan. Adhesively-bonded joints and repairs in metallic alloys, polymers and composite materials: Adhesives, adhesion theories and surface pretreatment. *Journal of Materials Science*, 39(1):1–49, 2004.

[8] M D Banea and L F M da Silva. Adhesively bonded joints in composite materials: an overview. *Journal of materials Design and Applications*, 223(L):1–18, 2009.

[9] Willard D Bascom and Robert L Cottington. Air Entrapment in the Use of Structural Adhesive Films. *The Journal of Adhesion*, 4(3):193–209, 1972.

[10] Theodore L. Bergman, Adrienne S. Lavine, Frank P. Incropera, and David P. DeWitt. *Fundamentals of Heat and Mass Transfer*. Wiley, Hoboken, NJ, 7th edition, 2011.

[11] P. P. Camanho and F. L. Matthews. Delamination onset prediction in mechanically fastened joints in composite laminates. *Journal of Composite Materials*, 33(10):906–927, 1999.

[12] Timotei Centea, Garrett Peters, and Steven R Nutt. Thermal Gradients During Out-of-Autoclave Prepreg Cure : Case Study Using a Heated Tool. *Proceedings of the SAMPE 2015 Technical Conference*, 2015.

[13] Demeng Che, Ishan Saxena, Peidong Han, Ping Guo, and Kornel F. Ehmann. Machining of Carbon Fiber Reinforced Plastics/Polymers: A Literature Review. *Journal of Manufacturing Science and Engineering*, 136(3):034001, 2014.

[14] Lucas F M Da Silva, R. D. Adams, and M. Gibbs. Manufacture of adhesive joints and bulk specimens with high-temperature adhesives. *International Journal of Adhesion and Adhesives*, 24(1):69–83, 2004.

[15] D.D. Edie. The effect of processing on the structure and properties of carbon fibers. *Carbon*, 36(4):345–362, 1998.

[16] B. Egan, C. T. McCarthy, M. A. McCarthy, P. J. Gray, and R. M. O'Higgins. Static and high-rate loading of single and multi-bolt carbon-epoxy aircraft fuselage joints. *Composites Part A: Applied Science and Manufacturing*, 53:97–108, 2013.

[17] A F Emery. Preliminary results for estimating the backside heat losses of a composite panel. *Proceedings of the 14th International Heat Transfer Conference*, 2010.

[18] Thierry Glauser, Mats Johansson, and Anders Hult. A comparison of radiation and thermal curing of thick composites. *Macromolecular Materials and Engineering*, 274:25–30, 2000.

[19] Petra Gruber, Sandra Häuplik, Barbara Imhof, Kürsad Özdemir, Rene Waclavicek, and Maria Antoinetta Perino. Deployable structures for a human lunar base. *Acta Astronautica*, 61(1-6):484–495, 2007.

[20] J.a. Harris and R.a. Adams. Strength prediction of bonded single lap joints by non-linear finite element methods. *International Journal of Adhesion and Adhesives*, 4(2):65–78, 1984.

[21] Abhishek Jain and Nejc Trost. Current and Near-Future Space Launch Vehicles for Manned Trans-Planetary Space Exploration: Phobos-Deimos Mission Architecture Case Study. *AIAA SPACE 2013 Conference and Exposition*, pages 1–8, 2013.

[22] Christopher Joseph and Christopher Viney. Electrical resistance curing of carbon-fibre/epoxy composites. *Composites Science and Technology*, 60(2):315–319, 2000.

[23] K. B. Katnam, L. F M Da Silva, and T. M. Young. Bonded repair of composite aircraft structures: A review of scientific challenges and opportunities. *Progress in Aerospace Sciences*, 61:26–42, 2013.

[24] Erwin Kreyszig. *Advanced Engineering Mathematics*. John Wiley, Hoboken, NJ, 9th edition, 2006.

[25] Steven A. Lane, John Higgins, Adam Biskner, Greg Sanford, Chris Springer, and Jerome Berg. Out-of-Autoclave Composite Fairing Design, Fabrication, and Test. *Journal of Manufacturing Science and Engineering*, 133(3):031020, 2011.

[26] Woo I. Lee and G. S. Springer. Interaction of Electromagnetic Radiation with Organic Matrix Composites. *Journal of Composite Materials*, 18(4):357–386, 1984.

[27] E. Madenci, S. Shkarayev, B. Sergeev, D.W. Oplinger, and P. Shyprykevich. Analysis of composite laminates with multiple fasteners. *International Journal of Solids and Structures*, 35(15):1793–1811, 1998.

[28] S. Mahdi, H.-J. Kim, B.A. Gama, S. Yarlagadda, and J.W. Gillespie. A comparison of Oven-cured and induction-cured adhesively bonded composite joints. *Journal of Composite Materials*, 37(6):519–542, 2003.

[29] Bartolomé Mas, Juan P. Fernández-Blázquez, Jonathan Duval, Humphrey Bunyan, and Juan J. Vilatela. Thermoset curing through Joule heating of nanocarbons for composite manufacture, repair and soldering. *Carbon*, 63:523–529, 2013.

[30] M C Natori, Nobuhisa Katsumata, Hiroshi Yamakawa, Hiraku Sakamoto, and Naoko Kishimoto. Conceptual model study using origami for membrane space structures. In *Proceedings of the ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2013.

[31] Nam Nguyen, Ayou Hao, Jin Gyu Park, and Richard Liang. In Situ Curing and Out-of-Autoclave of Interply Carbon Fiber/Carbon Nanotube Buckypaper Hybrid Composites Using Electrical Current. *Advanced Engineering Materials*, pages 1–7, 2016.

[32] C. Nightingale and R. J. Day. Flexural and interlaminar shear strength properties of carbon fibre/epoxy composites cured thermally and with microwave radiation. *Composites - Part A: Applied Science and Manufacturing*, 33(7):1021–1030, 2002.

[33] Katsuhiko Ogata. *Discrete-Time Control Systems*. Prentice Hall, NJ, 2nd edition, 1995.

[34] T. K. Papathanasiou, S. I. Markolefas, S. P. Filopoulos, and G. J. Tsamasphyros. Heat Transfer in Thin Multilayered PlatesPart II: Applications to the Composite Patch Repair Technique. *Journal of Heat Transfer*, 133(2):021303, 2011.

[35] Sanjay Parthasarathy, Susan C. Mantell, and Kim a. Stelson. Estimation, Control and Optimization of Curing in Thick-Sectioned Composite Parts. *Journal of Dynamic Systems, Measurement, and Control*, 126(4):824, 2004.

[36] Garth M. Pearce, Alastair F. Johnson, Rodney S. Thomson, and Donald W. Kelly. Experimental Investigation of Dynamically Loaded Bolted Joints in Carbon Fibre Composite Structures. *Applied Composite Materials*, 17(3):271–291, 2010.

[37] Neeraj Rai and Ranga Pitchumani. Rapid cure simulation using artificial neural networks. *Composites Part A: Applied Science and Manufacturing*, 28(9-10):847–859, 1997.

[38] B. Ramakrishnan, L Zhu, and R Pitchumani. Curing of Composites Using Internal Resistive Heating. *Journal of Manufacturing Science and Engineering*, 122(1):124–131, 2000.

[39] Gerald W Recktenwald. Finite-Difference Approximations to the Heat Equation. *Mechanical Engineering*, 0(2):1–27, 2004.

[40] F. L. Ribeiro, L. Borges, and J. R.M. Dalmeida. Numerical stress analysis of carbon-fibre-reinforced epoxy composite single-lap joints. *International Journal of Adhesion and Adhesives*, 31(5):331–337, 2011.

[41] A. Rider, C. H. Wang, and J. Cao. Internal resistance heating for homogeneous curing of adhesively bonded repairs. *International Journal of Adhesion and Adhesives*, 31:168–176, 2011.

[42] William G Roeseler, Branko Sarh, and Max U Kismarton. Composite Structures: The First 100 Years. *16th International Conference on Composite Materials*, pages 1–10, 2007.

[43] Phil D Sadler and Sadler Machine Company. Prototype BLSS Lunar-Mars Habitat Design. In *44th International Conference on Environmental Systems*, number July, 2014.

[44] Carlo Santos, Thomas Plaisted, Diego Arbelaez, and Siavouche Nemat-Nasser. Modeling and testing of temperature behavior and resistive heating in a multi-functional composite. *Proceedings of SPIE - The International Society for Optical Engineering*, 5387:24–26, 2004.

[45] Masaki Sato and Hideki Yokobori, A. Toshimitsu Ozawa, Yoshihito Kamiyama, Takayuki Miyanaga, Toshiaki Beaumont, P. W. R. Sekine. Experimental study of repair efficiency for single-sided composite patches bonded to aircraft structural panels. *Advanced Composite Materials*, 11(1):51–59, 2002.

[46] Sarah (FAA) Seaton, Charles (FAA) and Richter. Nonconforming Composite Repairs : Case Study Analysis. *DOT/FAA/TC-14/20*, 2014.

[47] Brandon P. Smith, Mahdi Ashrafi, Mark E. Tuttle, and Santosh Devasia. Bondline Temperature Control for Joining Composites With an Embedded Heater. *Journal of Manufacturing Science and Engineering*, 138(2):021011, 2015.

[48] Ping Cheng Sung and Shih Chin Chang. The adhesive bonding with buckypaper-carbon nanotube/epoxy composite adhesives cured by Joule heating. *Carbon*, 91:215–223, 2015.

[49] Varaporn Tanrattanakul and Dumrong Jaroendee. Comparison between microwave and thermal curing of glass fiber-epoxy composites: Effect of microwave-heating cycle on mechanical properties. *Journal of Applied Polymer Science*, 102(2):1059–1070, 2006.

[50] Erik T. Thostenson and Tsu Wei Chou. Microwave and conventional curing of thick-section thermoset composite laminates: Experiment and simulation. *Polymer Composites*, 22(2):197–212, 2001.

[51] T.E. Twardowski, S.E. Lin, and P.H. Geil. Curing in Thick Composite Laminates: Experiment and Simulation. *Journal of Composite Materials*, 27(3):216–250, 1993.

[52] Daniel Walczyk, Jaron Kuppers, and Casey Hoffman. Curing and Consolidation of Advanced Thermoset Composite Laminate Parts by Pressing Between a Heated Mold and Customized Rubber-Faced Mold. *Journal of Manufacturing Science and Engineering*, 133(1):011002, 2011.

[53] B L Welch. The Generalization of ‘ Student ’ s ’ Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1):28–35, 1947.

[54] Torsten Windhorst and Gordon Blount. Carbon-carbon composites: a summary of recent developments and applications. *Materials & Design*, 18(1):11–15, 1997.

[55] Shuangjie Zhou and Martin C. Hawley. A study of microwave reaction rate enhancement effect in adhesive bonding of polymers and composites. *Composite Structures*, 61(4):303–309, 2003.

[56] L. Zhu and R. Pitchumani. Analysis of a process for curing composites by the use of embedded resistive heating elements. *Composites Science and Technology*, 60:2699–2712, 2000.

[57] Shannon A. Zirbel, Brian P. Trease, Spencer P. Magleby, and Larry L. Howell. Deployment methods for an origami-inspired rigid-foldable array. *Proceedings of the 40th Aerospace Mechanisms Symposium*, NASA Godda:189–194, 2014.

Appendix A

DESIGN DOCUMENT: A LOW COST AND OPEN SOURCE TEMPERATURE MONITORING AND CONTROL SYSTEM

A.1 Overall system description

The system described in this document is intended for controlling the bondline temperature for curing a high strength adhesive joint. The specific temperature profile required is typically specified by the adhesive manufacturer, with a ramp-up rate, time to hold temperature, and ramp-down rate. To accomplish this, the system must be able to accurately measure temperature, and use this to control power to the bond area.

While there are off-the-shelf temperature measurement and control systems, there are several advantages to a ground-up design. One driver is cost. With cheap and easy to use microcontrollers, designing a system from individual parts is no longer prohibitive. Additionally, writing custom software allows for greater flexibility and expandability in design. Additional features are simple to add, and using open-source software avoids licensing issues. A high-level diagram of this system can be seen in Fig. A.1.

The hardware for this system is based around an Arduino microcontroller, and both Arduino Mega 2560 and Arduino Due have been validated for use with this system. To gather temperature measurement, a printed circuit board was designed and produced that multiplexes up to 8 thermocouples, and amplifies this signal. This signal is captured in the Arduino A/D circuit. Within the Arduino firmware this signal is then digitally filtered at a high frequency to provide a smooth and accurate temperature signal. A heater controller is built into the Arduino firmware that uses the temperature measurement as input, and outputs a PWM signal. This PWM signal is then attached to solid state switches, which

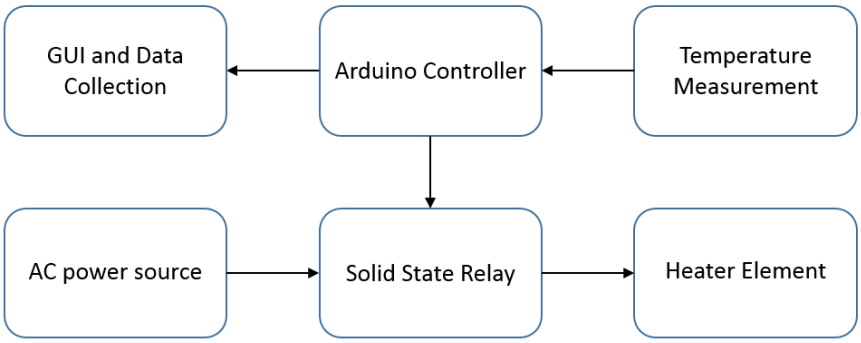


Figure A.1: A system level diagram showing all components.

modulate AC power to a physical heater.

For monitoring and data collection, the Arduino microcontroller is attached to a personal computer, using a graphical user interface (GUI) written in C# within the .NET framework. This GUI allows real-time monitoring of temperature measurement and control output, and has the option to store data to a file. Each of these system components will be described in more details in the following sections.

A.2 Measuring temperature

The goal of this system is to measure temperature accurately, quickly, and at a low cost. To sample thermocouples with appropriate resolution, it is necessary to amplify the signal, which then necessitates filtering. This entire design is visualized in Fig. A.2 and will be described in the following sections.

A.2.1 Thermocouple circuit board

To measure temperature through A/D sampling, it is necessary to have an amplifying circuit. There are many off-the-shelf chips that do this well, and for this system the AD595CQ was chosen due to its accuracy ($\pm$ 1 C°), cost, and form factor. This is an amplifier with a built

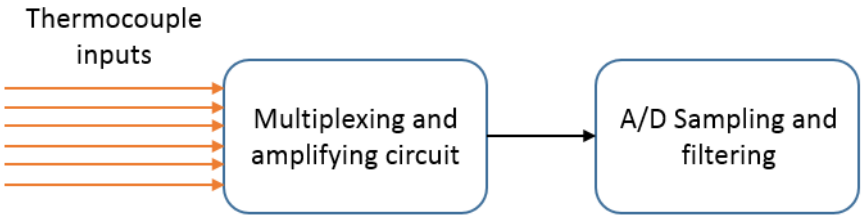


Figure A.2: Schematic of temperature measurement system.

in cold junction compensation, with a through hole configuration for circuit board mounting.

While the AD595CQ at around $25 per chip, is not expensive, it quickly becomes prohibitive if one were used for each thermocouple on a system with many thermocouples. For this reason, a multiplexer has been put in-line with this chip to connect up to 8 thermocouples to a single amplifier. The specific multiplexer chosen for this system is the MAXIM DG407, with is a dual 8-channel CMOS analog multiplexer. The advantage of using this chip is that with a single switch, it is possible to switch two signals at a time, which in this case are the negative and positive thermocouple terminals. Using these two chips, a circuit board was designed in EAGLE and printed. The design schematic can be seen in Fig. A.3, and a picture of a completed board can be seen in Fig. A.4.

A.2.2 Capturing temperature in software

The main loop performs multiple core functions at different rates. The reason for this is that some things, such as temperature measurement and checking for user input needs to be serviced more often than data capture and control. Having a fast rate of temperature measurement allows for a strong filter, while at the same time a quick response time to temperature change. The rates of these core function can be seen below in Fig. A.5.

The temperatures of all thermocouples are read sequentially through the built-in A/D circuit on the Arduino board. Each thermocouple is attached to its own digital filter, so

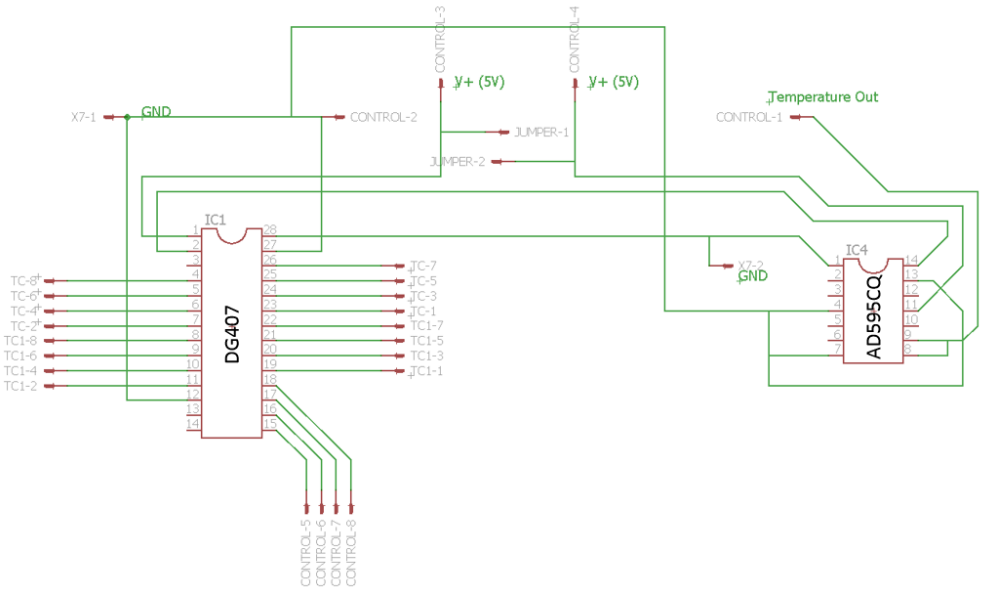


Figure A.3: EAGLE board schematic for multiplexer with thermocouple amplifier.

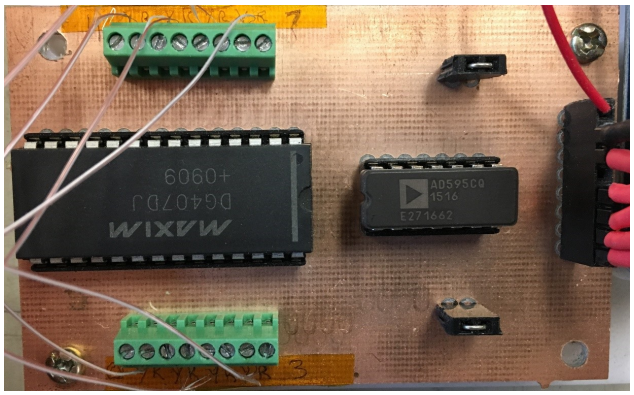


Figure A.4: Picture of a completed thermocouple multiplexing and amplification circuit board.

that as a new value is read, this value is added to the filter. When data is sent to the GUI, or control is calculated, the filtered value of each thermocouple is used. The filter used is a low-pass, second order difference equation defined as

Main loop	100 Hz
- Handle GUI input	100 Hz
- Read thermocouple estimates	100 Hz
- Update control law	4 Hz
- Send data to GUI	4 Hz

Figure A.5: Different rates of the core functions in the main loop of Arduino firmware.

$$y[n] = -\sum_{k=1}^{M} a_k y[n-k] + \sum_{k=0}^{N} b_k x[n-k], \tag{A.1}$$

where $y[n]$ is the filter value at instant n, and $x[n]$ is the temperature measurement at instant n, and the filter coefficients are defined by a and b. The filter coefficients were defined using MATLAB function BUTTER, where the cutoff frequency was chosen to be 1 Hz. This mean that variations that occur faster than a second should be attenuated, while variations happening slower will be visible in the signal.

A.3 Controlling temperature

This section is divided into two sub-parts. The first will provide high level-overview of the controller as implemented in software. The second part describes how that control signal is implemented in hardware.

A.3.1 Heater controller (software)

A temperature controller was designed to be able to follow a cure profile prescribed by an adhesive manufacturer. An example of a profile can be seen in Fig. A.6, which is a cure profile for HexPly 155 resin system made by HEXEL. Noteworthy is the slow trajectory relative to

the temperature measurement and control system. Since the trajectory is slow, a relatively simple control scheme will provide sufficient performance. In this case, a proportional and integral (PI) controller was chosen, with the option of a constant gain feedforward input. This control scheme is visualized in Fig. A.7. This control system was tuned to give the closed loop control system a natural frequency of around 2 minutes, which is more than sufficient to approximate the desired temperature profile.

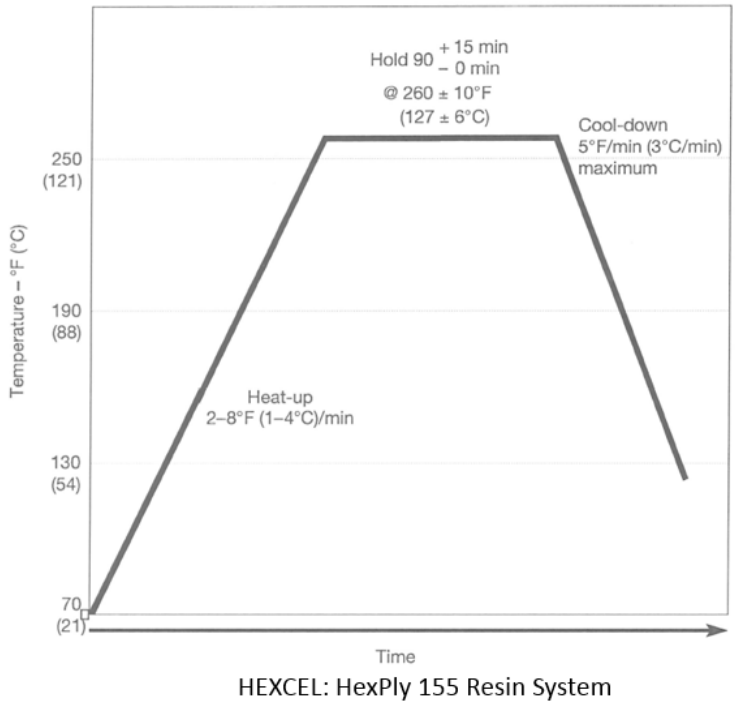


Figure A.6: Cure temperature profile for resin system.

To regulate power output from the controller, a common and simple scheme was chosen, which is pulse-width-modulation (PWM). The Arduino microcontroller has a variety of ways to output signal. One way would be to manually toggle a digital I/O pin between 5 volts and ground. This however requires a lot of overhead, and is difficult to maintain a consistent duty cycle with multiple threads. Instead, the Arduino timer hardware was used to set up

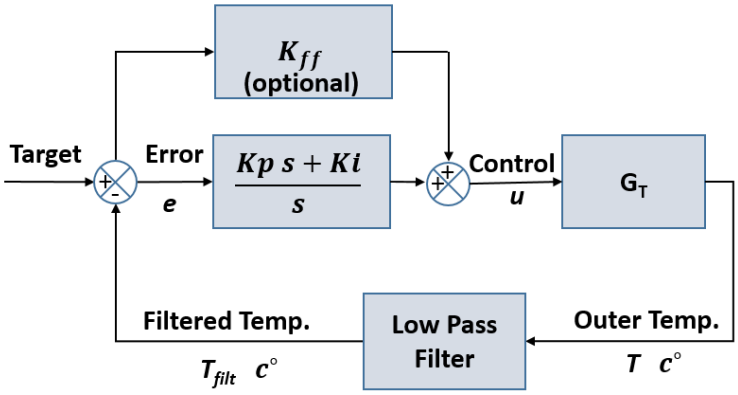


Figure A.7: Control system block diagram.

automatic PWM signals. For these times, all that is required is to set the frequency and duty cycle. The primary limitation of this approach is that frequencies in the desired range (less than 100 Hz) become limited by resolution. For this reason, a PWM frequency of 10 Hz was chosen.

A.3.2 Heater controller (hardware)

Power is delivered to the heater element through an AC power supply. Rather than adjust the AC voltage level, this power is modulated through solid state relays. The advantage of using solid state relays vs. mechanical relays is speed and durability. The specific switches chosen for this system are Omron G3NE-220TL. These operate at a nominal 5 volts, which is the digital output level of the Arduino Mega, have a switch on time of 1 ms, and can handle up to 20 Amps of output. This system was designed to work with multiple heating elements, which can be controlled independently. To accomplish this, and array of switches was made and hooked up to the Arduino controller, all connected to the same power source. This array of switches can be seen in Fig. A.8.

Figure A.8: Array of solid state relays for delivering power to heater elements. Software.

A.4 Software

The overall software layout can be seen in Fig. A.9. The separate pieces will be discussed in the following subsections.

A.4.1 Embedded software

The embedded software is the software written and written to the Arduino microcontroller. This software has an initialization function, which then leads into a main loop. This main loop is where all of the temperature collection, control, and recording happens.

Since the Arduino needs to communicate with the PC GUI, it is necessary to provide appropriate handshaking routines. In the main loop, the very first thing that happens is to check if there has been an input from the PC serial connection. If it finds something, it parses the input like a string, and then takes an action. These actions include turning on or off the controller, setting the thermocouples that are being measured and controlled from, and setting the number of controllers.

After handling PC input, the main loop then read all thermocouples, filters this input,

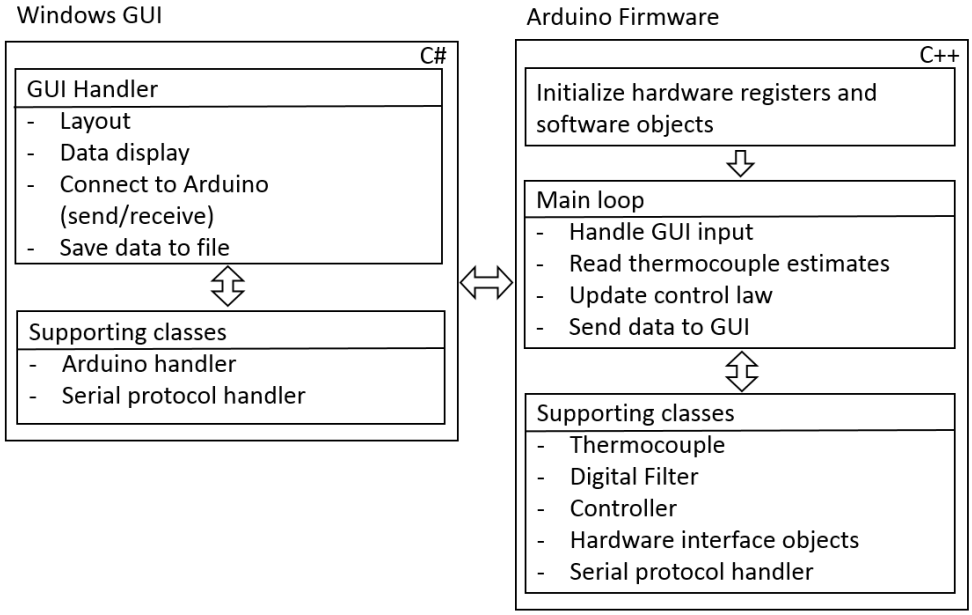


Figure A.9: Software architecture overview of GUI and Arduino firmware.

and then updates the control law. Last, the results and all data to be recorded is sent back to the PC using a data packet which will be described in a subsequent section.

A.4.2 Graphic user interface with data collection and display

To access the controller and save data in a user-friendly way, a GUI was written for Microsoft Windows sytems in C# in the .NET framework. One advantage of this approach is that this GUI can be transferred between computers with installing any additional software. A screen shot of this GUI can be seen in Fig. A.10.

A.4.3 Separate control board

It was discovered while running experiments that the output pin transitions from the PWM control creates power spikes in the Arduino microcontroller board. These spikes caused

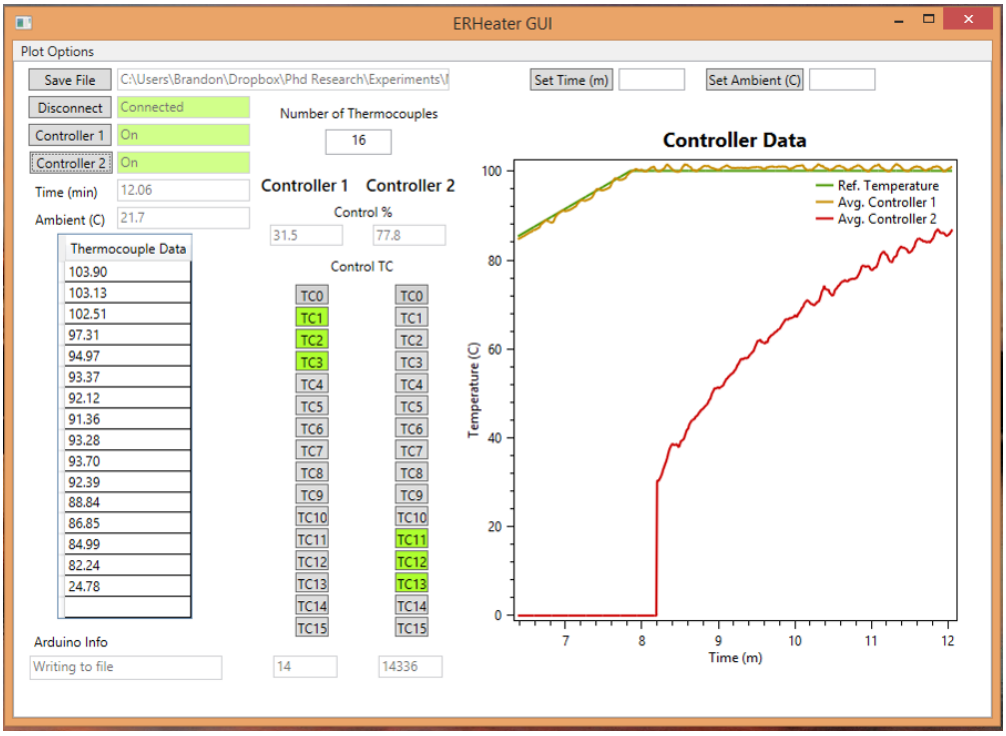


Figure A.10: Screen shot of the windows GUI written in C#.

misreads when they coincided with ADC reads. For this reason, a second Arduino board was used to output the PWM. This was accomplished by having the GUI receive the desired control signal from the main board that takes temperature measurement, and then calculates the control output values. The GUI then sends the control values to the control board. Decoupling the measurement and PWM control in this way allows the system to maintain clean temperature ADC reads.

A.4.4 Communication protocol

To communicate between the Arduino and Windows GUI a communication protocol was developed. Sending data from the Arduino board to the PC, this was done by creating a packet of data. At the front of each packet is an 8-bit checksum, followed the single byte

values '1', '2', '3', '4'. Following this key each piece of data is sent preceded by a key value to determine what data type it is so that the other side can decode it properly. This allows for integers, unsigned integers, and shorts to be sent and decoded efficiently.

Sending data from the PC to the Arduino board is simpler as it only needs to send commands and not long strings of data. The packet sent to the Arduino starts with a checksum byte, followed by a letter key to identify which command is being sent, followed by data needed for that command.

Using this bi-directional communication protocol, it is possible to communicate in both directions, while dropping any corrupted packets.

Appendix B

ARDUINO BASED CONTROL SYSTEM CODE

B.1 Main Arduino file

```
//Filename: multiZoneHeater.cpp

#include "DigFilter.h"
#include "multiZoneHeater.h"
#include "ThermoCoupleMux.h"
#include "PiController.h"
#include "TemperatureEstimator.h"
#include "BufferedSerial.h"
#include "BetterADC.h"

int16_t controlOn = FALSE;
PiController controllers[maxControlZones];
float tempAverage[maxControlZones];
float tempMax[maxControlZones];
float tempMin[maxControlZones];
int16_t CurrentOutput[maxControlZones];
uint16_t controllerTCs[maxControlZones];
ThermoCoupleMux TcHandler;
TemperatureEstimator tInnerEstimator;
int16_t tcEstRef = 0;  //Thermocouple used for estimating inner temperature
BufferedSerial serial = BufferedSerial(256, 256);
ByteBuffer send_buffer;

int16_t ambientTemperature;
int16_t manualAmbient = FALSE;
//int16_t testPinMain = 43;

/** ***************************************
 *  Setup function
 *
 * - Initialize serial port, handshake with GUI, and initialize send buffer.
 * - Initialize ADC hardware.
 * - Read and set initial thermocouple temperatures, and set ambient.
 * - Initialize estimator (not usually used).
 * - Initialize all controllers.
 ****************************************/
void setup()
{
  //
  // Setup serial communication with GUI
  //
```

```
serial.init(0, 250000);
// Send handshake signal
while (Serial.available() <= 0) {
    Serial.println("a");   // send a starting message
    delay(50);
}

//
// Initialize the send buffer that we will use to send data
//
send_buffer.init(200);

//
// Setup thermocouple measurements
//
#if defined(__SAM3X8E__)
  analogReadResolution(12);
#else
  analogReference(INTERNAL2V56); // Set ADC read to 0-2.56V resolution
  delay(5);
#endif

TcHandler = ThermoCoupleMux(numTCs, mvADCceiling, 0);
for (int16_t i = 0; i < 20; i++)
{
  TcHandler.ReadThermoCouples();
}

//
// Set ambient temperature
//
ambientTemperature = TcHandler.GetTCDataFiltered(0);

//
// Setup temperature estimator
//
float c[3] = {est_c1, est_c2, est_c3};
tInnerEstimator = TemperatureEstimator(c, 3, ambientTemperature, 0);

//
//Setup temperature contollers
//
for (int16_t i = 0; i < maxControlZones; i++)
{
    controllers[i] = PiController(targetDelay_ms*numDelaysForControl, cont_kp[i], cont_ki[i],
                          ffInv, ambientTemperature,minutesToHold,
                          rampUpRate,ssTemp,rampDownRate,controlType);
  controllerTCs[i] = 1; // Set control TCs to TC0
  tempAverage[i] = 0;
  tempMax[i] = 0;
  tempMin[i] = 0;
  CurrentOutput[i] = 0;
}
```

```
  // Initialize PWM
  delay(10);
  oldTime = micros();

}

/** ***************************************
 * Main loop function
 *
 * This loop runs the program. Reads thermocouples,
 * takes computer input, and runs controller
 ****************************************/
void loop()
{
  //digitalWrite(testPinMain,HIGH);
  //
  // For the reference outer thermocouple, estimate the inner temperature
  //
  //tInnerEstimator.UpdateEstimate(TcHandler.GetTCDataFiltered(1),TcHandler.GetTCDataFiltered(2));

  //
  // Take input from PC
  //
  handleComputerInput();

  //
  // Read all active thermocouples
  //
  TcHandler.ReadThermoCouples();


  //
  // Data collection and control loop
  //
  if (loopCount % numDelaysForControl == 0)
  {


    //
    // Update control reference targets. Controller 1 is master.
    //
    if (controlOn == TRUE)
    {
      controllers[0].UpdateReference(millis());
      for (int16_t i = 1; i < numControlZones; i++)
      {
        controllers[i].SetReference(controllers[0].GetRefTemp());
      }
    }

    int16_t num2avg = 0;
    for (int16_t zoneIdx = 0; zoneIdx < numControlZones; zoneIdx++)
    {
      num2avg = 0;
```

```
  tempAverage[zoneIdx] = 0;
  tempMax[zoneIdx] = 0;
  tempMin[zoneIdx] = 250;
  for (int16_t tcIdx = 0; tcIdx < numTCs; tcIdx++)
  {
    if ((controllerTCs[zoneIdx] & (1 << tcIdx)) == (1 << tcIdx))
    {
      float thisTemperature = TcHandler.GetTCDataFiltered(tcIdx);
      tempAverage[zoneIdx] += thisTemperature;
      if(thisTemperature > tempMax[zoneIdx])
      {
        tempMax[zoneIdx] = thisTemperature;
      }
      if(thisTemperature < tempMin[zoneIdx])
      {
        tempMin[zoneIdx] = thisTemperature;
      }
      num2avg++;
    }
  }
  if (num2avg < 1)
  {
    tempAverage[zoneIdx] = TcHandler.GetTCDataFiltered(0);
  }
  else
  {
    tempAverage[zoneIdx] = tempAverage[zoneIdx] / num2avg;
  }
  if (controlOn ==  TRUE)
  {
    if(controlOver == TCAVG)
    {
      controllers[zoneIdx].UpdateControl(tempAverage[zoneIdx]);
    }
    else if(controlOver == TCMAX)
    {
      controllers[zoneIdx].UpdateControl(tempMax[zoneIdx]);
    }
    else if(controlOver == TCMIN)
    {
      controllers[zoneIdx].UpdateControl(tempMin[zoneIdx]);
    }

  }
}

//
// Get new control output
//
if (controlOn == TRUE)
{
  for (int16_t i = 0; i < numControlZones; i++)
  {
    CurrentOutput[i] = controllers[i].GetOutput();
```

```
      }
    }

    //
    // Set PWM for controllers
    //
    OutPutControlToPWM(controlOn);

  } // Control Loop


  if (loopCount % numDelaysForData == 0)
  {
    //
    // Print out to serial port.
    //
    sendDataToSerial();
  }

  loopCount++;
  time1 = micros();
  //digitalWrite(testPinMain,LOW);
  while ( (time1 - oldTime) < (targetDelay_ms * 1000) )
  {
    //
    // Handle computer input while we are waiting
    //
    handleComputerInput();

    time1 = micros();
  }
  oldTime = time1;
}

/** ***************************************
 * Handles computer input. This input determines a variety of things.
 * - Sets controllers on and off.
 * - Sets which thermocouples to control on.
 * - Can set temperature controller thinks is ambient.
 * - Can set position along control profile.
 ****************************************/
void handleComputerInput(void)
{
  //
  // Computer input storage
  //
  static int16_t compInput  = 0;
  static const int16_t sizeOfStrIn = 17;
  // Used for taking string input from computer (serial port)
  static byte  stringIn[sizeOfStrIn];  // Enough space for 16 characters
  static byte  *pStringIn; // Used for taking

  //
  // Chech for toggle control on or off,
```

```
// and set control thermocouples
//
pStringIn = stringIn;
if (Serial.available())
{
  delay(5);
}
while (Serial.available())
{
  noInterrupts();
  if (pStringIn - stringIn < sizeOfStrIn)
  {
    *pStringIn++ = Serial.read();
  }
  else
  {
    Serial.read();
  }
  interrupts();
}

if (pStringIn > stringIn)
{
  byte checkSum = 0;
  for (int16_t idx = 0; idx < (pStringIn-stringIn-1); idx++)
  {
    checkSum += stringIn[idx];
  }
  int16_t passedCheckSum;

  if(checkSum == stringIn[pStringIn-stringIn-1])
  {
    stringIn[pStringIn-stringIn-1] = '\0';
    passedCheckSum = TRUE;
  }
  else
  {
    FALSE;
  }

  if (passedCheckSum)
  {

    // Separate input type from value
    byte inputType = stringIn[0];
    if ((pStringIn-stringIn-2) == 1) //Subtract inputType and checksum to find type
    {
      compInput = stringIn[1];
    }
    else if ((pStringIn-stringIn-2) == 2)
    {
      compInput = ((int16_t)stringIn[1]) + (((int16_t)stringIn[2]) << 8);
    }
    else // Something went wrong, discard input
```

```
{
  memset(stringIn, '\0', sizeOfStrIn);
  return;
}
// 'a' - controllers on and off
// 'b' - set thermocouples to control with
// 'c' - set time in controller
// 'd' - set ambient temperature in controller
// 'e' - set number of thermocouples
// 'f' - set number of control zones
// 'g' - set how to control over TCs (avg, max, etc)
// 'h' - set control type (RAMP, SINESWEEP,STEP,HOLD)
if (inputType == 'a')
{
  if (compInput == 0)
  {
    for (int16_t i = 0; i < numControlZones; i++)
    {
      controllers[i].TurnOff();
      CurrentOutput[i] = 0;
      controlOn = FALSE;
    }
    OutPutControlToPWM(FALSE);
  }
  else if (compInput == 1)
  {
    for (int16_t i = 0; i < numControlZones; i++)
    {
      // Initialize controllers, argument passes ambient temperature
      if (manualAmbient == TRUE)
      {
        controllers[i].InitController(ambientTemperature);
      }
      else
      {
        controllers[i].InitController(tempAverage[0]);
      }
    }
    controlOn = TRUE;
  }
}

else if (inputType == 'b')
{
  int16_t zoneToSet = compInput / 100;
  int16_t tcToSet = compInput % 100;
  if((zoneToSet >= 0) && (zoneToSet < maxControlZones) && (tcToSet >= 0) && (tcToSet < 16))
  {
    controllerTCs[zoneToSet] ^= 1 << (tcToSet);
  }

}
else if (inputType == 'c')
{
```

```
  controllers[0].SetControlStage(compInput);
}
else if (inputType == 'd')
{
  if ((compInput > 0) && (compInput < 150))
  {
    ambientTemperature = compInput;
    manualAmbient = TRUE;
    controllers[0].SetAmbient(compInput);
  }
}
else if (inputType == 'e')
{
  if ((compInput > 0) && (compInput < 17))
  {
    numTCs = compInput;
    TcHandler.SetNumberOfTC(numTCs);
    for (int16_t i = 0; i < 20; i++)
    {
      TcHandler.ReadThermoCouples();
    }
  }
}
else if (inputType == 'f')
{
  if ((compInput > 0) && (compInput < 7))
  {
    numControlZones = compInput;
  }
}
else if (inputType == 'g')
{
  if(compInput == 0)
  {
    controlOver = TCAVG;
  }
  if(compInput == 1)
  {
    controlOver = TCMAX;
  }
  if(compInput == 2)
  {
    controlOver = TCMIN;
  }
}
else if (inputType == 'h')
{
  int16_t old_controlType = controlType;
  if(compInput == 0)
  {
    controlType = RAMP;
  }
  if(compInput == 1)
  {
```

```
          controlType = SINESWEEP;
        }
        if(compInput == 2)
        {
          controlType = STEP;
        }
        if(compInput == 3)
        {
          controlType = HOLD;
        }
        if (controlType != old_controlType)
        {
          for (int16_t i = 0; i < maxControlZones; i++)
          {
            controllers[i].SetControlType(controlType);
          }
        }
      }
    }
  }
  //
  // At the end of processing string, clear string
  //
  memset(stringIn, '\0', sizeOfStrIn);
}

/** ***************************************
 * Send data to serial port
 ****************************************/
void sendDataToSerial(void)
{
  time1 = controllers[0].GetTime();
  if (!serial.isBusySending())
  {

    send_buffer.clear();

    //send_buffer.putInt(ErrorCode);
    send_buffer.putFloat(controllers[0].GetAmbient());

    send_buffer.putLong(time1);
    for (int16_t i = 0; i < numTCs; i++)
    {
      send_buffer.putFloat(TcHandler.GetTCDataFiltered(i));
    }

    send_buffer.putFloat(controllers[0].GetRefTemp());
    for (int16_t i = 0; i < numControlZones; i++)
    {
      if(controlOver == TCAVG)
      {
        send_buffer.putFloat(tempAverage[i]);
      }
      if(controlOver == TCMAX)
```

```
      {
        send_buffer.putFloat(tempMax[i]);
      }
      if(controlOver == TCMIN)
      {
        send_buffer.putFloat(tempMin[i]);
      }
      send_buffer.putInt(CurrentOutput[i]);
      send_buffer.putUInt(controllerTCs[i]);
    }
    serial.sendSerialPacket( &send_buffer );
  }

  while (serial.isBusySending())
  {
    serial.update();
  }
}

/** ****************************************
 * Send PWM data to slave
 *
 * This is legacy code. Data to slave board is now sent
 * directly from the GUI. This is because communication
 * between the boards was causing noise in the ADC read
 * line.
 ******************************************/
void OutPutControlToPWM(int16_t SendOutput)
{ /*
  byte low = 0;
  byte high = 0;
  Wire.beginTransmission(_i2caddr);
  if (SendOutput)
  {
    for (int i = 0; i < numControlZones; i++)
    {
      Wire.write((uint8_t)(CurrentOutput[i] >> 8));
      Wire.write((uint8_t)(CurrentOutput[i] & 255));

    }
  }
  else // Turning off PWM, send single value > 1000
  {

    Wire.write((uint8_t)(0x00));
    Wire.write((uint8_t)(0x00));
  }
  Wire.endTransmission();
 */
}
```

B.2 Main Arduino file header

```
// Filename: multiZoneHeater.h
```

```
#define TRUE 1
#define FALSE 0

#include "ByteBuffer.h"
#include "PiController.h"
//#include "Wire.h"


///* Types and Enums          */
enum tcType {TCAVG, TCMAX, TCMIN};

///* Declare Global variables */
uint32_t time1;
uint32_t oldTime;
const int32_t targetDelay_ms = 12;
const int16_t numDelaysForControl = 25;
const int32_t numDelaysForData = 25;
uint32_t loopCount = 0;
int16_t ErrorCode = 0;

// Temperature estimator constants
const float  est_c1 = 1.1901;
const float  est_c2 = 129.0375;
const float  est_c3 = -1.942022;

// Controller gains
//const float cont_kp = 209.0244;
//const float cont_ki = -208.0521;
const float cont_kp[6] = { 114.9663,  94.5808,  88.2111,  80.2145,  85.3578,  96.32598};
const float cont_ki[6] = {-114.4744, -94.3287, -88.0212, -80.0409, -85.1336, -96.16303};
//const float cont_kp[6] = { 98.1064,  104.1416,  129.1314,  102.3228,  95.7606,  96.32598}; //for 5 zones
//const float cont_ki[6] = {-97.9525, -103.9678, -128.8536, -102.1386, -95.5984, -96.16303}; //for 5 zones
//const float cont_kp[6] = { 129.1314, 129.1314,  129.1314,  129.1314,  129.1314,  129.1314}; //for 5
    zones
//const float cont_ki[6] = {-128.8536, -128.8536, -128.8536, -128.8536, -128.8536, -128.8536}; //for 5
    zones
const float ffInv = 0;
const int32_t minutesToHold = 120;
const float rampUpRate = 3; // In degrees per minute
const int16_t ssTemp = 171;
const int16_t rampDownRate = 3; // positive means slope down
int16_t controlType = RAMP; //  RAMP  //  SINESWEEP  //  STEP
int16_t numTCs = 16; // Default value
int16_t numControlZones = 2; //Default value
const int16_t maxControlZones = 6;
//byte _i2caddr = 8;

// Other user settings
const int16_t controlOnEstimate = FALSE;
tcType controlOver = TCAVG;


#if defined(__SAM3X8E__)
  const int16_t mvADCceiling = 3300;
```

```
#else
  const int16_t mvADCceiling = 2560;
#endif

//-----------------------------------------------
//-----------------------------------------------

void turnOffPin(void);
void sendDataToSerial(void);
void handlePacket(ByteBuffer* packet);
```

B.3 Arduino controller library

```
//-------------------------------------------------------------------------------
//
// FILE:         PiController.cpp
// DESCRIPTION: PI controller
//
//-------------------------------------------------------------------------------
//

#include "PiController.h"
#include "math.h"
#include "Arduino.h"

#define PI 3.14159265
#define STEPVALUE 500
/**
 * Constructor
 *
 * @param - int sampleTimems
 * @param - float a0
 * @param - float a1
 * @param - float ambientTemp
 */

PiController::PiController(int16_t sampleTimems, float a0, float a1, float ffInv,
                                    float ambientTemp, int32_t minutesToHold,
                                    float rampUpRate, int16_t ssTemp, float rampDownRate, int16_t
                                        controlType)
{
        // Initialize variables
        InitController(ambientTemp);

        this->sampleTimems = sampleTimems;
        this->a0 = a0;
        this->a1 = a1;
        this->ffInv = ffInv;
        this->ambientTemp = ambientTemp;
        this->minutesToHold = minutesToHold;
        this->rampUpRate = rampUpRate;
        this->ssTemp = ssTemp;
        this->rampDownRate = rampDownRate;
        this->controlType = controlType;
```

```
        //
        // Designed with butter(2,(1/8)/2)
        //
        float bArray [3] = {0.008442693, 0.01688539, 0.008442693};
        float aArray [2] = {-1.723776, 0.7575469};

        //
        // Designed with butter(2,(1/10)/2)
        //
//      float bArray [3] = {0.005542717, 0.01108543, 0.005542717};
//      float aArray [2] = {-1.778632, 0.8008026};

        cfilter = DigFilter(bArray, 3, aArray, 2);
        rampDown = 0;

}

/**
 * Initialize controller
 *
 * @param - float ambientTemp
 */
void PiController::InitController(float ambientTemp)
{
        this->ambientTemp = ambientTemp;
        errOld = 0;
        newError = 0;
        outputOld = 0;
        output = 0;
        rampDown = 0;
        stimulus = 0;
        if((ambientTemp > 200) || (ambientTemp < 10))
        {
                this->ambientTemp = 20;
        }
        SetReference(this->ambientTemp);
        timeAtTemp = 0;
        errOld = reftemp - (this->ambientTemp);
        TurnOn();
        cfilter.Init();
        sinestate = 0;
        lastRefTempTime = 0;
        timeOffset = millis();

}

/**
 * Calculate and send control
 *
 * @param - float currentTemp
 */
void PiController::UpdateControl(float currentTemp)
{
```

```
        if (controlOn == 1)
        {
                // Calculate control
                newError = reftemp - currentTemp;
                output = a0 * newError + a1 * errOld;
                output += outputOld;
                //Protect against saturation
                if (output > 1000)
                {
                        output = 1000;
                }
                if (output < -1000)
                {
                        output = -1000;
                }
                errOld = newError;
                outputOld = output;
                cfilter.UpdateFilter(output);
        }

}

/**
 * Update reference temperature
 */
void PiController::UpdateReference(int32_t timeIn)
{
        int32_t offsetTime = timeIn-timeOffset;

        if (controlOn == 1)
        {
                if (lastRefTempTime == 0)
                {
                        lastRefTempTime = offsetTime;
                }
                //
                // Control ramp up, hold, then ramp down
                //
                if (controlType == RAMP)
                {
                        int32_t deltatime = offsetTime - lastRefTempTime;
                        float rampAdjust;
                        if ((deltatime > 2000) || (deltatime < 0))
                        {
                                deltatime = 0;
                        }
                        if (rampDown) //Ramp down
                        {
                                rampAdjust = -((float)(rampDownRate * deltatime)) / 1000 / 60;
                                if ((reftemp < 40) || (rampDownRate == 0))
                                {
                                        controlOn = 0;
                                        output = 0;
                                        cfilter.Init();
```

```
                        reftemp = 20;
                }
        }
        else if (reftemp < ssTemp) //Ramp up
        {
                rampAdjust = ((float)(rampUpRate * deltatime)) / 1000 / 60;
        }
        else // Hold temp
        {
                rampAdjust = 0;
                if (timeAtTemp == 0)
                {
                        timeAtTemp = offsetTime;
                }
                if ((offsetTime - timeAtTemp) > (minutesToHold * 60 * 1000))
                {
                        rampDown = 1;
                }
        }
        if ((rampAdjust > 1) || (rampAdjust < -1))
        {
                rampAdjust = 0;
                // Throw error
        }
        reftemp += rampAdjust;
        lastRefTempTime = offsetTime;
}
//
// Apply sine wave at various frequencies
//
else if (controlType == SINESWEEP)
{
        // Begin timeing first time through
        if (sinestate == 0)
        {
            timeAtTemp = millis();
            sinestate = 1;
        }
        // Soak to 100 deg C for 30 minutes
        else if (sinestate == 1)
        {
                reftemp = 100;
                if ((millis() - timeAtTemp) > (((int32_t)(30*60))*1000))
                {
                        sinestate = 2;
                        timeAtTemp = millis();
                }

        }
        // Send sine wave as reference signal for all specified frequencies
        else
        {
                if(sinestate < (numFreqs + 2))
                {
```

```
                                int32_t tcurrent = millis() - timeAtTemp;
                                stimulus = 20 * sin(2*PI*tcurrent/1000/Tperiod[sinestate-2]);
                                reftemp = 100 + stimulus;
                                if ((millis() - timeAtTemp) > (((int32_t)
                                        (Tcycles[sinestate-2]*Tperiod[sinestate-2]))*1000))
                                {
                                        sinestate++;
                                        timeAtTemp = millis();
                                }
                        }
                        else
                        {
                                controlOn = 0;
                                output = 0;
                                stimulus = 0;
                                cfilter.Init();
                                reftemp = 20;
                        }
                }

        }
        //
        // Command a step
        //
        else if (controlType == STEP)
        {
                cfilter.Init(STEPVALUE);
                ffInv = 0;
        }
        //
        // Control to and hold reference temperature
        //
        else if (controlType == HOLD)
        {
                // Don't need to do anything
        }
        else
        {
                // Do nothing I guess. Should throw an error.
        }
    }

}

/**
 * Set reference temperature
 *
 * @param - float temp
 */
void PiController::SetReference(float temp)
{
        reftemp = temp;

}
```

```
/**
 * Get current control output
 *
 * @return int output
 */
int16_t PiController::GetOutput()
{
        float filteredOutput = cfilter.GetCurrentValue();
        // NEED TO FIX AMBIENTTEMP BEFORE TURNING ON FF CONTROL
        //filteredOutput += (reftemp-ambientTemp) * ffInv;

        if (filteredOutput > 1000)
        {
                filteredOutput = 1000;
        }
        if (filteredOutput < 0)
        {
                filteredOutput = 0;
        }

        if (controlType == STEP)
        {
                if (controlOn == 1)
                {
                        return (int16_t)STEPVALUE;
                }
        }
        return (int16_t)filteredOutput;
}

/**
 * Get reference temperature
 *
 * @return - float reference temperature
 */
float PiController::GetRefTemp()
{
        return reftemp;
}

/**
 * Turn control on
 */
void PiController::TurnOn()
{
        controlOn = 1;
}

/**
 * Turn control off
 */
void PiController::TurnOff()
{
```

```
        controlOn = 0;
        output = 0;
        cfilter.Init();
}

/**
 * Take in time in minutes and set the time on the controller
 */
void PiController::SetControlStage(int16_t minutesIn)
{
        timeOffset = (int32_t)(millis())-((int32_t)(minutesIn))*60*1000;
        lastRefTempTime = 0;

        // Define transition points in temperature profile
        int16_t p1 = (ssTemp-ambientTemp)/rampUpRate;
        int16_t p2 = p1 + minutesToHold;
        int16_t p3 = p2 + (ssTemp-40)/rampDownRate;
        // Time is during ramp up
        if ((minutesIn > 0) && (minutesIn < p1))
        {
                rampDown = 0;
                reftemp = ambientTemp + rampUpRate*minutesIn;
                timeAtTemp = 0;
        }
        // Time is during hold
        else if ((minutesIn >= p1) && (minutesIn < p2))
        {
                rampDown = 0;
                reftemp = ssTemp;
                timeAtTemp = p1*60*1000;
        }
        // Time is during ramp down
        else if ((minutesIn >= p2) && (minutesIn < p3))
        {
                rampDown = 1;
                reftemp = ssTemp-(minutesIn-p2)*rampDownRate;
                timeAtTemp = p1*60*1000;
        }
}

/**
 * Get time in ms
 */
uint32_t PiController::GetTime(void)
{
        return (uint32_t)(millis()-timeOffset);
}

/**
 * Set ambient temperature
 */
void PiController::SetAmbient(int16_t ambientManual)
{
        ambientTemp = (float)ambientManual;
```

```
}

/**
 * Get ambient temperature
 */
float PiController::GetAmbient(void)
{
        return ambientTemp;
}

/**
 * Set control type
 */
void PiController::SetControlType(int16_t controlType)
{
        this->controlType = controlType;
}
```

B.4 Arduino controller library header

```
//-------------------------------------------------------------------------
//
// FILE:         PiController.h
// DESCRIPTION: PI controller
//
//-------------------------------------------------------------------------
//
#ifndef __PICONTROLLER_H__
#define __PICONTROLLER_H__

#include "DigFilter.h"


const int16_t Tperiod[] = {1500, 1024,  750,  500, 350, 250, 128, 64, 32, 16, 8,   4};
const int16_t Tcycles[] = {   2,    4,    4,   6,   6,   8,   8, 10, 10, 12, 14, 16};
const int16_t numFreqs = 9;

typedef enum
{
    RAMP = 0,
    SINESWEEP,
    STEP,
        HOLD
} controlTypes;

class PiController
{
  public:
    PiController(){};
    PiController(int16_t sampleTimems, float a0, float a1, float ffInv,
                    float ambientTemp, int32_t minutesToHold,
                    float rampUpRate, int16_t ssTemp, float rampDownRate, int16_t controlType);
    void InitController(float ambientTemp);
    void UpdateReference(int32_t timeIn);
```

```
    void UpdateControl(float newError);
    void SetReference(float temp);
    int16_t GetOutput();
    float GetRefTemp();
    void TurnOn();
    void TurnOff();
    void SetControlStage(int16_t minutesIn);
    uint32_t GetTime(void);
    void SetAmbient(int16_t ambientManual);
    float GetAmbient(void);
    void SetControlType(int16_t controlType);


  private:
    float reftemp;
    float errOld;
    float outputOld;
    float output;
    float newError;
    float ambientTemp;
    float ffInv;
    int16_t   sampleTimems;
    float a0;
    float a1;
    int16_t   controlOn;
    int32_t  timeAtTemp;
    int16_t   rampDown;
    DigFilter cfilter;
    int32_t minutesToHold;
    int16_t rampUpRate; // In degrees per minute
    int16_t ssTemp;
    int16_t rampDownRate;
    int16_t controlType;
    int16_t   sinestate;
    float stimulus;
    int32_t lastRefTempTime;
    int32_t timeOffset;
};

#endif // __PICONTROLLER_H__
```

B.5 Arduino temperature estimator (Chapter 2)

```
//-------------------------------------------------------------------------
//
// FILE:         TemperatureEstimator.cpp
// DESCRIPTION: Implements a temperature estimator
//
//-------------------------------------------------------------------------
//

#include "TemperatureEstimator.h"
#include "string.h"
```

```
/**
 * Constuctor
 */
TemperatureEstimator::TemperatureEstimator(float c[], int16_t cLen, float initialTemp, float diffOffset)
{
        //
        // See UpdateEstimate for description for meaning of c
        //
        if (cLen > numCvals)
        {
                // The current design has 3 coefficients
                return;
        }
        cLength = cLen;
        for (int16_t i = 0; i < cLen; i++)
        {
                cVals[i] = c[i];
        }
        lastValue = initialTemp;
        this->diffOffset = diffOffset;
        //
        // Filter for derivative of t_o
        // from butter(2,(1/25)/2)
//      float bArray [3] = {0.0009446918, 0.001889384, 0.0009446918};
//      float aArray [2] = {-1.9111971, 0.9149758};
//      diffFilter = DigFilter(bArray, 3, aArray, 2);
//      diffFilter.Init();
}

/**
 * Upate the filter with the new value as input
 *
 * @param float toIn - To measured
 * @param float t1In - T1 measured
 *
 */
void TemperatureEstimator::UpdateEstimate(float toIn, float t1In, float toDiffNew)
{
//      toDiffhat = (toIn-lastValue);
//      diffFilter.UpdateFilter(toDiff);
//      lastValue = toIn;
        toDiff = toDiffNew - diffOffset;
        //
        // Ti = c0 * To + c1 * To^2 + c2 * Todiff
        //
        currentEstimate = cVals[0] * toIn + cVals[1] * toDiff + cVals[2] * t1In;
//      currentEstimate = cVals[0] * toIn + cVals[1] * diffFilter.GetCurrentValue() + cVals[2] * t1In;
//      currentEstimate = cVals[0] * toIn + cVals[1] * toDiffHat + cVals[2] * t1In;

}


/**
```

```
 * Return current value of the filter
 *
 * @return float returnValue
 */
float TemperatureEstimator::GetCurrentValue(void)
{
        if ((currentEstimate > 300) || (currentEstimate < 10))
        {
                return 20;
        }
        return currentEstimate;
}

/**
 * Return current value of the To Diff
 *
 * @return float returnValue
 */
float TemperatureEstimator::GetToDiff(void)
{
        return toDiff;
//      return diffFilter.GetCurrentValue();
}
```

B.6 Arduino temperature estimator (Chapter 2) header

```
//-------------------------------------------------------------------------
//
// FILE:         TemperatureEstimator.h
// DESCRIPTION: Temperature estimator class
//
//-------------------------------------------------------------------------
//

#ifndef __TEMPERATUREESTIMATOR_H__
#define __TEMPERATUREESTIMATOR_H__

//#include "DigFilter.h"
#include "Arduino.h"

const int16_t numCvals = 3;

class TemperatureEstimator
{
  public:
        TemperatureEstimator(){};
        TemperatureEstimator(float c[], int16_t cLen, float initialTemp, float diffOffset);
        void UpdateEstimate(float toIn, float t1In, float toDiff);
        float GetCurrentValue(void);
        float GetToDiff(void);


  private:
        float cVals[numCvals];
```

```
        int16_t    cLength;
        float currentEstimate;
//      DigFilter diffFilter;
        float temperatureOffset;
        int16_t sampletime_ms;
        float lastValue;
        float toDiff;
        float diffOffset;

};

#endif // __TEMPERATUREESTIMATOR_H__
```

B.7 Windows GUI main window xaml

```
<Window x:Class="MultiZone_OffBoardControl.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:oxy="http://oxyplot.org/wpf"
        xmlns:local="clr-namespace:MultiZone_OffBoardControl"
        mc:Ignorable="d"
        Title="ERHeater_GUI" Height="700
________" Width="950"
        ResizeMode="CanResize"
        WindowState="Maximized">
    <Grid x:Name="grid" Margin="0,0,0,0">
        <DockPanel Name="mainDockPanel">
            <Menu Name="mainUpperMenu" DockPanel.Dock="Top" Background="White">
                <MenuItem Header="_Plot_Options">
                    <MenuItem Header="_Profile_Plot"
Click="SwitchViewProfilePlot_Click" />
                    <MenuItem Header="_1-D_Plot" Click="SwitchView1D_Click" />
                    <MenuItem Header="_2-D_Plot" Click="SwitchView2D_Click" />
                    <MenuItem Header="_Clear_plot" Click="ClearPlot_Click"/>
                </MenuItem>
                <MenuItem Name="uploadCodeMenu" Header="_Upload_Code">
                    <!--<MenuItem Header="_Upload_Code"
Click="UploadCode_Click"/>-->
                </MenuItem>
                <MenuItem Header="_Control_Type">
                    <MenuItem Header="_RAMP" Click="ControlRamp_Click" />
                    <MenuItem Header="_SINESWEEP" Click="ControlSineSweep_Click" />
                    <MenuItem Header="_STEP" Click="ControlStep_Click" />
                    <MenuItem Header="_HOLD" Click="ControlHold_Click"/>
                </MenuItem>
            </Menu>
            <!--<TextBox AcceptsReturn="True" />-->
        </DockPanel>
        <TextBox x:Name="contstatus" HorizontalAlignment="Left" Height="20"
Margin="95,116,0,0" TextWrapping="Wrap" Text="Off" VerticalAlignment="Top"
 Width="100" IsEnabled="False" Background="LightSalmon" />
        <Button x:Name="savefilebutton" Content="Save_File"
```

```
HorizontalAlignment="Left" Margin="14,28,0,0" VerticalAlignment="Top" Width="75"
Click="savefilebutton_Click" Height="35" />
        <TextBox x:Name="savefiletextbox" HorizontalAlignment="Left" Height="20"
 Margin="94,35,0,0" TextWrapping="NoWrap"  VerticalAlignment="Top" Width="300"
IsEnabled="False"/>
        <Button x:Name="connectArduino" Content="Connect"
HorizontalAlignment="Left" Margin="15,68,0,0" VerticalAlignment="Top" Width="75"
Click="connectArduino_Click" Height="35" />
        <Button x:Name="contbutton" Content="Controller"
HorizontalAlignment="Left" Margin="15,108,0,0" VerticalAlignment="Top" Width="75"
Click="contbutton_Click" Height="35" />
        <TextBox x:Name="connectedstatus" HorizontalAlignment="Left" Height="20"
 Margin="95,76,0,0" TextWrapping="Wrap" Text="Not Connected"
VerticalAlignment="Top" Width="100" Background="LightSalmon"  IsEnabled="False"
/>
        <TextBox x:Name="errorBox" HorizontalAlignment="Left" Height="25"
Margin="14,621,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="306"
IsEnabled="False" Text="{Binding MyArduino.ErrorBox}"/>
        <oxy:PlotView Model="{Binding MyArduino.MyModel}"
RenderTransformOrigin="0.73,0.32" Margin="467,75,11,28" />
        <Label x:Name="label" Content="Time (min)" HorizontalAlignment="Left"
VerticalAlignment="Top" Margin="15,148,0,0"/>
        <TextBox x:Name="timeBox" HorizontalAlignment="Left" Height="20"
Margin="94,148,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="100"
 IsEnabled="False" Text="{Binding MyArduino.CurrentTime}"/>
        <DataGrid x:Name="dataGrid" HorizontalAlignment="Left"
Margin="15,225,0,0" VerticalAlignment="Top" Height="Auto" Width="Auto"
 ItemsSource="{Binding MyArduino.TemperatureData}"
RenderTransformOrigin="0.583,1.615" />
        <DataGrid x:Name="dataGrid2" HorizontalAlignment="Left"
Margin="112,225,0,0" VerticalAlignment="Top" Height="Auto" Width="Auto"
 ItemsSource="{Binding MyArduino.ControllerData}"
RenderTransformOrigin="0.583,1.615" />
        <Label x:Name="label1" Content="Thermocouples" HorizontalAlignment="Left"
Margin="219,52,0,0" VerticalAlignment="Top" Height="23" Width="101"/>
        <TextBox x:Name="thermoCoupleText" HorizontalAlignment="Left" Height="20"
Margin="234,76,0,0" TextWrapping="Wrap" TextAlignment="Center"
VerticalAlignment="Top" Width="60" KeyDown="thermoCoupleText_TextChanged"
IsEnabled="False"/>
        <Label x:Name="label2" Content="Control TC for each zone"
HorizontalAlignment="Left" Margin="249,148,0,0" VerticalAlignment="Top" />
        <!--<Button x:Name="clearButton" Content="Clear Plot"
HorizontalAlignment="Right" Margin="0,0,150,15" VerticalAlignment="Bottom"
Width="74" Click="ClearPlot_Click" RenderTransformOrigin="0.518,0.767"/>
        <Button x:Name="switchPlotButton" Content="Across Bondline"
HorizontalAlignment="Right" Margin="0,0,244,15" VerticalAlignment="Bottom"
Width="106" Click="SwitchView_Click" RenderTransformOrigin="0.518,0.767"/>-->
        <Label x:Name="label3" Content="Arduino Info" HorizontalAlignment="Left"
 Margin="13,591,0,0" VerticalAlignment="Top"/>
        <Button x:Name="setTimeButton" Content="Set Time (m)"
HorizontalAlignment="Left" Margin="658,25,0,0" VerticalAlignment="Top" Width="75"
Click="setTime_Click" Height="35" />
        <Button x:Name="setAmbientButton" Content="Set Ambient (C)"
HorizontalAlignment="Left" Margin="471,25,0,0" VerticalAlignment="Top" Width="88"
```

```
Click="setAmbient_Click" Height="35" />
        <TextBox x:Name="setTimeTextBox" HorizontalAlignment="Left" Height="35"
Margin="738,25,0,0" TextWrapping="Wrap" TextAlignment="Center"
 VerticalAlignment="Top" Width="60" IsEnabled="False"/>
        <TextBox x:Name="setAmbientTextBox" HorizontalAlignment="Left"
Height="35" Margin="564,25,0,0" TextWrapping="Wrap" TextAlignment="Center"
VerticalAlignment="Top" Width="60" IsEnabled="False"/>
        <Label x:Name="labelAmbient" Content="Ambient (C)"
HorizontalAlignment="Left" VerticalAlignment="Top" Margin="14,169,0,0"
Height="32"/>
        <TextBox x:Name="timeBox_Copy" HorizontalAlignment="Left" Height="20"
Margin="94,174,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="100"
IsEnabled="False" Text="{Binding MyArduino.AmbientTemperature}"/>
        <Button x:Name="clearErrorBox" Content="Clear" HorizontalAlignment="Left"
Margin="95,581,0,0" VerticalAlignment="Top" Width="51"
Click="clearErrorBox_Click" Height="35" />
        <TextBox x:Name="numHeatZoneText" HorizontalAlignment="Left" Height="20"
Margin="330,76,0,0" TextWrapping="Wrap" TextAlignment="Center"
VerticalAlignment="Top" Width="60" KeyDown="heatZoneText_TextChanged"
IsEnabled="False"/>
        <Label x:Name="heatZoneLabel" Content="Heat Zones"
HorizontalAlignment="Left" Margin="325,52,0,0" VerticalAlignment="Top"
Height="23" Width="69"/>
        <Label x:Name="label5" Content="Zone Pins: 11, 12, 13, 5, 2, 3"
HorizontalAlignment="Left" Margin="161,590,0,0" VerticalAlignment="Top"
Width="159"/>
        <Button x:Name="tcToAvg" Content="AVERAGE" HorizontalAlignment="Left"
Margin="214,116,0,0" VerticalAlignment="Top" Width="64" Click="tcToAvg_Click"
 Height="35" />
        <Button x:Name="tcToMax" Content="MAX" HorizontalAlignment="Left"
Margin="299,116,0,0" VerticalAlignment="Top" Width="45" Click="tcToMax_Click"
Height="35" />
        <CheckBox x:Name="checkBox" Content="Off-Board Control"
IsChecked="{Binding MyArduino.OffBoardControl}" Margin="337,590,473,2"/>
        <Button x:Name="reconnectSlaveButton" Content="Reconnect Slave"
HorizontalAlignment="Left" Margin="337,611,0,0" VerticalAlignment="Top"
Width="115" Click="reconnectSlave_Click" Height="35"
RenderTransformOrigin="0.235,-0.057" />
        <TextBox x:Name="targetBox" HorizontalAlignment="Left" Height="20"
Margin="94,200,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="100"
 IsEnabled="False" Text="{Binding MyArduino.TargetTemperature}"/>
        <Label x:Name="labelAmbient_Copy" Content="Target (C)"
HorizontalAlignment="Left" VerticalAlignment="Top" Margin="14,195,0,0"
 Height="32"/>
        <Button x:Name="tcToMin" Content="MIN" HorizontalAlignment="Left"
Margin="364,116,0,0" VerticalAlignment="Top" Width="46" Click="tcToMin_Click"
 Height="35" />
    </Grid>
</Window>
```

B.8 Windows GUI main window xaml.cs

```
// Filename: MainWindow.xaml.cs
```

```
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.IO;
using Microsoft.Win32;

namespace MultiZone_OffBoardControl
{

    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public bool controllerOn = false;
        public bool connectedToBoard = false;
        private StreamWriter myStream;
        public ArduinoConnect _arduinoBoard;
        private string initialThermocouples = "16";
        private string initialHeatZones = "2";

        private List<Button> controllerTCForControl = new List<Button>();

        public ArduinoConnect MyArduino
        {
            get { return _arduinoBoard; }
            set
            {
                if (_arduinoBoard != value)
                {
                    _arduinoBoard = value;
                }
            }
        }

        public MainWindow()
        {


            InitializeComponent();

            //Add found hex files to Upload Code Menu
            List<string> hexFiles = GetListOfHexFiles();
            for (int i = 0; i < hexFiles.Count; i++)
            {
                MenuItem newMenuItem = new MenuItem();
                newMenuItem.Header = hexFiles[i];
                newMenuItem.Click += new RoutedEventHandler(UploadCode_Click);
                uploadCodeMenu.Items.Add(newMenuItem);
            }
```

```
    DataContext = this;
    this.MyArduino = new ArduinoConnect(Int32.Parse(initialThermocouples),
                                        Int32.Parse(initialHeatZones));
    this.thermoCoupleText.Text = initialThermocouples;
    this.numHeatZoneText.Text = initialHeatZones;

    UpdateTCControlButtons();
}

private void UpdateTCControlButtons()
{
    //
    // Clear any currently existing buttons
    //
    if (controllerTCForControl.Count > 0)
    {
        foreach (Button cButtons in controllerTCForControl)
        {
            this.grid.Children.Remove(cButtons);
        }
        controllerTCForControl.Clear();
    }
    int idxTotal = 0;
    for (int iZone = 0; iZone < MyArduino.NumberOfHeatZones; iZone++)
    {
        for (int iTC = 0; iTC < MyArduino.NumberOfThermocouples; iTC++)
        {
            controllerTCForControl.Add(new Button());
            controllerTCForControl[idxTotal].Content = "TC" + iTC;
            controllerTCForControl[idxTotal].Name = "contTCbutton" + iTC + "zone" + iZone;
            controllerTCForControl[idxTotal].HorizontalAlignment = 0;
            controllerTCForControl[idxTotal].VerticalAlignment = 0;
            controllerTCForControl[idxTotal].Width = 30;
            controllerTCForControl[idxTotal].Height = 18;
            controllerTCForControl[idxTotal].Click += new RoutedEventHandler(
                controllerTCSelect_Click);
            controllerTCForControl[idxTotal].Margin = new Thickness(230 + iZone * 32, 200 + iTC *
                20, 0, 0);
            if (iTC == 0)
            {
                controllerTCForControl[idxTotal].Background = Brushes.GreenYellow;
            }

            this.grid.Children.Add(controllerTCForControl[idxTotal]);
            idxTotal++;
        }
    }


}
```

```
private List<string> GetListOfHexFiles()
{
    string[] fileList = Directory.GetFiles(@"..\..\avrdude", "*.cpp.hex");
    List<string> hexFiles = new List<string>(fileList);
    for (int i = 0; i < hexFiles.Count; i++)
    {
        hexFiles[i] = System.IO.Path.GetFileNameWithoutExtension(hexFiles[i]);
        hexFiles[i] = System.IO.Path.GetFileNameWithoutExtension(hexFiles[i]);
    }
    return hexFiles;
}

private void contbutton_Click(object sender, RoutedEventArgs e)
{
    if (connectedToBoard)
    {
        if (controllerOn == false)
        {
            if (MyArduino.savingFile == false)
            {
                MessageBox.Show("Don't forget to Save!", "Confirm", MessageBoxButton.OK);
            }

            byte[] outputStr = new byte[] { (byte)'a', 1 };
            MyArduino.WriteToBoard("MASTER", outputStr, 2);
            controllerOn = true;
            this.contstatus.Text = "On";
            this.contstatus.Background = Brushes.GreenYellow;
            setAmbientTextBox.IsEnabled = false;
            setTimeTextBox.IsEnabled = true;

        }
        else
        {
            byte[] outputStr = new byte[] { (byte)'a', 0 };
            MyArduino.WriteToBoard("MASTER", outputStr, 2);
            controllerOn = false;
            this.contstatus.Text = "Off";
            this.contstatus.Background = Brushes.LightSalmon;
            setAmbientTextBox.IsEnabled = true;
            setTimeTextBox.IsEnabled = false;
        }
    }
}


private void savefilebutton_Click(object sender, RoutedEventArgs e)
{
    if (MyArduino.savingFile == false)
    {
        SaveFileDialog saveFileDialog1 = new SaveFileDialog();

        saveFileDialog1.Filter = "txt files (*.txt)|*.txt";
        saveFileDialog1.FilterIndex = 2;
```

```
        saveFileDialog1.RestoreDirectory = true;

        if (saveFileDialog1.ShowDialog() == true)
        {
            savefiletextbox.Text = saveFileDialog1.FileName;
            myStream = new StreamWriter(saveFileDialog1.FileName);

            if (myStream != null)
            {
                bool streamSet = MyArduino.BeginSavingFile(myStream);
                if(streamSet)
                {
                    savefilebutton.Content = "Stop Saving";
                }
            }

        }
    }
    else
    {
        savefilebutton.Content = "Save File";
        savefiletextbox.Text = "";
        MyArduino.StopSavingFile();
    }

}

private void connectArduino_Click(object sender, RoutedEventArgs e)
{
    StringComparer stringComparer = StringComparer.OrdinalIgnoreCase;

    // If not connected, try to connect
    if (connectedToBoard == false)
    {
        connectedstatus.Text = "Trying...";
        for (int numTries = 0; numTries < 5; numTries++)
        {
            if (MyArduino.Connect() == true) // Able to connect
            {
                UpdateTCControlButtons();
                connectedToBoard = true;
                connectedstatus.Text = "Connected";
                connectedstatus.Background = Brushes.GreenYellow;
                connectArduino.Content = "Disconnect";
                thermoCoupleText.IsEnabled = true;
                numHeatZoneText.IsEnabled = true;
                setAmbientTextBox.IsEnabled = true;
                break;
            }
        }
        if (connectedToBoard == false)
        {
            connectedstatus.Text = "Not connected";
        }
```

```
    }
    else  // Disconnect
    {
        DisconnectArduino();
    }
}

private void DisconnectArduino()
{
    MyArduino.Disconnect();
    connectedToBoard = false;
    connectedstatus.Text = "Not connected";
    connectedstatus.Background = Brushes.LightSalmon;
    connectArduino.Content = "Connect";
    controllerOn = false;
    contstatus.Text = "Off";
    contstatus.Background = Brushes.LightSalmon;
    thermoCoupleText.Text = initialThermocouples;
    MyArduino.NumberOfThermocouples = Int32.Parse(initialThermocouples);
    thermoCoupleText.IsEnabled = false;
    numHeatZoneText.Text = initialHeatZones;
    MyArduino.NumberOfHeatZones = Int32.Parse(initialHeatZones);
    UpdateTCControlButtons();
    numHeatZoneText.IsEnabled = false;
    setAmbientTextBox.IsEnabled = false;
    setTimeTextBox.IsEnabled = false;
}

protected override void OnClosed(EventArgs e)
{
    MyArduino.Disconnect();
    if (myStream != null)
    {
        myStream.WriteLine("");
        myStream.Close();
    }
    base.OnClosed(e);
    Application.Current.Shutdown();
}




private void controllerTCSelect_Click(object sender, RoutedEventArgs e)
{
    if (connectedToBoard)
    {
        Button buttonClicked = (Button)sender;
        string bname = buttonClicked.Name;
        string tcStr = "contTCbutton";
        string zoneStr = "zone";
        int tcLoc1 = tcStr.Length;
        int tcLoc2 = bname.IndexOf(zoneStr, 0);
        int zoneLoc1 = bname.IndexOf(zoneStr, 0) + zoneStr.Length;
```

```
        int zoneLoc2 = bname.Length;

        int buttonNum = Int32.Parse(bname.Substring(tcLoc1, tcLoc2 - tcLoc1));
        int zoneNum = Int32.Parse(bname.Substring(zoneLoc1, zoneLoc2 - zoneLoc1));

        if (buttonClicked.Background == Brushes.GreenYellow)
        {
            buttonClicked.Background = Brushes.LightGray;
        }
        else
        {
            buttonClicked.Background = Brushes.GreenYellow;
        }
        //string outstring = "b" + Convert.ToString((zoneNum * 100) + buttonNum);
        byte[] numInBytes = BitConverter.GetBytes((UInt16)((zoneNum * 100) + buttonNum));
        byte[] outputStr = new byte[] { (byte)'b', numInBytes[0], numInBytes[1] };
        MyArduino.WriteToBoard("MASTER", outputStr, 3);
    }
}

private void ClearPlot_Click(object sender, RoutedEventArgs e)
{
    MyArduino.ClearPlot();
}

private void SwitchViewProfilePlot_Click(object sender, RoutedEventArgs e)
{
    ViewType currentPlot = MyArduino.CurrentPlotType();
    if (currentPlot != ViewType.TimePlot)
    {
        MyArduino.InitiatePlot(ViewType.TimePlot);
    }
}

private void SwitchView1D_Click(object sender, RoutedEventArgs e)
{
    ViewType currentPlot = MyArduino.CurrentPlotType();
    if (currentPlot != ViewType.AcrossBond)
    {
        MyArduino.InitiatePlot(ViewType.AcrossBond);
    }
}

private void SwitchView2D_Click(object sender, RoutedEventArgs e)
{
    ViewType currentPlot = MyArduino.CurrentPlotType();
    if (currentPlot != ViewType.ContourPlot)
    {
        MyArduino.InitiatePlot(ViewType.ContourPlot);
        MyArduino.MyModel.Axes[0].Title = "";
        //  switchPlotButton.Content = "Time Plot";

    }
}
```

```
private void setTime_Click(object sender, RoutedEventArgs e)
{
    if (connectedToBoard)
    {
        try
        {
            int numIn = Int32.Parse(setTimeTextBox.Text);
            if ((numIn >= 0) && (numIn < 400))
            {
                //string stringout = "c" + Convert.ToString(numIn);
                byte[] numInBytes = BitConverter.GetBytes((Int16)numIn);
                byte[] outputStr = new byte[] { (byte)'c', numInBytes [0], numInBytes [1]};
                MyArduino.WriteToBoard("MASTER", outputStr, 3);
            }
        }
        catch (Exception)
        {
            MyArduino.ErrorBox = "Could not write time";
        }

    }
}

private void setAmbient_Click(object sender, RoutedEventArgs e)
{
    if (connectedToBoard)
    {
        try
        {
            int numIn = Int32.Parse(setAmbientTextBox.Text);
            if ((numIn >= 0) && (numIn < 150))
            {
                string stringout = "d" + Convert.ToString(numIn);
                byte[] numInBytes = BitConverter.GetBytes((Int16)numIn);
                byte[] outputStr = new byte[] { (byte)'d', numInBytes[0], numInBytes[1] };
                MyArduino.WriteToBoard("MASTER", outputStr, 3);
            }
        }
        catch (Exception)
        {
            MyArduino.ErrorBox = "Could not write Ambient";
        }
    }
}

private void clearErrorBox_Click(object sender, RoutedEventArgs e)
{
    MyArduino.ErrorBox = "";
}


private void UploadCode_Click(object sender, RoutedEventArgs e)
{
```

```
    MenuItem itemClicked = (MenuItem)sender;
    DisconnectArduino();
    MyArduino.UploadHex((string)itemClicked.Header);
}

private void thermoCoupleText_TextChanged(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter)
    {
        //enter key is down
        try
        {
            int numIn = Int32.Parse(thermoCoupleText.Text);
            if (numIn > 0 && numIn < 17)
            {
                MyArduino.SetNumberOfThermocouples(numIn);
                UpdateTCControlButtons();
            }
            else
            {
                // Send error somewhere
            }
        }
        catch (Exception)
        {
            // Send error somewhere
        }
    }
}

private void heatZoneText_TextChanged(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter)
    {
        try
        {
            int numIn = Int32.Parse(numHeatZoneText.Text);
            if (numIn > 0 && numIn < 7)
            {
                MyArduino.SetNumberOfControlZones(numIn);
                UpdateTCControlButtons();
            }
            else
            {
                // Send error somewhere
            }
        }
        catch (Exception)
        {
            // Send error somewhere
        }
    }
}
```

```
        private void tcToAvg_Click(object sender, RoutedEventArgs e)
        {
            byte[] outputStr = new byte[] { (byte)'g', 0 };
            MyArduino.WriteToBoard("MASTER", outputStr, 2);
        }

        private void tcToMax_Click(object sender, RoutedEventArgs e)
        {
            byte[] outputStr = new byte[] { (byte)'g', 1 };
            MyArduino.WriteToBoard("MASTER", outputStr, 2);
        }

        private void reconnectSlave_Click(object sender, RoutedEventArgs e)
        {
            MyArduino.ReconnectSlave = true;
        }

        private void ControlRamp_Click(object sender, RoutedEventArgs e)
        {
            byte[] outputStr = new byte[] { (byte)'h', 0 };
            MyArduino.WriteToBoard("MASTER", outputStr, 2);
        }

        private void ControlSineSweep_Click(object sender, RoutedEventArgs e)
        {
            byte[] outputStr = new byte[] { (byte)'h', 1 };
            MyArduino.WriteToBoard("MASTER", outputStr, 2);
        }

        private void ControlStep_Click(object sender, RoutedEventArgs e)
        {
            byte[] outputStr = new byte[] { (byte)'h', 2 };
            MyArduino.WriteToBoard("MASTER", outputStr, 2);
        }

        private void ControlHold_Click(object sender, RoutedEventArgs e)
        {
            byte[] outputStr = new byte[] { (byte)'h', 3 };
            MyArduino.WriteToBoard("MASTER", outputStr, 2);
        }

        private void tcToMin_Click(object sender, RoutedEventArgs e)
        {
            byte[] outputStr = new byte[] { (byte)'g', 2 };
            MyArduino.WriteToBoard("MASTER", outputStr, 2);
        }
    }
}
```

B.9 Windows GUI Arduino Connection Class

```
// Filename: ArduinoConnect.cs

using System;
```

```
using System.IO.Ports;
using System.Threading;
using System.IO;
using System.ComponentModel;
using OxyPlot;
using OxyPlot.Series;
using OxyPlot.Axes;
using System.Collections.Generic;
using System.Data;
using System.Diagnostics;

namespace MultiZone_OffBoardControl
{
    public enum ViewType
    {
        TimePlot = 0,
        AcrossBond = 1,
        ContourPlot = 2,
    };

    public class ArduinoConnect : INotifyPropertyChanged
    {
        // variable and objects used in class
        public event PropertyChangedEventHandler PropertyChanged;
        private SerialPort _serialPort;
        private SerialPort _serialPortSlave;

        private bool _continue;
        private byte[] recentPacket = null;
        private float[] recentPacketUnpacked = null;
        private StreamWriter myStream;
        public bool savingFile = false;
        SerialProtocolBS serialBS = new SerialProtocolBS();
        private int[] idxToPlot;
        private string[] plotLabels;
        private bool firstRead = true;
        private Thread readThread;
        private Object lockObject = new Object();
        private ViewType myPlotView = ViewType.TimePlot;
        private List<LineSeries> timeSeries = new List<LineSeries>();
        public int maxDataPoints = 5000;

        // variables to store INotifyPropertyChanged information
        private string _currentTime = "At constructor";
        private DataTable _temperatureData;
        private DataTable _controllerData;
        private string _errorBox;
        private int _numberOfThermocouples; //Should be reset on construction anyway
        private string _ambientTemperature = "20";
        private string _targetTemperature = "20";
        private int _numberOfHeatZones; //Should be reset on construction anyway
        private bool _offBoardControl = true;
        private bool _reconnectSlave = false;
```

```
/*******************************************
This section implements INotifyPropertyChanged
*******************************************/
protected virtual void OnPropertyChanged(string propertyName)
{
    PropertyChangedEventHandler handler = PropertyChanged;
    if (handler != null) handler(this, new PropertyChangedEventArgs(propertyName));
}

public PlotModel MyModel { get; set; }

public string CurrentTime
{
    get { return _currentTime; }
    set
    {
        if (_currentTime != value)
        {
            _currentTime = value;
            OnPropertyChanged("CurrentTime");
        }
    }
}

public DataTable TemperatureData
{
    get { return _temperatureData; }
    set
    {
        if (_temperatureData != value)
        {
            _temperatureData = value;
            OnPropertyChanged("TemperatureData");
        }
    }
}

public DataTable ControllerData
{
    get { return _controllerData; }
    set
    {
        if (_controllerData != value)
        {
            _controllerData = value;
            OnPropertyChanged("ControllerData");
        }
    }
}

public string ErrorBox
{
    get { return _errorBox; }
```

```
    set
    {
        if (_errorBox != value)
        {
            _errorBox = value;
            OnPropertyChanged("ErrorBox");
        }
    }
}

public int NumberOfThermocouples
{
    get { return _numberOfThermocouples; }
    set
    {
        if (_numberOfThermocouples != value)
        {
            _numberOfThermocouples = value;
        }
    }
}

public string AmbientTemperature
{
    get { return _ambientTemperature; }
    set
    {
        if (_ambientTemperature != value)
        {
            _ambientTemperature = value;
            OnPropertyChanged("AmbientTemperature");
        }
    }
}

public string TargetTemperature
{
    get { return _targetTemperature; }
    set
    {
        if (_targetTemperature != value)
        {
            _targetTemperature = value;
            OnPropertyChanged("TargetTemperature");
        }
    }
}

public int NumberOfHeatZones
{
    get { return _numberOfHeatZones; }
    set
    {
        if (_numberOfHeatZones != value)
```

```
        {
            _numberOfHeatZones = value;
        }
    }
}

public bool OffBoardControl
{
    get { return _offBoardControl; }
    set
    {
        if (_offBoardControl != value)
        {
            _offBoardControl = value;
            OnPropertyChanged("OffBoardControl");
        }
    }
}

public bool ReconnectSlave
{
    get { return _reconnectSlave; }
    set
    {
        if (_reconnectSlave != value)
        {
            _reconnectSlave = value;
            OnPropertyChanged("ReconnectSlave");
        }
    }
}
/*******************************
Begin class function definitions
*******************************/

/*
/ Class constructor
/ Sets up serial port
/ Sets up Plotmodel MyModel
**/
public ArduinoConnect(Int32 numTC, Int32 numZone)
{
    CurrentTime = "Set at Arduino Init";
    _serialPort = null;
    _serialPortSlave =null;

    MyModel = new PlotModel {};
    MyModel.Series.Add(new LineSeries());
    MyModel.InvalidatePlot(true);

    NumberOfThermocouples = numTC;
    NumberOfHeatZones = numZone;

    ErrorBox = "Initialized Arduino object";
```

```
} // public ArduinoConnect()

/*
/ public bool Connect - uses serial port connection and connects to
/ Arduino board. Also initializes plotting.
/
/ Arguments: none
/
/ Returns: bool - true if connection made, false if failed
**/
public bool Connect()
{
    bool foundMainPort = false;
    bool foundSlavePort = false;

    // Close any open ports
    Disconnect();

    CurrentTime = "Set at ArduinoConnect";
    readThread = new Thread(new ThreadStart(Read));

    ClearPlot();
    InitiatePlot(myPlotView);

    for (int i = 0; i < 15; i++)
    {
        string comname = "COM" + i;
        try
        {

            SerialPort _serialPortTest = new SerialPort();
            SetupNewPort(ref _serialPortTest);

            _serialPortTest.PortName = comname;
            _serialPortTest.Open();
            int boardType = 0;
            int tries = 0;
            while (((boardType != 'a') && (boardType != 'b')) && (tries < 10))
            {
                tries++;
                try
                {
                    boardType = _serialPortTest.ReadChar();
                } catch { }
            }

            if (boardType == 'a')
            {

                _serialPort = _serialPortTest;
                _serialPortTest = null;
                _serialPort.Write("a");
                while (_serialPort.BytesToRead > 0)
```

```
            {
                _serialPort.ReadByte();
            }
            foundMainPort = true;
        }
        else if (boardType == 'b')
        {
            _serialPortSlave = _serialPortTest;
            _serialPortTest = null;
            _serialPortSlave.Write("b");
            while (_serialPortSlave.BytesToRead > 0)
            {
                _serialPortSlave.ReadByte();
            }
            foundSlavePort = true;
        }

    }
    catch { }

    if(((!OffBoardControl) && foundMainPort) || (foundMainPort && foundSlavePort))
    {
        // Make sure receiving packets
        for (int numTries = 0; numTries < 5; numTries++)
        {
            byte[] packet;
            try
            {
                lock (lockObject)
                {
                    packet = serialBS.GetPacket(ref _serialPort);
                    if (serialBS.PacketChecksum(packet) == true)
                    {
                        firstRead = true;
                        _continue = true;
                        readThread.Start();
                        ErrorBox = "Arduino Connected";
                        return true;
                    }
                }
            }
            catch
            {
            }
        }
    }
}

if (foundMainPort)
{
    ErrorBox = "Slave connect fail";
}
else if (foundSlavePort)
{
```

```
        ErrorBox = "Master connect fail";
    }
    else
    {
        ErrorBox = "No connection";
    }
    return false;
} // public bool Connect(ref TextBox errorBoxRef)

public bool ReconnectSlaveFunc()
{
    int boardType = 0;
    if (_serialPortSlave != null)
    {
       // _continue = false;
        try
        {
            _serialPortSlave.Close();
        }
        catch { }

        try
        {
            _serialPortSlave.Open();
            int tries = 0;
            while ((boardType != 'b') && (tries < 10))
            {
                tries++;
                try
                {
                    boardType = _serialPortSlave.ReadChar();
                }
                catch { }
            }
        }
        catch { }

        if (boardType == 'b')
        {
            _serialPortSlave.Write("b");
            while (_serialPortSlave.BytesToRead > 0)
            {
                _serialPortSlave.ReadByte();
            }
            ErrorBox = "Reconnected slave";
            return true;
        }
        else
        {
            ErrorBox = "Failed to reconnect slave";
            return false;
        }
    }
    return false;
```

```
}


/*
/ public void BeginSavingFile(StreamWriter stream)
/ Sets up ability to save to a file, and turns switch on
/
/ Arguments: StreamWriter stream - stream to save information
/
/ Returns: bool - if true, stream passed was not null
**/
public bool BeginSavingFile(StreamWriter stream)
{
    if (stream != null)
    {
        if (myStream != null)
        {
            myStream.Close();
        }
        myStream = stream;
        savingFile = true;
        return true;
    }
    return false;
}

/*
/ public void StopSavingFile(void)
/ stop saving file
/
/ Arguments:none
/
/ Returns: none
**/
public void StopSavingFile()
{
    if (myStream != null)
    {
        myStream.Close();
    }
    savingFile = false;
}
/*
/ public void Disconnect()
/ Disconnects from Board. Writes 0 to board first to turn off
/ any controllers that are turned on.
/
/ Arguments:none
/
/ Returns: none
**/
public void Disconnect()
{
    _continue = false;
```

```
    if(_serialPort != null)
    {
        if (_serialPort.IsOpen)
        {
            try
            {
                byte[] outputStr = new byte[] { (byte)'a', 0 };
                WriteToBoard("MASTER",outputStr,2);
            }
            catch (Exception)
            {
                // Failed
            }
            Thread.Sleep(100);
            try
            {
                _serialPort.Close();
            } catch { }
            _serialPort = null;
        }
    }
    if(_serialPortSlave != null)
    {
        if (_serialPortSlave.IsOpen)
        {
            try
            {
                byte[] outArray = new byte[] { (byte)'c', 0, 0};
                WriteToBoard("SLAVE", outArray, 3);
                _serialPortSlave.Close();
            }
            catch { }
        }
    }

} // public void Disconnect()

/*
/ public void Read()
/ This is meant to be run in a thread that continually  looks
/ for a packet sent from the Serial port, plots it, and writes
/ it to a file if saveFile is turned on.
/
/ Arguments:none
/
/ Returns: none
**/
public void Read()
{
    while (_continue)
    {
        try
        {
            if (_serialPort.IsOpen)
```

```
{
    byte[] packet;
    if (firstRead)
    {
        lock (lockObject)
        {
            try
            {
                serialBS.GetPacket(ref _serialPort);
                ErrorBox = "Getting packets";
            }
            catch (TimeoutException)
            {
                // Happens every time, not worth recording
                //ErrorBox = "Timed out on first packet";
            }
            catch (Exception)
            {
                ErrorBox = "Error on getting first packet";
            }

        }
        firstRead = false;
    }

    lock (lockObject)
    {
        packet = serialBS.GetPacket(ref _serialPort);
    }
    recentPacket = packet;

    if (packet != null)
    {
        if (serialBS.PacketChecksum(packet) == false)
        {
            ErrorBox = "Packet failed checksum";
        }
        else
        {

            recentPacketUnpacked = serialBS.Unpack(packet);
            if (recentPacketUnpacked == null)
            {
                throw new Exception("Unpacked packet contained null");
            }
            CurrentTime = (recentPacketUnpacked[1] / 1000 / 60).ToString("F2");


            if (savingFile)
            {
                string strout = string.Join(", ", recentPacketUnpacked);
                DateTimeOffset currentTime = DateTimeOffset.Now;
                strout = strout + ", " + currentTime.ToString("hh:mm:ss.fff");
```

```
    if (myStream != null)
    {
        myStream.WriteLine(strout);
    }
    else
    {
        ErrorBox = "savingFile_true,_myStream_null";
    }

}

//
// Store all time data (up to maxDataPoints)
//
for (int i = 0; i < idxToPlot.Length; i++)
{
    DataPoint thisPoint = new DataPoint(recentPacketUnpacked[1] / 1000 /
        60,
                                                recentPacketUnpacked[
                                                    idxToPlot[i]]);
    timeSeries[i].Points.Add(thisPoint);
    if (timeSeries[i].Points.Count > maxDataPoints)
    {
        timeSeries[i].Points.RemoveAt(0);
    }

    //
    // Add to MyModel if in TimePlot mode
    //
    if (myPlotView == ViewType.TimePlot)
    {
        (MyModel.Series[i] as LineSeries).Points.Add(thisPoint);
        if ((MyModel.Series[i] as LineSeries).Points.Count > maxDataPoints
            )
        {
            (MyModel.Series[i] as LineSeries).Points.RemoveAt(0); //remove
                first point
            timeSeries[i].Points.RemoveAt(0);
        }
    }
}

//
// Add current TC info if in AcrossBond mode
//
if (myPlotView == ViewType.AcrossBond)
{
    if ((MyModel.Series[0] as LineSeries).Points.Count > 0)
    {
        (MyModel.Series[0] as LineSeries).Points.Clear();
    }
    for (int i = 0; i < _numberOfThermocouples; i++)
    {
        (MyModel.Series[0] as
```

```
            LineSeries).Points.Add(new DataPoint(i, recentPacketUnpacked[2 + i]));
                }
            }

            if (myPlotView == ViewType.ContourPlot)
            {
                if (_numberOfThermocouples == 16)
                {
                    MyModel.Series.Clear();
                    MyModel.Axes.Clear();
                    MyModel.Axes.Add(new LinearColorAxis
                    {
                        Position = OxyPlot.Axes.AxisPosition.Right,
                        Palette = OxyPalettes.Jet(500),
                        Title = "Temperature (C)",
                        //  HighColor = OxyColors.Gray,
                        //  LowColor = OxyColors.Black
                    });
                    int idxContour = 0;
                    var dataHM = new double[4, 4];
                    for (int j = 0; j < _numberOfThermocouples / 4; j++)
                    {
                        for (int k = 0; k < _numberOfThermocouples / 4; k++)
                        {
                            dataHM[k, j] = recentPacketUnpacked[2 + idxContour];
                            idxContour++;
                        }
                    }
                    MyModel.Series.Add(new HeatMapSeries
{ X0 = 0, X1 = 3, Y0 = 0, Y1 = 3, Data = dataHM });
                }
            }
            MyModel.InvalidatePlot(true); // Refresh plot

            //
            // Update Temperature data for TC grid on GUI
            //
            DataTable tempData = new DataTable();
            tempData.Columns.Add("TC_Data");
            for (int i = 0; i < _numberOfThermocouples; i++)
            {
                tempData.Rows.Add(recentPacketUnpacked[2 + i].ToString("F2"));
            }
            TemperatureData = tempData;

            //
            // Update Controller data for grid on GUI
            //
            DataTable contData = new DataTable();
            contData.Columns.Add("Cont_%");
            contData.Columns.Add("TC_Flag");
            Int16 [] ControlArray = new Int16[_numberOfHeatZones];
            for (int i = 0; i < _numberOfHeatZones; i++)
            {
```

```
                    ControlArray[i] = (Int16)
(recentPacketUnpacked[_numberOfThermocouples + 4 + (i * 3)]);
                    contData.Rows.Add((recentPacketUnpacked[_numberOfThermocouples + 4 + (
                        i * 3)] /
10).ToString("F1"),
                                        Convert.ToString(((uint)
recentPacketUnpacked[_numberOfThermocouples + 5 + (i * 3)]),16));
                }
                if(OffBoardControl)
                {
                    if (ReconnectSlave)
                    {
                        ReconnectSlave = !(ReconnectSlaveFunc()); // passes back true if
                            reconnected
                    }
                    if (ReconnectSlave == false)
                    {
                        byte[] outArray = new byte[(_numberOfHeatZones * 2) + 1];
                        outArray[0] = (byte)'c';
                        Buffer.BlockCopy(ControlArray, 0, outArray, 1, (_numberOfHeatZones
                            * 2));
                        WriteToBoard("SLAVE", outArray, (_numberOfHeatZones * 2) + 1);
                        try
                        {
                            if (_serialPortSlave.ReadChar() != 'b')
                            {
                                ReconnectSlave = true;
                            }
                        }
                        catch { }
                    }

                }
                ControllerData = contData;

                //
                // Update control percentages
                // Update TCs to control over for GUI. Gives int to be converted to binary
                //
                AmbientTemperature = recentPacketUnpacked[0].ToString("F1");
                TargetTemperature = recentPacketUnpacked[2+ _numberOfThermocouples].
                    ToString("F1");
            }
            Thread.Sleep(50);
        }



    }
}
catch (TimeoutException)
{
    // Happens on startup, not worth recording
    //ErrorBox = "Timed out on Read";
```

```
        }
        catch (Exception e)
        {
            ErrorBox = "Exception at Read: " + e.Message;
        }

    }
} // public void Read()

/*
/public void WriteToBoard(string boardType, byte[]  outString, int count)
/
/ Arguments:string string boardType - "MASTER" or "SLAVE"
/    byte[]  outString - buffer of bytes to be sent
/    int count - length of output buffer
/
/ Returns: none
**/
public void WriteToBoard(string boardType, byte[]  outString, int count)
{
    if (_continue)
    {
        lock (lockObject)
        {
            // Do a checksum and add this to a new output buffer
            byte checkSum = 0;
            for (int idx = 0; idx < count; idx++)
            {
                checkSum += outString[idx];
            }
            byte[] outWithCheckSum = new byte[count + 1];
            Buffer.BlockCopy(outString, 0, outWithCheckSum, 0, count);
            outWithCheckSum[count] = checkSum;

            // Send output with checksum to specified board
            if (boardType == "MASTER")
            {
                _serialPort.Write(outWithCheckSum, 0, count + 1);                    }
            if (boardType == "SLAVE")
            {
                _serialPortSlave.Write(outWithCheckSum, 0, count + 1);
            }
        }
    }
}

/*
/public void ClearPlot()
/ Clears current plot as well as the stored time plot information
/
/ Arguments: none
/
/ Returns: none
```

```
**/
public void ClearPlot()
{
    try
    {
        if (idxToPlot != null)
        {
            for (int i = 0; i < idxToPlot.Length; i++)
            {
                (MyModel.Series[i] as LineSeries).Points.Clear(); //remove first point
                if (timeSeries.Count > 0)
                {
                    timeSeries[i].Points.Clear();
                }
            }
            MyModel.InvalidatePlot(true);
        }
    }
    catch (Exception)
    {
        // Failed to clear plot
    }

}

/*
/public void InitiatePlot(ViewType view)
/ Initiates plot to specified view. Can also be used to
/ switch plot types
/
/ Arguments: ViewType view - view type to used in plotting
/
/ Returns: none
**/
public void InitiatePlot(ViewType view)
{
    myPlotView = view;

    if (myPlotView == ViewType.TimePlot)
    {
        MyModel.Title = "Controller Data";
        MyModel.Axes.Clear();
        MyModel.Series.Clear();
        idxToPlot = new int[_numberOfHeatZones + 1];
        plotLabels = new string[_numberOfHeatZones + 1];

        idxToPlot[0] = 2 + _numberOfThermocouples;
        plotLabels[0] = "Ref. Temperature";
        for (int i = 0; i < _numberOfHeatZones; i++)
        {
            idxToPlot[i + 1] = 3 + _numberOfThermocouples + i * 3;
            plotLabels[i + 1] = "Avg. Controller " + (i + 1).ToString();
        }
```

```
    if (timeSeries.Count == idxToPlot.Length)
    {
        for (int i = 0; i < idxToPlot.Length; i++)
        {
            MyModel.Series.Add(timeSeries[i]);
            MyModel.Series[i].Title = plotLabels[i];
        }

    }
    else
    {
        if (timeSeries.Count > 0)
        {
            timeSeries.Clear();
        }
        for (int i = 0; i < idxToPlot.Length; i++)
        {

            timeSeries.Add(new LineSeries());
            MyModel.Series.Add(new LineSeries());
            MyModel.Series[i].Title = plotLabels[i];
        }
    }
    MyModel.InvalidatePlot(true);
    MyModel.Axes[0].Title = "Time (m)";
    MyModel.Axes[1].Title = "Temperature (C)";
}
else if (myPlotView == ViewType.AcrossBond)
{
    MyModel.Title = "1-D Temperature";
    MyModel.Axes.Clear();
    MyModel.Series.Clear();
    MyModel.Series.Add(new LineSeries());
    MyModel.Series[0].Title = "Thermocouples";
    MyModel.InvalidatePlot(true);
    MyModel.Axes[0].Title = "TC #";
    MyModel.Axes[1].Title = "Temperature (C)";
}
else if (myPlotView == ViewType.ContourPlot)
{
    MyModel.Title = "2-D Temperature";

    MyModel.Series.Clear();
    MyModel.Axes.Add(new LinearColorAxis
    {
        Position = OxyPlot.Axes.AxisPosition.Right,
        Palette = OxyPalettes.Jet(500),
        Title = "Temperature (C)",
      //  HighColor = OxyColors.Gray,
      //  LowColor = OxyColors.Black
    });
}

MyModel.InvalidatePlot(true);
```

```
}

/*
/public ViewType CurrentPlotType()
/ Wrapper to return current view type
/
/ Arguments: none
/
/ Returns: ViewType - current view type
**/
public ViewType CurrentPlotType()
{
    return myPlotView;
}

/*
/public void SetNumberOfThermocouples(int numIn)
/ Sets number of thermocouples in the GUI and in Arduino
/
/ Arguments: int numIn - to set as number of thermocouples
/
/ Returns: void
**/
public void SetNumberOfThermocouples(int numIn)
{
    NumberOfThermocouples = numIn;
    string stringout = "e" + Convert.ToString(numIn);
    byte[] outputStr = new byte[] { (byte)'e', (byte)numIn };
    WriteToBoard("MASTER", outputStr,2);
    lock (lockObject)
    {
        try
        {
            serialBS.GetPacket(ref _serialPort);
        }
        catch { }
    }
    InitiatePlot(myPlotView);
}

/*
/public void SetNumberOfControlZones(int numIn)
/ Sets number of control zones in the GUI and in Arduino
/
/ Arguments: int numIn - to set as number of zones
/
/ Returns: void
**/
public void SetNumberOfControlZones(int numIn)
{
    NumberOfHeatZones = numIn;
    byte[] outputStr = new byte[] { (byte)'f', (byte)numIn };
    WriteToBoard("MASTER", outputStr, 2);
    InitiatePlot(myPlotView);
```

```
    lock (lockObject)
    {
        try
        {
            serialBS.GetPacket(ref _serialPort);
        }catch { }
    }
}

/*
/public void UploadHex()
/ Sends hex file down to arduino board. Uses first connected board
/ that is found on the serial port
/
/ Arguments: none
/
/ Returns: void
**/
public void UploadHex(string hexFileName)
{
    string[] AvailablePorts = SerialPort.GetPortNames();
    string ComPort = null;

    if (AvailablePorts.Length > 0)
    {
        ComPort = AvailablePorts[0];
    }


    if ((ComPort == null) || (ComPort == ""))
    {
        ErrorBox = "ComPort not found";
        return;
    }

    string AvrDudeBin = "..\\..\\avrdude\\upload_code.bat";
    string AvrDudeParams = " " + ComPort + " " + hexFileName;

    Process p = new Process();
    p.StartInfo.FileName = AvrDudeBin;
    p.StartInfo.Arguments = AvrDudeParams;
    p.StartInfo.RedirectStandardError = false;
    p.StartInfo.RedirectStandardOutput = false;
    p.StartInfo.UseShellExecute = false;
    p.Start();
    p.WaitForExit(15000);
}


/*
/private void SetupNewPort(ref SerialPort _port)
/ Sets up a serial port with standard settings
/
```

```
        / Arguments: ref SerialPort _port - port to put setting into
        /
        / Returns: void
        **/
        private void SetupNewPort(ref SerialPort _port)
        {
            _port.BaudRate = 250000;
            _port.ReadTimeout = 500;
            _port.WriteTimeout = 500;
            _port.DtrEnable = true;
            _port.RtsEnable = true;
        } // private void SetupNewPort(ref SerialPort _port)
    } // public class ArduinoConnect
} // namespace MultiZone_OffBoardControl
```

B.10 Windows GUI Arduino/Computer serial protocol

```
// Filename: SerialProtocolBS.cs

using System;
using System.IO.Ports;

namespace MultiZone_OffBoardControl
{
    class SerialProtocolBS
    {
        private int maxPacketLength = 1000;

        /*
        / public byte[] GetPacket(ref SerialPort _serialPort)
        / Reads a packet from the serial port passed by reference
        /
        / Arguments: ref SerialPort _serialPort - serial port to read
        /
        / Returns: byte[] packet recieved from port, or null if error
        **/
        public byte[] GetPacket(ref SerialPort _serialPort)
        {
            int currentLocation = 0;
            byte[] packet = new byte[maxPacketLength];
            bool foundPacket = false;

            while (foundPacket == false)
            {
                //read byte from serial
                packet[currentLocation] = (byte)_serialPort.ReadByte();

                currentLocation++;
                foundPacket = FoundKey(packet, currentLocation);
                if (currentLocation >= maxPacketLength)
                {
                    return null;
                }
            }
```

```
    // Copy truncated array to output array
    byte[] outputArray = new byte[currentLocation];
    Array.Copy(packet, outputArray, outputArray.Length);

    return outputArray;

} // public byte[] GetPacket(ref SerialPort _serialPort)

/*
/ public float[] Unpack(byte[] packet)
/ Unpacked a recieved packet by using keys defined in Arduino embedded code
/
/ Arguments: byte[] packet - a packet to be unpacked
/
/ Returns: float[] an unpacked packet, null if error
**/
public float[] Unpack(byte[] packet)
{
    float[] unpacked = new float[maxPacketLength / 4];
    float[] outputArray = null;
    int unpackedLocation = 0;
    int packetLocation = 0;
    try
    {
        while (packetLocation < packet.Length - 6)
        {
            int caseSwitch = (int)packet[packetLocation];
            packetLocation++;
            switch (caseSwitch)
            {
                case 1:
                    byte[] temporary = new byte[2];
                    Array.Copy(packet, packetLocation, temporary, 0, 2);
                    unpacked[unpackedLocation] = FloatFromInt(temporary);
                    packetLocation += 2;
                    unpackedLocation++;
                    break;
                case 2:
                    byte[] temporary2 = new byte[4];
                    Array.Copy(packet, packetLocation, temporary2, 0, 4);
                    unpacked[unpackedLocation] = FloatFromLong(temporary2);
                    packetLocation += 4;
                    unpackedLocation++;
                    break;
                case 3:
                    byte[] temporary3 = new byte[4];
                    Array.Copy(packet, packetLocation, temporary3, 0, 4);
                    unpacked[unpackedLocation] = FloatFromFloat(temporary3);
                    packetLocation += 4;
                    unpackedLocation++;
                    break;
                case 4:
                    byte[] temporary4 = new byte[2];
```

```
                    Array.Copy(packet, packetLocation, temporary4, 0, 2);
                    unpacked[unpackedLocation] = FloatFromUInt(temporary4);
                    packetLocation += 2;
                    unpackedLocation++;
                    break;

                }
            }
            outputArray = new float[unpackedLocation];
            Array.Copy(unpacked, outputArray, outputArray.Length);
        }
        catch (Exception)
        {
            outputArray = null;
        }
        return outputArray;
    } // public float[] Unpack(char[] packet)

    /*
    / public bool PacketChecksum(byte[] packet)
    / Does a checksum on the packet
    /
    / Arguments: byte[] packet to do a checksum on
    /
    / Returns: bool true if passed, false if failed
    **/
    public bool PacketChecksum(byte[] packet)
    {
        int sumFromPacket = 0;

        for (int i = 0; i < packet.Length - 5; i++)
        {
            sumFromPacket += packet[i];
        }
        if ((sumFromPacket % 256) == packet[packet.Length - 5])
        {
            return true; // Checksum passed
        }

        return false;    // Checksum failed
    } // private bool PacketChecksum(char[] packet)

    /*
    / private bool FoundKey(byte[] packet, int currentLocation)
    / Looks for key to signal an end of a packet
    /
    / Arguments: byte[] packet
    /            int current location in packet
    /
    / Returns: bool true if found, false if not found
    **/
    private bool FoundKey(byte[] packet, int currentLocation)
    {
        if (currentLocation > 5)
```

```
    {
        for (int i = 1; i < 5; i++)
        {
            if (packet[currentLocation - i] != (5 - i))
            {
                return false; // Not at end of packet yet
            }
        }
        // Found a packet!!
        return true;
    }

    // Not large enough to be a packet...
    return false;
} // private bool FoundKey(byte[] packet, int currentLocation)

/*
/ private float FloatFromInt(byte[] input)
/ Converts input to a float. Assumes input to be
/ a length of 2 byte array
/
/ Arguments: byte[] input
/
/ Returns: float
**/
private float FloatFromInt(byte[] input)
{
    float output = -1;
    if (input.Length == 2)
    {
        if (BitConverter.IsLittleEndian)
        {
            Array.Reverse(input); // Data is sent in little endian
        }
        short tempInt = BitConverter.ToInt16(input, 0);
        output = (float)tempInt;
    }

    return output;
}

/*
/ private float FloatFromUInt(byte[] input)
/ Converts input to a float. Assumes input to be
/ a length of 2 byte array
/
/ Arguments: byte[] input
/
/ Returns: float
**/
private float FloatFromUInt(byte[] input)
{
    float output = -1;
    if (input.Length == 2)
```

```
    {
        if (BitConverter.IsLittleEndian)
        {
            Array.Reverse(input); // Data is sent in little endian
        }
        ushort tempInt = BitConverter.ToUInt16(input, 0);
        output = (float)tempInt;
    }

    return output;
}

/*
/ private float FloatFromLong(byte[] input)
/ Converts input to a float. Assumes input to be
/ a length of 4 byte array
/
/ Arguments: byte[] input
/
/ Returns: float
**/
private float FloatFromLong(byte[] input)
{
    float output = -1;
    if (input.Length == 4)
    {
        if (BitConverter.IsLittleEndian)
        {
            Array.Reverse(input); // Data is sent in little endian
        }
        Int32 tempInt = BitConverter.ToInt32(input, 0);
        output = (float)tempInt;
    }
    return output;
}

/*
/ private float FloatFromFloat(byte[] input)
/ Converts input to a float. Assumes input to be
/ a length of 4 byte array
/
/ Arguments: byte[] input
/
/ Returns: float
**/
private float FloatFromFloat(byte[] input)
{
    float output = -1;
    if (input.Length == 4)
    {
        if (BitConverter.IsLittleEndian)
        {
            Array.Reverse(input); // Data is sent in little endian. Need to reverse to match
                system
```

```
                }

                output = BitConverter.ToSingle(input, 0);
            }

            return output;
        }

    } // class SerialProtocolBS

} // namespace HeaterGui
```

Appendix C

MATLAB CODE

C.1 Supporting Chapter 2: main modeling file

```
%% Generate model from step response
% Data format:
%   Serial.print(time);
%   Serial.print(",");
%   TcHandler.PrintTCData();
%   Serial.print(",");
%   Serial.print(controller.GetRefTemp());
%   Serial.print(",");
%   if (controlOnEstimate)
%   {
%     Serial.print(tInnerEstimator.GetCurrentValue());
%     Serial.print(",");
%   }
%   Serial.println(CurrentOutput);

clear all; close all
global tau  w0  zeta  k  ki  kp ref timestep input a b
rawdata = csvread('step1_from12_10.txt');

% figure;
% plot([rawdata(:,3:end-1) rawdata(:,end)/10])

% For this dataset, first sample of step is at:

data1 = 110;

% Strip off pre-step data, go for 60 minutes
rawdata = rawdata(data1:end,:);
rawdata = rawdata(1:4*60*54.4,:);


time = rawdata(:,2)/1000;
sampPerSecond = 4;

tcdata = [rawdata(:,3) rawdata(:,4) rawdata(:,5)];
tcdata(:,1) = tcdata(:,1);% - ones(length(tcdata),1).*tcdata(1,1);
tcdata(:,2) = tcdata(:,2);% - ones(length(tcdata),1).*tcdata(1,2);
tcdata(:,3) = tcdata(:,3);% - ones(length(tcdata),1).*tcdata(1,3);

figure; plot(time/60,tcdata);
set(gca,'FontSize',14)
```

```
legend('T_i','T_o','T_1','Location','SouthEast')
xlabel('Time (min)'); ylabel('Temp. ($C^\circ$)')
title('Temperature Measurements from Step Input')
% xlim([0 60])

actualVoltage = 13.6;
willSetVoltage = sqrt(actualVoltage^2/.5)
input = (actualVoltage^2)/(willSetVoltage^2)*1000;

%T/I = k/(tau*s+1)
%tau*T' + T = kI
% T = [I -T'] * [k tau]'
offset = tcdata(1,1);
T = tcdata(:,1)-offset;

cutoffsamples = 30; % Cut off beginning and end samples to get
                    % rid of edge artifacts
[bz,az] = butter(2,(1/15)/2,'low'); % Filter at 1/15 Hz
Tdiff = filtfilt(bz,az, [0;diff(T)*sampPerSecond]);
Tdiff(1:cutoffsamples) = Tdiff(cutoffsamples+1);
Tdiff(end-cutoffsamples:end) = Tdiff(end-cutoffsamples-1); % Cut off tail
% Fit Tdiff to a 6th order polynomial to get smooth signal
p = polyfit(time(cutoffsamples:end-cutoffsamples),...
    Tdiff(cutoffsamples:end-cutoffsamples),6);
Tdifffit = polyval(p,time);
figure; plot([Tdiff Tdifffit]);
set(gca,'FontSize',14,'LineWidth',.5)
legend({'Tdiff','Tdiff polyfit'},'interpreter','latex','Fontsize',14,'Location','NorthEast')
xlabel('Sample'); ylabel('Tdiff (s)')

Tdiff = Tdifffit;

% Fit data to get transfer function
controlgain = input;
inp = ones(length(T),1) * controlgain;
Amat = [inp -Tdiff];
xhat = (Amat' * Amat) \( Amat' * T);
k = (xhat(1) * controlgain)/controlgain
tau = xhat(2)

% Plot transfer function
s = tf('s');
sys = k/(tau*s+1);
figure;
[tol,yol] = ode45(@olsysfunc, time',0);
plot((time-time(1))/60,tcdata(:,1),'r-',tol/60,yol+ offset,'b--','LineWidth',2)
set(gca,'FontSize',14,'LineWidth',.5)
legend({'Measured $T_{i}$','$1^{st}$ Order ...
Fit'},'interpreter','latex','Fontsize',14,'Location','SouthEast')
xlabel('Time (min)'); ylabel('Temp. ($^\circ C$)')
axis([0 60 10 170])
title('Fitting model to measurement')

%% Design PI controller
```

```
ff_gain = 1/k

w0 = 4/tau; % This value was chosen first
zeta = 1; % Choose a slightly overdamped system;
% From closed loop, we get the following in the demoninator:
%     s^2 + 2 zeta*w0 * s + w0^2
% tau*s^2 +  (1+k*kp) * s + k*ki,
% Solving, we get:

kp = (2*zeta*w0*tau-1)/k
ki = tau*w0^2/k

% ref = k*input;
ref = 125 - offset;

s = tf('s');
cont = kp + ki/s;

refinput = 20;
rampinput = 20*ones(1,4*60*5);
rampUpRate = 4; %deg C per minute
while refinput < 125
    rampinput(end+1) = refinput;
    refinput = refinput + (rampUpRate * .25) / 60;
end
rampinput = [rampinput 125*ones(1,4*60*45)];
noise = wgn(1,length(rampinput),5);
rampinput = rampinput+noise;

contd = c2d(cont,1/sampPerSecond)
[contd_num,contd_den] = tfdata(contd);

num_str = sprintf('float bArray [%s] = {',num2str(length(contd_num{1})));
den_str = sprintf('float aArray [%s] = {',num2str(length(contd_den{1})));
for i = 1:length(contd_num{1})
    num_str = strcat(num_str, ([num2str( contd_num{1}(i)) ',']));
    den_str = strcat(den_str, ([num2str( contd_den{1}(i)) ',']));
end
num_str = strcat(num_str(1:end-1),'};')
den_str = strcat(den_str(1:end-1),'};')

timestep = 1;
timesim = 0:timestep:30*60;

[toutsat,youtsat] = ode45(@clsysfuncsat, timesim',[0;0]);


figure
[AX,H1,H2] = plotyy(toutsat/60,[youtsat(:,2) + offset] ,toutsat/60, [youtsat(:,1)/10]);
set(gca,'FontSize',14)
set(AX,'FontSize',14)
xlabel('Time (min)','FontSize',14);
set(get(AX(1),'Ylabel'),'String','Temp. ($^\circ C$)','FontSize',14)
```

```
set(get(AX(2),'Ylabel'),'String','$\% T_{on}$','FontSize',14,'Color','r')
set(AX(1),'YLim',[0 150])
set(AX(1),'YTick',[0:25:150])
set(AX(2),'YLim',[0 100])
set(AX(2),'YTick',[0:10:100])
set(H1(1),'LineStyle','-','Color','b','LineWidth',2)
set(H2(1),'LineStyle','-','Color','r','LineWidth',2)
set(AX(2),'ycolor','r')
legend('$T_i$','$Control$')


%% Temperature Estimator design

firstpoint = 1;
tci = tcdata(firstpoint:end,1);
tco = tcdata(firstpoint:end,2);
tc1 = tcdata(firstpoint:end,3);

tcodiff = rawdata(:,7);
diffoffset = 26.5;

% Take a filtered differential of To's
[bz,az] = butter(2,(1/25)/2);

tcodiff_mat = [0; diff(tco)];
tcodiff_mat = filter(bz,az,tcodiff_mat);
tcodiff_mat = filter(bz,az,flipud(tcodiff_mat));
tcodiff_mat = flipud(tcodiff_mat)/.273*400;

% tcodiffhat = filter(bz,az,[0; diff(tco)/.265]);
tcodiffhat = tcodiff - diffoffset;
tcodiffhat(1:30) = 0;
tcodiffhat = filter(bz,az,tcodiffhat);
tcodiffhat = filter(bz,az,flipud(tcodiffhat));
tcodiffhat = flipud(tcodiffhat);

% tcodiffhat(end-100:end) = tcodiffhat(end-100);
% tcodiffhat = rawdata(:,7);
figure; plot(time/60, tcodiff - diffoffset)
set(gca,'FontSize',14)
xlabel('Time(min)','FontSize',14);
ylabel('Diff, raw','FontSize',14);
legend({'$T_0$ derivative'},'interpreter','latex','FontSize',14)
A0 = [tco tcodiffhat tc1];

figure; plot(time/60, [tcodiffhat tcodiff_mat])
set(gca,'FontSize',14)
xlabel('Time(min)','FontSize',14);
ylabel('Diff, raw','FontSize',14);
legend({'$\dot{T}_0$ Analog','$\dot{T}_0$ calculated post-process'},'interpreter','latex','FontSize',14)
A0 = [tco tcodiffhat tc1];
axis([0 60 0 45])

y = tci;
```

```
xhat0 = (A0' * A0) \( A0' * y)
tcodiffhat = tcodiff - diffoffset;
xhat0 = (A0' * A0) \( A0' * y)
tcip = A0 * xhat0;

percentChange = [-.2 -.15 -.12 -.1 -.09 -.08 -.07 -.06 -.05 -.04 -.03 -.02 -.01...
                  0 .01 .02 .03 .04 .05 .06 .07 .08 .09 .1 .12 .15 .2];
c1change = zeros(size(percentChange));
c2change = zeros(size(percentChange));
c3change = zeros(size(percentChange));

for i = 1:length(percentChange)
    % K1
    tcip = A0 * [1+(1+percentChange(i))*(xhat0(1)-1) ...
                 xhat0(2) ...
                 (1+percentChange(i))*xhat0(3)]';
    c1change(i) = sqrt(sum((tci - tcip).^2)/length(tci));

    % Ka
    tcip = A0 * [1+1/(1+percentChange(i))*(xhat0(1)-1) ...
                 1/(1+percentChange(i))*xhat0(2) ...
                 1/(1+percentChange(i))*xhat0(3)]';
    c2change(i) = sqrt(sum((tci - tcip).^2)/length(tci));

    % Cp, L, rho
    tcip = A0 * [xhat0(1) ...
                (1+percentChange(i))*xhat0(2) ...
                xhat0(3)]';
    c3change(i) = sqrt(sum((tci - tcip).^2)/length(tci));
end

figure;
plot(100*percentChange',[c1change' c2change' c3change'])
xlabel('$\%$ Change in parameter')
ylabel('RMS error')
legend('$p_1 = k_1$','$p_2 = k_a$','$p_3 = \rho L c_p$','Location','North')
return

tcip1(1) = tci(1);
for i = 2:length(tci)
    tcip1(i) = tco(i)*xhat0(1)+(tco(i)-tco(i-1))*xhat0(2)+tc1(i)*xhat0(3);
end

figure;
h1 = plot(time(firstpoint:end)/60,[tci tco tc1 tcip]);
set(gca,'FontSize',14)
set(h1(1),'Marker','o','LineWidth',1)
set(h1(2),'LineWidth',4)
set(h1(3),'LineWidth',4)
set(h1(4),'LineWidth',4,'Color','g')
legend({'$T_i$','$T_o$','$T_1...
$','$T_{i,e}$'},'interpreter','latex','FontSize',14,'Location','SouthEast')
xlabel('Time(min)','FontSize',14);
ylabel('Temp. ($^{\circ} C)$','FontSize',14);
```

```
xlim([0 60])

figure;
h1 = plot(time(firstpoint:end)/60,[tci tco tc1]);
set(gca,'FontSize',14)
set(h1(1),'Marker','o','LineWidth',1)
set(h1(2),'LineWidth',4)
set(h1(3),'LineWidth',4)
legend({'$T_i$','$T_o$','$T_1...
$','$T_{i,e}$'},'interpreter','latex','FontSize',14,'Location','SouthEast')
xlabel('Time(min)','FontSize',14);
ylabel('Temp. ($^{\circ} C)$','FontSize',14);
xlim([0 60])

RMSE = sqrt(sum((tci - tcip).^2)/length(tci))

figure;
plot(time(firstpoint:end)/60,[tci - tcip]);
set(gca,'FontSize',14)
xlabel('Time(min)','FontSize',14);
ylabel('Temp. Difference ($^{\circ} C)$','FontSize',14);
legend({'$T_i$ - $T_{i,e}$ '},'interpreter','latex','FontSize',14)
axis([1 60 -2 2])

%%%%%%%%%%%%%%
figure; subplot(121)
h1 = plot(time(firstpoint:end)/60,[tci tco tc1 tcip]);
set(gca,'FontSize',14)
set(h1(1),'LineStyle','o','LineWidth',2)
set(h1(2),'LineWidth',2)
set(h1(3),'LineWidth',2)
legend({'$T_i$','$T_o$','$T_1...
$','$T_{i,e}$'},'interpreter','latex','FontSize',14,'Location','SouthEast')
xlabel('Time(min)','FontSize',14);
ylabel('Temp. ($^{\circ} C)$','FontSize',14);
xlim([0 60])
%%%%%%%%%%%%%%
subplot(122)
plot(time(firstpoint:end)/60,[tci - tcip]);
set(gca,'FontSize',14)
xlabel('Time(min)','FontSize',14);
ylabel('Temp. Difference ($^{\circ} C)$','FontSize',14);
legend({'$T_i$ - $T_{i,e}$ '},'interpreter','latex','FontSize',14)
axis([1 60 -2 2])

%% Validate results in frequency domain

rawdata = csvread('sinesweep_from12_9.txt');
p1 = 1;
% pLast = 11025;
time = rawdata(p1:end,2)/1000;
tcdata = rawdata(p1:end,3:5);
reftemp = rawdata(p1:end,9);
figure; plot([tcdata reftemp])
```

```
figure; plot(time/60,[tcdata])
set(gca,'FontSize',14)
legend({'$T_i$','$T_o$','$T_1$'},...
    'interpreter','latex','FontSize',14,'Location','SouthEast')
xlabel('Time (min)','FontSize',14);
ylabel('Temperature ($^{\circ} C)$','FontSize',14);
% title('Sine Sweep Data')
xlim([20 320])
ylim([40 130])

transpoints = [6740 17780 32860 43903 54942 62669 70030 73800 76155 77336];
Tperiod =     [      1500, 1024,  750,  500,  350,  250,  128];
Tcycles =     [         2,    4,    4,    6,    6,    8,    8, 10, 10, 12];

xhatall = [];
t = [];
mag = [];
phase = [];

for i = 1:length(Tperiod)
    numpoints = length([transpoints(i):transpoints(i+1)]);
    t = 0:.25:numpoints*.25-.25;
    y = detrend(tcdata(transpoints(i):transpoints(i+1),:));
%     H = [[sin(2*pi/Tperiod(i)*t)]' [cos(2*pi/Tperiod(i)*t)]'];
    H = [[sin(2*pi/Tperiod(i)*t)]' [cos(2*pi/Tperiod(i)*t)]' t' ones(size(t'))];

    xhat = (H'*H)\(H'*y);
%     xhatall(:,i) = reshape(xhat,6,1);
    xhatall(:,i) = reshape(xhat,12,1);

    Y(i,:) = sqrt(xhat(1,:).^2 + xhat(2,:).^2);
    phase(i,:) = atan2(xhat(2,:),xhat(1,:));

    Tops = 0;
    Bottoms = 0;
    for j = 1:Tcycles(i)*2+1
        zerolocs(j,:) = (pi*(j-1) - phase(i,:))*Tperiod(i)/2/pi*4;
    end
    zerolocs = round(zerolocs);
    idx_negative = find( zerolocs <1 );
    zerolocs(idx_negative) = 1;
    idx_off_end = find( zerolocs >  transpoints(i+1)-transpoints(i) );
    zerolocs(idx_off_end) = transpoints(i+1)-transpoints(i)-1;
end

Freqs = 1./Tperiod;
w = 2*pi*Freqs;

A_i = xhatall(1,:); B_i = xhatall(2,:);
A_0 = xhatall(5,:); B_0 = xhatall(6,:);
A_1 = xhatall(9,:); B_1 = xhatall(10,:);

C1 = xhat0(1);
```

```
C2 = xhat0(2)*.273/400;
C3 = xhat0(3);

Mag_left = sqrt((A_0.*w).^2 + (B_0.*w).^2);
Mag_right = sqrt((A_i .*w).^2 + (B_i.*w).^2)./sqrt((C1*ones(size(w))).^2 + (C2*w).^2)...
- sqrt((C3*A_1 .*w).^2 + (C3*B_1.*w).^2)./sqrt((C1*ones(size(w))).^2 + (C2*w).^2);

Mag_right =  sqrt((A_i.*w - C3*A_1.*w).^2 + (B_i.*w - C3*B_1.*w).^2)...
            ./sqrt((C1*ones(size(w))).^2 + (C2*w).^2);

Phase_left = atan2(B_0.*w,A_0.*w);
Phase_right = atan2(B_i.*w - C3 *B_1.*w,A_i.*w-C3*A_1.*w) -...
    atan2(C2.*w,C1*ones(size(w)));

for i = 1:length(w)
    if Phase_left(i) < 0
        Phase_left(i) = Phase_left(i) + pi;
    end
    if Phase_right(i) < 0
        Phase_right(i) = Phase_right(i) + pi;
    end

end

for i = 1:length(w)
    if Phase_left(i) < pi/2
        Phase_left(i) = Phase_left(i) + pi;
    end
    if Phase_right(i) < pi/2
        Phase_right(i) = Phase_right(i) + pi;
    end

end
figure;
subplot(211)
semilogx(w,20*log10(Mag_left),'ko'); hold on
semilogx(w,20*log10(Mag_right),'ro');
xlim([w(1) w(end)])
set(gca,'FontSize',14)
legend({'Mag($T_0$)','Mag($1/(C_1 +  C_2 s) T_i$ - $C_3/(C_1 + C_2 s) T_1$)'},'FontSize',14,'Location','
    SouthEast')
xlabel('Freq (rad/s)'); ylabel('Mag (dB)')
% title('Testing $T_0 = 1/(C_1 +  C_2 s) T_i$ - $C_3/(C_1 + C_2 s) T_1$')
% Add a vertical line to indicate w0 of system
line('XData', [10*w0/2/pi 10*w0/2/pi], 'YData', [-60 -5], 'LineStyle', '-.', ...
    'LineWidth', 2, 'Color','g');
text(10.1*w0/2/pi, -16, '10 $\times$ C.L. natural freq.','FontSize',14)
ylim([-70 -10])

dBerrors = 20*log10(Mag_left) - 20*log10(Mag_right);

subplot(212)
semilogx(w,Phase_left*180/pi,'ko'); hold on
semilogx(w,Phase_right*180/pi,'ro');
```

```
xlim([w(1) w(end)])
set(gca,'FontSize',14)
legend({'Phase($T_0$)','Phase($1/(C_1 +  C_2 s) T_i$ - $C_3/(C_1 + C_2 s) T_1$)'},'FontSize',14,'Location'
    ,'SouthEast')
xlabel('Freq (rad/s)'); ylabel('Phase ($^\circ$)')
line('XData', [10*w0/2/pi 10*w0/2/pi], 'YData', [0 300], 'LineStyle', '-.', ...
    'LineWidth', 2, 'Color','g');
phaseerrors = (Phase_left*180/pi-Phase_right*180/pi)./(Phase_left*180/pi);

[w' (-1)*(Mag_left' - Mag_right')./Mag_left' (-1)*phaseerrors']
%% Estimator in FW
rawdata3 = csvread('bonding1_from_12_11.txt');
% rawdata3 = csvread('ramp1from_12_10.txt');

% 1 - error code
% 2 - time
% 3 - Tci
% 4 - Tc0
% 5 - Tc1
% 6 - Tca
% 7 - Tc0 Diff
% 8 - Ref Temp
% 9 - Ti_est
% 10 - ouput

time3 = rawdata3(:,2)/1000;
ti = rawdata3(:,3);
to = rawdata3(:,4);
todiff = rawdata3(:,7);
t1 = rawdata3(:,5);
tie = rawdata3(:,10);
tid = rawdata3(:,9);


figure; plot(time3/60,[ti to t1],'LineWidth',2);
set(gca,'FontSize',14)
legend({'$T_i$','$T_{o}$','$T_1$'},'FontSize',14,'Location','SouthEast')
xlabel('Time (min)'); ylabel('Temp. ($^\circ C$)')
axis([.5 108.5 15 140])

figure; plot(time3/60,[ti-tie]);
set(gca,'FontSize',14)
legend({'$T_i$ - $T_{i,e}$'},'interpreter','latex','FontSize',14)
xlabel('Time (min)'); ylabel('Temp. Difference ($^\circ C$)')
axis([1 108 -4 4])

yout = filtfilt(b,a,tid-tie);
figure; plot(time3/60,[tid-tie yout]);
set(gca,'FontSize',14)
legend({'$T_{d}$ - $T_{i,e}$','Filtered'},'FontSize',14)
xlabel('Time (min)'); ylabel('Temp. ($^\circ C$)')
axis([.5 108.5 -6 6])

figure; plot(time3/60,[tie]);
```

```
set(gca,'FontSize',14)
legend({'$T_{i,e}$'},'FontSize',14)
xlabel('Time (min)'); ylabel('Temp. ($^\circ C$)')
axis([60 100 122 128])

figure; plot(time3/60,[ti tie tid]);
set(gca,'FontSize',14)
legend({'$T_{i}$','$T_{i,e}$','$T_{i,d}$'},'FontSize',14)
xlabel('Time (min)'); ylabel('Temp. ($^\circ C$)')
axis([0 108 20 140])

figure; plot(time3/60,[ti-tid]);
set(gca,'FontSize',14)
legend({'$T_i$ - $T_{i,d}$'},'FontSize',14)
xlabel('Time (min)'); ylabel('Temp. ($^\circ C$)')
axis([1 108 -4 4])

figure;
h1 = plot(time3/60,[ti to t1 tie]);
set(gca,'FontSize',14)
set(h1(1),'Marker','o','LineWidth',1)
set(h1(2),'LineWidth',4)
set(h1(3),'LineWidth',4)
set(h1(4),'LineWidth',4,'Color','g')
legend({'$T_i$','$T_o$','$T_1$','$T_{i,e}$'},'interpreter','latex','FontSize',14,'Location','SouthEast')
xlabel('Time(min)','FontSize',14);
ylabel('Temp. ($^{\circ} C)$','FontSize',14);
xlim([0 108])
%
figure;
h1 = plot(time3/60,[ti to t1]);
set(gca,'FontSize',14)
set(h1(1),'Marker','o','LineWidth',1)
set(h1(2),'LineWidth',4)
set(h1(3),'LineWidth',4)
legend({'$T_i$','$T_o$','$T_1$','$T_{i,e}$'},'interpreter','latex','FontSize',14,'Location','SouthEast')
xlabel('Time(min)','FontSize',14);
ylabel('Temp. ($^{\circ} C)$','FontSize',14);
xlim([0 108])

%% Plot target profile

ramp = [];
ramp(1) = 25;
while ramp(end) < 125
    ramp(end+1) = ramp(end) + 0.5;
end
ramp = [ramp 125*ones(1,240)];
figure;
ramptime = [0:.25:length(ramp)*(.25) -.25];
plot(ramptime, ramp,'LineWidth',2)
set(gca,'FontSize',14)
xlabel('Time (min)','FontSize',14);
ylabel('Temperature ($^{\circ} C)$','FontSize',14);
```

```
legend({'Target Profile'},'interpreter','latex','FontSize',14,'Location','SouthEast')
line([17.5 32.5],[60 60],'Color','k','LineWidth',2)
line([32.5 32.5],[60 90],'Color','k','LineWidth',2)
text(40,70,'2 $^\circ C$/minute','FontSize',14)
axis([0 110 0 140])

line([35 50],[125 125],'Color','k','LineWidth',1.5,'LineStyle','-.')
text(25,125,'125','FontSize',14)

line([50 50],[123 110],'Color','k','LineWidth',2)
line([109 109],[123 110],'Color','k','LineWidth',2)
line([50 109],[116.5 116.5],'Color','k','LineWidth',2)
line([50 52.5],[116.5 119],'Color','k','LineWidth',2)
line([50 52.5],[116.5 114],'Color','k','LineWidth',2)
line([108.5 106],[116.5 119],'Color','k','LineWidth',2)
line([108.5 106],[116.5 114],'Color','k','LineWidth',2)
text(72,113,'60 minutes','FontSize',14)
```

C.2 Supporting Chapter 2: supporting function file

```
function dy = clsysfuncsat(t,y)
global tau  w0  zeta  k  ki  kp ref timestep

dy = zeros(2,1);

% Control ... u
dy(1) = -k*kp/tau * y(1) +(kp/tau - ki) * y(2) + ki*ref;

satValue = 1000;
if (y(1) + dy(1) * timestep) > satValue
    dy(1) = (satValue - y(1))/timestep;
end

if (y(1) + dy(1) * timestep) < 0
    dy(1) = (0 - y(1))/timestep;
end

% T_i ... y
dy(2) = k/tau * y(1) - 1/tau * y(2);
```

C.3 Supporting Chapter 3: FDM model

```
function [x,y,T,xpoints,graddata,BCidx_out]= ...
      LaplaceExplicitGapLocs(n,m,tablocs,tabson,tabthickness,tabgap,silent,BDonly)
% n - width of heater, x, 7 in
% m - length of heater, y, 6 in
% tab gap is gap between tabs (defined below)
% tablocs defines both the node locations for the center of the tab gaps,
% and the last element in the array indicates the width of the heater.
% If tablocs is different than n,m, adjust by last element of tablocs
if nargin < 8
    BDonly = 0;
end
```

```
% return uniform solution if 1 tab

R = 20.0; % Voltage
xnLen = 0.1778;
ymLen = m/20*0.0254;
initialGuess = 0:(R/(m)):R;
scalefactor = (R/(m-1)); % Scale for 6 inch long heater and m nodes
epsilon = 1e-6; % tolerance for convergence

x = [0:n];y=[0:m];
xpoints = 0:7/(n):7; % Convert scale to 9 inch wide heater

T = repmat(initialGuess,n+1,1);


if length(tablocs) == 1
        BCidx = ones(n+1,1);
        offTabIdx = [];
        if tabson(1) == 0
            BCidx_out = BCidx * 0;
            T = T*0;
            graddata = T';
            return;
        end
else
    tablocs_scaled = round(tablocs(1:end-1)* n/tablocs(end));
    if tablocs_scaled(1) < tabgap
        tablocs_scaled(1) = [];
        tablocs(1) = [];
        tabson(1) = [];
    end

    idx = 2;
    while idx <= length(tablocs_scaled)
        if (tablocs_scaled(idx)-tablocs_scaled(idx-1)) < (tabgap + 1)
            if(idx == length(tablocs_scaled))
                tablocs_scaled(end) = [];
                tablocs_scaled(end) = m+1;
                tablocs(idx) = [];
                tabson(idx) = [];
                break;
            else
                tablocs_scaled(idx-1) = ...
                  round((tablocs_scaled(idx) + tablocs_scaled(idx-1))/2);
                tablocs_scaled(idx) = [];
                tablocs(idx) = [];
                tabson(idx) = [];
            end
        else
            idx = idx+1;
        end
    end

    numtabs = length(tablocs);
```

```
    if (nargin < 5)
        silent = 0;
    end


    BCidx = ones(n+1,1);
    tgap = (tabgap-1)/2;

    if (tabgap*(numtabs-1)+numtabs) > n
        error('Tabs with spacing will not fit the geometry');
    end

    for idx = 1:length(tablocs_scaled)
        BCidx(tablocs_scaled(idx)-tgap:tablocs_scaled(idx)+tgap) = 0;
    end

    BCidx_out = BCidx;
    if BDonly == 1
        graddata = T';
        return;
    end
    % Deactivate all unpowered tabs by setting to -1 in BCidx
    loc = 1;
    offTabIdx = [];
    for idx = 1:numtabs
        startIdx = find(BCidx);
        startIdx = startIdx(find(startIdx >= loc));
        loc = startIdx(1);
        while (BCidx(loc) == 1) && (loc < length(BCidx));
            loc = loc+1;
        end
        if (loc == length(BCidx))
            loc = loc+1;
        end
        if(tabson(idx) == 0)
            offTabIdx(end+1) = startIdx(1);
            offTabIdx(end+1) = loc-1;
            BCidx(startIdx(1):loc-1) = -1;
        end
    end
end
% Plot tabbed boundary, 1 for powered, -2 for unpowered, 0 for gap
% figure; plot(BCidx,'.-');
for idx = 1:tabthickness
    T(:,m+2-idx) = R*BCidx;
end
T(:,1:tabthickness) = 0;



TN = T; % TN = new iteration for solution
err = TN-T;
% Parameters in the solution
```

```
denom = 4;
beta = 1;
% Iterative procedure
imax = 1000000; % maximum number of iterations allowed
k = 1; % initial index value for iteration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculation loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while k<= imax
    for i = 2:n % solve interior points
        for j = 1+tabthickness:m+1-tabthickness
            TN(i,j)=(TN(i-1,j)+T(i+1,j)+beta^2*(TN(i,j-1)+T(i,j+1)))/denom;
            err(i,j) = abs(TN(i,j)-T(i,j));
        end;
    end;
    for i = 2:n % solve points along tabbed boundary
        if BCidx(i) == 0 % Solve tab gaps on edge
            TN(i,1)=(T(i-1,1)+T(i+1,1)+beta^2*(2*T(i,2)))/denom;
            err(i,1) = abs(TN(i,1)-T(i,1));
            TN(i,m+1)=(T(i-1,m+1)+T(i+1,m+1)+beta^2*(2*T(i,m)))/denom;
            err(i,m+1) = abs(TN(i,m+1)-T(i,m+1));
        end
        for j = 2:tabthickness % Solve space between tabs
            if BCidx(i) == 0
                TN(i,j)=(TN(i-1,j)+T(i+1,j)+beta^2*(TN(i,j-1)+T(i,j+1)))/denom;
                err(i,j) = abs(TN(i,j)-T(i,j));
                TN(i,m+2-j)=(TN(i-1,m+2-j)+T(i+1,m+2-j)+beta^2*(TN(i,m+2-j-1)+T(i,m+1-j)))/denom;
                err(i,m+2-j) = abs(TN(i,m+2-j)-T(i,m+2-j));
            end
        end
    end
    for i = 1:length(offTabIdx)/2 % Solve along tabs where no voltage applied
        if (offTabIdx(i*2-1) == 1) % first tab
            TN(1:offTabIdx(2),1:tabthickness) = ...
                (sum(T(offTabIdx(2)+1,1:tabthickness))+...
                sum(T(1:offTabIdx(2),1+tabthickness)))/(offTabIdx(2)+tabthickness);
            TN(1:offTabIdx(2),m+2-tabthickness:m+1) = ...
                (sum(T(offTabIdx(2)+1,m+2-tabthickness:m+1))...
                +sum(T(1:offTabIdx(2),m+1-tabthickness)))/(offTabIdx(2)+tabthickness);
        elseif (offTabIdx(i*2) == n+1) % last tab
            TN(offTabIdx(end-1):offTabIdx(end),1:tabthickness) = ...
                (sum(T(offTabIdx(end-1)-1,1:tabthickness))+...
                sum((T(offTabIdx(end-1):offTabIdx(end),1+tabthickness))))/...
                (tabthickness+length(offTabIdx(end-1):offTabIdx(end)));
            TN(offTabIdx(end-1):offTabIdx(end),m+2-tabthickness:m+1) =...
                (sum(T(offTabIdx(end-1)-1,m+2-tabthickness:m+1))+...
                sum(T(offTabIdx(end-1):offTabIdx(end),m+1-tabthickness)))/...
                (tabthickness+length(offTabIdx(end-1):offTabIdx(end)));
        else % a central tabs
            TN(offTabIdx(i*2-1):offTabIdx(i*2),1:tabthickness) = ...
                (sum(T(offTabIdx(i*2-1)-1,1:tabthickness))+...
                sum(T(offTabIdx(i*2)+1,1:tabthickness))+...
                sum(T(offTabIdx(i*2-1):offTabIdx(i*2),1+tabthickness)))/...
                (2*tabthickness+length(offTabIdx(i*2-1):offTabIdx(i*2)));
```

```
            TN(offTabIdx(i*2-1):offTabIdx(i*2),m+2-tabthickness:m+1) =...
                (sum(T(offTabIdx(i*2-1)-1,m+2-tabthickness:m+1))+...
                sum(T(offTabIdx(i*2)+1,m+2-tabthickness:m+1))+...
                sum(T(offTabIdx(i*2-1):offTabIdx(i*2),m+1-tabthickness)))/...
                (2*tabthickness+length(offTabIdx(i*2-1):offTabIdx(i*2)));
        end
    end
    for j = 1+tabthickness:m+1-tabthickness % solve points along non-tabbed boundary
        TN(1,j)=(2*T(2,j)+beta^2*(T(1,j-1)+T(1,j+1)))/denom;
        err(1,j) = abs(TN(1,j)-T(1,j));
        TN(n+1,j)=(2*T(n,j)+beta^2*(T(n+1,j-1)+T(n+1,j+1)))/denom;
        err(n+1,j) = abs(TN(n+1,j)-T(n+1,j));
    end

    T = TN; k = k + 1;
    errmax = max(max(err));
    if errmax < epsilon
        [X,Y] = meshgrid(x,y);
%         figure(2);contour(X,Y,T',20);xlabel('x');ylabel('y');
%         title('Laplace equation solution - Dirichlet boundary conditions - Explicit');
%         figure;surfc(X,Y,T');xlabel('x');ylabel('y');zlabel('T(x,y)');
%         title('Laplace equation solution - Dirichlet boundary conditions - Explicit');
%         figure;
        [px,py] = gradient(T');
%         surfc(X,Y,sqrt(px.^2+py.^2));xlabel('x');ylabel('y');zlabel('Grad T(x,y)');
        graddata = (sqrt(px.^2+py.^2)).^2*m^2/ymLen  ; % Convert to power (field^2)
%         ydelt = 8.8582677/(n+1);
%         ydelt = 9/(n+1);
%         xpoints = 0:8.8582677/n:8.8582677; % Convert scale to 9 inch wide heater
%         figure; plot(xpoints,graddata(round(m/2),:))
        if silent == 0
            fprintf('Convergence achieved after %i iterations.\n',k);
        end
        return
    end;
end;
fprintf('\n No convergence after %i iterations.',k);
```

C.4 Supporting Chapter 3: estimating desired power

```
function out = estimatingDesiredPower(fitType,tcskip)
out = struct;
if nargin < 1
    tcskip = 1;
    fitType = 1;
elseif nargin < 2
    fitType = 1;
end

out.fitType = fitType;
plottingOn = [0 0 0 1 0 0 1 1 1 0  0];
%             1 2 3 4 5 6 7 8 9 10 11
% Plotting options
% 1 - raw data
```

```
% 2 - Average of each tc at end
% 3 - Plot control
% 4 - Filtered tcs
% 5 - tcsdiff
% 6 - simulated tcs
% 7 - tcs end points (temperature profile at end of dataset)
% 8 - tcs sim vs measured end points
% 9 - Error in sim vs measured jetted through time
% 10 - beta plot
% 11 - weighting matrix for original fit
% Equation fitting:
% gamma Tdot_i = -beta (T_i) - alpha_{i-}(T_i-T_{i-1}) -...
%                alpha_{i+}(T_i-T_{i+1}) + Q_i

% Tmat = tcsfilt;
%   fitType
%     1 - alpha and gamma constant
%     2 - alpha variable, gamma constant
%     3 - alpha constant, gamma variable
%     4 - alpha = 0, gamma is constant
%     5 - alpha and gamma constant
%         Betas are a linear function of temperature

%%%%%%%%%%%%%%%%%%% switching between files %%%%%%%%%%%%%%%%%%%%%%%%
% data = csvread('singleZoneW_HS_4_18_16.txt');
% % Input should be in V^2/m^2
% input =((17.5/0.1778)^2);

% dataset 1
% fname1 = 'C:\Users\Brandon\Dropbox\Brandon\ExpData\Multizone_fullpaper\thickdrystep\inner_innercontrol2.
    csv';
% fname1 = 'C:\Users\Brandon\Dropbox\Brandon\ExpData\Multizone_fullpaper\thickdryramp\inner_continner_6_14
    .csv';
fname1 = 'C:\Users\Brandon\Dropbox\Brandon\ExpData\Multizone_fullpaper\thickdrystep\
    step2_inner_innercont_6_23.csv';


[data,timedata,~]=xlsread(fname1);
% input =(0.5*(20^2/0.1524));
input =((19.9^2/0.1524));

controlIndex = 21;

% input =((14.8/0.1524)^2);

out.inputV = input;
%dataset 2
% fname2 = 'C:\Users\Brandon\Dropbox\Brandon\ExpData\Multizone_fullpaper\thickdryramp\inner_continner_6_14
    .csv';
% fname2 = 'C:\Users\Brandon\Dropbox\Brandon\ExpData\Multizone_fullpaper\thickdryramp\
    ramp1_inner_innercont_6_22.csv';
fname2 = 'C:\Users\Brandon\Dropbox\Brandon\ExpData\Multizone_fullpaper\thickdryramp\
    ramp1_inner_innercont_6_22.csv';
```

```
[data2,timedata2,~]=xlsread(fname2);
input2 =((20.1^2/0.1524));
controlIndex2 = 21;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
firstpoint = find(data(:,2) == min(data(:,2)));
firstpoint2 = find(data2(:,2) == min(data2(:,2)));

lastpoint = 0;
time = datetime(timedata,'Format','HH:mm:ss.SSS');
time = time.Hour*60*60+time.Minute*60+time.Second;
time = time(firstpoint:end-lastpoint);
time = time-time(1);

tsample = mean(diff(time));
Tsim = 0:tsample:time(end)+10;
Tsim = Tsim(1:length(time));
out.Tsim = Tsim;


time2 = datetime(timedata2,'Format','HH:mm:ss.SSS');
time2 = time2.Hour*60*60+time2.Minute*60+time2.Second;
time2 = time2(firstpoint2:end-lastpoint);
time2 = time2-time2(1);
tsample2 = mean(diff(time2));
Tsim2 = 0:tsample2:time2(end)+10;
Tsim2 = Tsim2(1:length(time2));
out.Tsim2 = Tsim2;

if tcskip < 6
    dataArtificial = 0;
else
    dataArtificial = 1;
end

tcnumbers = {};
tclocs = {}; tcnumbers{1} = 3:17;
tclocs{1} = 0:.5:7; % Thermocouple locations assuming starts at 0 and goes to 7
tcnumbers{2} = 3:2:17; tclocs{2} = 0:1:7;
tcnumbers{3} = 4:3:17; tclocs{3} = .5:1.5:7;
tcnumbers{4} = 4:4:17; tclocs{4} = .5:2:7;
tcnumbers{5} = 5:5:17; tclocs{5} = 1:2.5:7;

% Artificial data
if dataArtificial == 1
    npoints = tcskip; dx = 7/(npoints-1);
    tclocs{tcskip} = 0:dx:7;
end

out.tclocs = tclocs{tcskip};
reftemp = data(firstpoint:end-lastpoint,19);
rampreftemp = data2(firstpoint:end-lastpoint,19);
out.rampreftemp = rampreftemp;
```

```
if dataArtificial == 0
    tcs = data(firstpoint:end-lastpoint,tcnumbers{tcskip});
    tcs_offset = mean(mean(tcs(1:20,:)))
    out.tcs_offset = tcs_offset;
    tcs = tcs-mean(mean(tcs(1:20,:)));
    tcs2 = data2(firstpoint2:end-lastpoint,tcnumbers{tcskip});
    tcs2 = tcs2-mean(mean(tcs2(1:20,:)));
else
    X1 = 0:.5:7;
    X2 = tclocs{tcskip};
    artificialdata = [];
    tempData = data(firstpoint:end-lastpoint,3:17);
    artificialdata2 = [];
    tempData2 = data2(firstpoint2:end-lastpoint,3:17);
    for idx = 1:size(data(firstpoint:end-lastpoint,1),1)
        artificialdata(idx,:) = interp1(X1,tempData(idx,:),X2,'pchip','extrap');
    end
    for idx = 1:size(data2(firstpoint2:end-lastpoint,1),1)
        artificialdata2(idx,:) = interp1(X1,tempData2(idx,:),X2,'pchip','extrap');
    end
    tcs = artificialdata;
    tcs_offset = mean(mean(tcs(1:20,:)))
    tcs = tcs-mean(mean(tcs(1:20,:)));

    tcs2 = artificialdata2;
    tcs2 = tcs2-mean(mean(tcs2(1:20,:)));
end
out.tcs = tcs;

%plot raw data
if (plottingOn(1) == 1)
    figure; plot(time/60, tcs);
    set(gca,'FontSize',14)
    legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15')
    xlabel('Time (min)'); ylabel('Temp. ($C^\circ$)')
end

tcavgs = mean(tcs(end-10:end,:));
orig_xpoints = 1:7/(size(tcs,2)-1):8;

% Plot average of each tc at end
if (plottingOn(2) == 1)
    figure; plot(orig_xpoints,tcavgs)
    set(gca,'FontSize',18)
    xlabel('$x$ (in)'); ylabel('Temperature ($C^\circ$)')
    xlim([1 8])
end

controlvals = data(firstpoint:end-lastpoint,controlIndex);
controlvals2 = data2(firstpoint2:end-lastpoint,controlIndex2);
[bz,az] = butter(2,(1/10)/2,'low'); % Filter at 1/15 Hz
controlFiltered = filtfilt(bz,az,controlvals);
controlFiltered2 = filtfilt(bz,az,controlvals2);
out.control = controlFiltered/1000;
```

```
if (plottingOn(3) == 1)
    figure;
    plot(time/60,input*controlFiltered/1000)
    xlabel('Time (min)'); ylabel('Control input $(V^2/m)$')
end

[bz,az] = butter(2,(1/25)/2,'low'); % Filter at 1/15 Hz
tcsfilt = filtfilt(bz,az,tcs);
out.tcsfilt = tcsfilt;
tcsfilt2 = filtfilt(bz,az,tcs2);

if (plottingOn(4) == 1)
    figure; plot(time/60, tcsfilt);
    set(gca,'FontSize',14)
    xlabel('Time (min)'); ylabel('Temp. ($C^\circ$)')
end

% get filtered diff
[bz,az] = butter(3,(1/50)/2,'low'); % Filter at 1/15 Hz
tcsdiff = diff(filtfilt(bz,az,tcs))/.308;
tcsdiff = [tcsdiff(1,:); tcsdiff];
if (plottingOn(5) == 1)
    figure; plot(tcsdiff);title('tcsdiff from real exp')
end
numtcs = size(tcs,2);
out.numtcs = numtcs;


%%%%%%%%%%%%%%%%%%%%%%%%%
ScaleDiff = 1;
maxtemp = max(tcsfilt2(end,:));
maxp = size(tcsfilt,1);

%%%%%%%%%%%%%%%%%%%%%%%%%

[Tmat, Umat] = getTandU(fitType,tcsfilt(1:maxp,:),...
                        ScaleDiff*tcsdiff(1:maxp,:),...
                        input*controlFiltered(1:maxp)/1000);

xsoln = (Tmat'*Tmat)\(Tmat'*Umat);

%%
%% Weighting matrix
%%
if (sum(diff(reftemp)) < 10)

    slopeM =1/length(tcs(1:maxp,1));
    Wmat = [1:length(tcs(1:maxp,1))]'*slopeM;

else
    Wmat = (reftemp(1:maxp)).^2;
    slopeM =1/length(tcs(1:maxp,1));
    Wmat = [1:length(tcs(1:maxp,1))]'*slopeM;
```

```
    Wmat = ones(size(Wmat));
end

if (plottingOn(11) == 1)
    figure; plot(Wmat)
    set(gca,'FontSize',14)
    xlabel('n'); ylabel('Weighting')
end
Wmat = repmat(Wmat,numtcs,1);
%
f = @(x) norm(Wmat.*(Umat - Tmat*x),2);

xsoln = fmincon(f,xsoln,-eye(size(Tmat,2)),...
    0*ones(size(Tmat,2),1),[],[],[],[],[],...
    optimset('Algorithm','interior-point','MaxFunEvals',10000));
xsoln(end) =  ScaleDiff * xsoln(end);
out.xsoln = xsoln;

%% Going the other direction
if fitType == 5
    zeta  = xsoln(1:2:end-3);
    eta   = xsoln(2:2:end-2);
    alpha = xsoln(end-1);
    gamma = xsoln(end);
    thisrange = [1:100];
    fn = @(t,zeta,eta) zeta*t + eta;
    for idx = 1:numtcs
        figure; plot(thisrange,fn(thisrange,zeta(idx),eta(idx)))
    end

    U = [Tsim', input*controlFiltered/1000];
    X0 = mean(tcsfilt(1:10,:));
    [Tout,Yout]=ode45('nonLinFitODE45',Tsim,X0,[],U,xsoln);
    figure; plot(Tout,Yout);

else
    [Cp, beta, alpha, Amat, Bmat, Cmat, Dmat] = getABCD(fitType,xsoln,numtcs);
    out.alpha = alpha;
    out.beta = beta;
    out.gamma = Cp;

    Sysfake = ss(Amat,Bmat,Cmat,Dmat);


    % X0 = zeros(numtcs,1);
    X0 = mean(tcsfilt(1:10,:));
    [Yout,Tout,~] = lsim(Sysfake,input*controlFiltered/1000,Tsim,X0);
    out.origYout = Yout;

end
if (plottingOn(6) == 1)
    figure; plot(Tout/60,Yout)
    set(gca,'FontSize',14)
    xlabel('Time (min)'); ylabel('Temp. ($C^\circ$)')
```

```
end
if (plottingOn(7) == 1)
    figure; plot(orig_xpoints,mean(tcsfilt(end-20:end,:))+tcs_offset);
    set(gca,'FontSize',14)
    xlabel('x (in)'); ylabel('Temperature ($C^\circ$)')
    xlim([1 8])
end

if (plottingOn(8) == 1)
    figure;
    plot(orig_xpoints,Yout(end,:),orig_xpoints,mean(tcsfilt(end-10:end,:))); hold on
    plot(orig_xpoints,Yout(end,:),'bx',orig_xpoints,mean(tcsfilt(end-10:end,:)),'ro')
    set(gca,'FontSize',14)
    legend('Simulation','Experiment','Location','SouthEast')
    xlabel('Thermocouple $\#$'); ylabel('Temp. ($C^\circ$)')
    xlim([1 8])
    figure;
    plot(orig_xpoints,Yout(end,:)-mean(tcsfilt(end-10:end,:)))
    set(gca,'FontSize',14)
    legend('Simulation - Experiment','Location','SouthEast')
    xlabel('Thermocouple $\#$'); ylabel('Temp. ($C^\circ$)')
    xlim([1 8])
end

if (plottingOn(9) == 1)
    plotpoints = 100:200:length(Yout)-10;
    colors = jet(length(plotpoints));
    idx1 = 1;
    figure
    for idx = plotpoints
        plot(orig_xpoints,Yout(idx,:)-mean(tcsfilt(idx-10:idx+10,:)),'Color',colors(idx1,:));
        hold on
        idx1 = idx1 + 1;
    end
    xlim([orig_xpoints(1) orig_xpoints(end)])
    set(gca,'FontSize',14)
    xlabel('Thermocouple $\#$'); ylabel('Temp. ($C^\circ$)')
end

if (plottingOn(10) == 1)
    figure; plot(beta,'o-');
    set(gca,'FontSize',14)
    xlabel('Index');ylabel('$\beta$ value')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Test fit with second data set
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
plottingOn2 = [0 0 1 1 1 1];
%              1 2 3 4 5 6
% Plotting options
% 1 - Simulation tcs for input2 with system 1 fit
% 2 - tcs2 end points
% 3 - Plots of sim and measured at end, and error between
```

```
% 4 - Jetted error between fit and tcs2
% 5 - tcs2 filtered
% 6 - Control

X0 = mean(tcsfilt2(1:10,:));
lastpoint = size(tcsfilt2,1);
[Yout,Tout,~] = lsim(Sysfake,input2*controlFiltered2(1:lastpoint)/1000,Tsim2(1:lastpoint),X0);

if (plottingOn2(1) == 1)
    figure; plot(Tout/60,Yout)
    set(gca,'FontSize',14)
    xlabel('Time (min)'); ylabel('Temp. ($C^\circ$)')
end
if (plottingOn2(2) == 1)
    figure; plot(orig_xpoints,mean(tcsfilt2(lastpoint-20:lastpoint,:)))
    set(gca,'FontSize',14)
    xlabel('Thermocouple $\#$'); ylabel('Temp. ($C^\circ$)')
end

if (plottingOn2(6) == 1)
    figure;
%     plot(input*controlvals/1000); hold on

    plot(time2/60,input2*controlFiltered2/1000)
    xlabel('Time (min)'); ylabel('Control input $(V^2/m)$')
end

if (plottingOn2(3) == 1)
    figure;
    plot(orig_xpoints,Yout(end,:),orig_xpoints,mean(tcsfilt2(lastpoint-10:lastpoint,:))); hold on
    plot(orig_xpoints,Yout(end,:),'bx',orig_xpoints,mean(tcsfilt2(lastpoint-10:lastpoint,:)),'ro')
    set(gca,'FontSize',14)
    legend('Simulation (predicted)','Experiment','Location','SouthEast')
    xlabel('Thermocouple $\#$'); ylabel('Temp. ($C^\circ$)')
    xlim([1 8])

    figure;
    plot(orig_xpoints,Yout(end,:)-mean(tcsfilt2(end-10:end,:)))
    set(gca,'FontSize',14)
    legend('Simulation - Experiment','Location','SouthEast')
    xlabel('Thermocouple $\#$'); ylabel('Temp. ($C^\circ$)')
    xlim([1 8])

end
% rmsError = rms(Yout(end,:)-mean(tcs2(end-20:end,:)));
% out.rmsError = rmsError;

if (plottingOn2(4) == 1)
    plotpoints = 100:200:length(Yout)-10;
    colors = jet(length(plotpoints));
    idx1 = 1;
    figure
    for idx = plotpoints
        errorCurrentIdx = Yout(idx,:)-mean(tcsfilt2(idx-10:idx+10,:));
```

```
        plot(orig_xpoints,errorCurrentIdx-mean(errorCurrentIdx),'Color',colors(idx1,:));
        hold on
        idx1 = idx1 + 1;
    end
    xlim([orig_xpoints(1) orig_xpoints(end)])
    set(gca,'FontSize',14)
    xlabel('Thermocouple $\#$'); ylabel('Temp. ($C^\circ$)')
end

if (plottingOn2(5) == 1)
    figure; plot(time2/60, tcsfilt2);
    set(gca,'FontSize',14)
%     legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15')
    xlabel('Time (min)'); ylabel('Temp. ($C^\circ$)')
end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 function [Tmat, Umat] = getTandU(fitType,tcs,tcsdiff,u_input)
    %   fitType
    %     1 - alpha and gamma constant
    %     2 - alpha variable, gamma constant
    %     3 - alpha constant, gamma variable
    %     4 - alpha = 0, gamma is constant
    %     5 - alpha and gamma constant
    %         Betas are a linear function of temperature

    Tmat = [];
    Umat = [];
    numtcs = size(tcs,2);
    numk = size(tcs,1);

    if fitType == 1
        for idx = 1:size(tcs,2)
            if idx == 1
                Tmat = [tcs(:,1), zeros(numk,numtcs-1),...
                        (tcs(:,1)-tcs(:,2)),tcsdiff(:,idx)];
            elseif idx == size(tcs,2)
                Tmat = [Tmat;  zeros(numk,numtcs-1),...
                                tcs(:,end), (tcs(:,end)-tcs(:,end-1)),...
                                tcsdiff(:,idx)];
            else
                Tmat = [Tmat; zeros(numk,idx-1),...
                    tcs(:,idx),zeros(numk,numtcs-idx),...
                    2*tcs(:,idx)-tcs(:,idx+1)-tcs(:,idx-1),...
                    tcsdiff(:,idx)];
            end
             Umat = [Umat;u_input];
        end
    end

    if fitType == 2
        for idx = 1:size(tcs,2)
```

```
        if idx == 1
           Tmat = [tcs(:,1), (tcs(:,1)-tcs(:,2)), zeros(numk,2*numtcs-3),...
                     tcsdiff(:,idx)];
        elseif idx == size(tcs,2)
           Tmat = [Tmat;  zeros(numk,2*numtcs-3),...
                                (tcs(:,end)-tcs(:,end-1)), tcs(:,end),...
                                tcsdiff(:,idx)];

        else
           Tmat = [Tmat; zeros(numk,(idx-1)*2-1),...
              tcs(:,idx)-tcs(:,idx-1), tcs(:,idx), tcs(:,idx)-tcs(:,idx+1),...
              zeros(numk,(numtcs-idx)*2-1), tcsdiff(:,idx)];
        end
        %    Umat = [Umat;input-tcsdiff(:,idx)];
        Umat = [Umat;u_input];
    end
end

if fitType == 3
    for idx = 1:size(tcs,2)
        if idx == 1
            Tmat = [tcs(:,1), tcsdiff(:,idx), zeros(numk,2*numtcs-2),...
                      tcs(:,1)-tcs(:,2)];
        elseif idx == size(tcs,2)
            Tmat = [Tmat;  zeros(numk,2*(idx-1)), tcs(:,idx), tcsdiff(:,idx)...
                         (tcs(:,end)-tcs(:,end-1))];
        else
            Tmat = [Tmat; zeros(numk,2*(idx-1)),...
              tcs(:,idx), tcsdiff(:,idx), zeros(numk,2*(numtcs-idx)),...
              2*tcs(:,idx)-tcs(:,idx+1)-tcs(:,idx-1)];
        end
        Umat = [Umat;u_input];
    end

end

if fitType == 4
    for idx = 1:size(tcs,2)
        if idx == 1
            Tmat = [tcs(:,1), zeros(numk,numtcs-1),...
                    tcsdiff(:,idx)];
        elseif idx == size(tcs,2)
            Tmat = [Tmat;  zeros(numk,numtcs-1),...
                                tcs(:,end),...
                                tcsdiff(:,idx)];
        else
            Tmat = [Tmat; zeros(numk,idx-1),...
              tcs(:,idx),zeros(numk,numtcs-idx),...
              tcsdiff(:,idx)];
        end
        Umat = [Umat;u_input];
    end
end
```

```
    if fitType == 5 % Betas are a linear function of temperature
        for idx = 1:size(tcs,2)
            if idx == 1
                Tmat = [tcs(:,1).*tcs(:,1), tcs(:,1) ,...
                        zeros(numk,2*(numtcs-1)) ,...
                        (tcs(:,1)-tcs(:,2)),tcsdiff(:,idx)];
            elseif idx == size(tcs,2)
                Tmat = [Tmat;  zeros(numk,2*(numtcs-1)) ,...
                        tcs(:,end).*tcs(:,end), tcs(:,end) ,...
                        (tcs(:,end)-tcs(:,end-1)), tcsdiff(:,idx)];
            else
                Tmat = [Tmat; zeros(numk,2*(idx-1)) ,...
                    tcs(:,idx).*tcs(:,idx), tcs(:,idx) ,...
                    zeros(numk,2*(numtcs-idx)) ,...
                    2*tcs(:,idx)-tcs(:,idx+1)-tcs(:,idx-1), tcsdiff(:,idx)];
            end
             Umat = [Umat;u_input];
        end
    end
end
```

C.5 Supporting Chapter 3: estimating desired power refined

```
function out = estimatingDesiredPowerRefined(modelData,modelpoints)
%ESTIMATINGDESIREDPOWERREFINED Takes model data and produces a model with a
% finer mesh.
%
% out = estimatingDesiredPowerRefined(modelData,modelpoints)
%
% modelData - structure containing information for model
%   - inputV -  V^2/m^2 level
%   - control - time domain control applied to heater (scaled 0 to 1)
%   - fitType
%   - xsoln
%   - numtcs
%   - Tsim
%
% modelpoints - number of points to refine to. Points will be aligned so
%               that the first and last point are at first and last
%               thermocouple and the rest are placed equally between.
%


    plottingOn = [0 0 0 0 0 0 0 1 0  0  0  0  0];
    %             1 2 3 4 5 6 7 8 9 10 11 12 13
    % Plotting options
    % 1 - raw data
    % 2 - Average of each tc at end
    % 3 - Plot control
    % 4 - Filtered tcs
    % 5 - Fitting beta with different models
    % 6 - simulated tcs
    % 7 - simulated end points (temperature profile at end of dataset)
    % 8 - tcs sim vs measured end points
    % 9 - Compare refined to unrefined solution
```

```
    % 10 - Simulated temperature with optimal input
    % 11 - Optimal input at end of profile
    % 12 - Plot optimal input for both full inverse and simple inverse
    % 13 - Plot beta values and interpolated beta curve



    if (plottingOn(1) == 1)
        figure; plot(modelData.Tsim/60,modelData.tcs)
        set(gca,'FontSize',14)
        title('Raw data');
        xlabel('Time (min)'); ylabel('Temp. ($C^\circ$)')
    end
    tcavgs = mean(modelData.tcs(end-10:end,:));
%     orig_xpoints = 1:7/(modelData.numtcs-1):8;
    orig_xpoints = modelData.tclocs+1;

    model_xpoints = 1:7/(modelpoints-1):8;
    out.model_xpoints = model_xpoints;
    if (plottingOn(2) == 1)
        figure; plot(orig_xpoints,tcavgs)
        set(gca,'FontSize',18)
        xlabel('$x$ (in)'); ylabel('Temperature ($C^\circ$)')
        axis([1 8 95 135])
        title('Measured temperature at end')
    end
    if (plottingOn(3) == 1)
        figure;
        plot(modelData.control)
        title('Control Signal');
    end
    [bz,az] = butter(2,(1/25)/2,'low'); % Filter at 1/15 Hz
    tcsfilt = filtfilt(bz,az,modelData.tcs);
    if (plottingOn(4) == 1)
        figure; plot(modelData.Tsim/60, tcsfilt);
        set(gca,'FontSize',14)
        xlabel('Time (min)'); ylabel('Temp. ($C^\circ$)')
        title('Filtered tcs')
    end

    [Cp, beta, alpha, Amat, Bmat, Cmat, Dmat, qin_scale, beta2,beta3,beta4] = ...
            getABCD(modelData.fitType,modelData.xsoln,modelData.numtcs, modelpoints,modelData.tclocs);
    out.alpha = alpha;
    out.beta = beta;
    out.gamma = Cp;

    if (plottingOn(5) == 1)
        figure; plot(model_xpoints,beta,model_xpoints,beta2,model_xpoints,beta4,modelData.tclocs+1,beta3,'
            ro');
        set(gca,'FontSize',14)
        xlabel('$x$ (in)'); ylabel('\beta value')
        xlim([1 8]); legend('Piecewise cubic','Machine Learning','Spline','Original \beta')
    end
```

```
Sysfake = ss(Amat,Bmat,Cmat,Dmat);
out.Amat = Amat;
out.Bmat = Bmat;
out.Cmat = Cmat;
out.Dmat = Dmat;
controlScale = modelData.inputV*qin_scale;
controlinput = controlScale*modelData.control;
out.controlScale = controlScale;

X0 = zeros(modelpoints,1);
[Yout,Tout,~] = lsim(Sysfake,controlinput,modelData.Tsim,X0);
out.Yout = Yout;
if (plottingOn(6) == 1)
    figure; plot(Tout/60,Yout)
    set(gca,'FontSize',14)
    xlabel('Time (min)'); ylabel('Temp. ($C^\circ$)')
end
if (plottingOn(7) == 1)
    figure;
    plot(modelData.tclocs+1,mean(modelData.tcs(end-20:end,:)),'k*');hold on
    plot(modelData.tclocs+1,modelData.origYout(end,:),'ro')
    plot(model_xpoints,Yout(end,:));
    set(gca,'FontSize',14)
    xlabel('$x$ (in)'); ylabel('Temp. ($C^\circ$)')
    legend('Measured','Predicted','Refined')
    xlim([1 8])

    figure;
    plot(model_xpoints,Yout(end,:));
    set(gca,'FontSize',14)
    xlabel('$x$ (in)'); ylabel('Temp. ($C^\circ$)')
    legend('Refined model')
    xlim([1 8])
end

if (plottingOn(8) == 1)
    figure;
    plot(orig_xpoints,mean(modelData.tcs(end-20:end,:))+modelData.tcs_offset,'ro',...
        orig_xpoints,modelData.origYout(end,:)+modelData.tcs_offset,'s',...
        model_xpoints,Yout(end,:)+modelData.tcs_offset,'b')
    set(gca,'FontSize',14)
    legend('Experiment','Simulation','Simulation extrapolated','Location','North')
    xlabel('$x$ (in)'); ylabel('Temp. ($C^\circ$)')
    xlim([1 8])
end
if (plottingOn(9) == 1)
    figure;
    plot(model_xpoints,Yout(end,:),orig_xpoints,modelData.origYout(end,:),'o')
    set(gca,'FontSize',14)
    xlim([1 8])
    legend('Refined model',[num2str(modelData.numtcs) '-point model'],'Location','SouthEast')
    xlabel('Thermocouple $\#$'); ylabel('Temp. ($C^\circ$)')
end
```

```
%% Close loop and find steady state control input

rampreftemp = modelData.rampreftemp;
if length(rampreftemp) < length(modelData.Tsim2)
        rampreftemp = [ rampreftemp;rampreftemp(end)...
    *ones(length(modelData.Tsim2)-length(modelData.rampreftemp),1)];
rampreftemp = rampreftemp-rampreftemp(1);
elseif length(rampreftemp) > length(modelData.Tsim2)
    rampreftemp = rampreftemp(1:length(modelData.Tsim2));
end
yd = rampreftemp;
ydmat = repmat(yd',modelpoints,1);
ydd = diff(yd)/.31;
ydd(end+1) = ydd(end);
yddmat = repmat(ydd',modelpoints,1);

Bmat2 = diag(Bmat);
uopt =(Cmat*Bmat2)\(yddmat - Cmat*Amat*ydmat);
out.uopt = uopt;

Sysfake2 = ss(Amat,Bmat2,Cmat,zeros(size(Bmat2)));

X0 = zeros(modelpoints,1);
[Yout2,Tout2,~] = lsim(Sysfake2,uopt',modelData.Tsim2,X0);

if (plottingOn(10) == 1)
    % Simulated temperature with optimal input
    figure; plot(Tout2/60,Yout2)
    set(gca,'FontSize',14)
    xlabel('Time (min)'); ylabel('Temp. ($C^\circ$)')
end

if (plottingOn(11) == 1)
    % Optimal input at end of profile
    figure; plot(model_xpoints,controlScale*uopt(:,end))
    set(gca,'FontSize',14)
    xlabel('$x$ (in)'); ylabel('Optimal control input $(V^2/m)$')
    xlim([1 8])
end
Tc_modeled = Yout(end,:);
input_avg_end = mean(controlinput(end-20:end,:));
Tdesired = yd(end)*ones(size(Tc_modeled));
udes = Tdesired .* input_avg_end ./ Tc_modeled;

if (plottingOn(12) == 1)
    % Plot optimal input for both full inverse and simple inverse
    figure; plot(model_xpoints,udes,model_xpoints,mean(uopt(:,end-20:end),2))
    set(gca,'FontSize',14)
    xlabel('$x$ (in)'); ylabel('Optimal control input')
    legend('Simple inverse','Modeled inverse')
    xlim([1 8])
end

origbeta_scaled = (modelData.numtcs-1)/(modelpoints-1) * modelData.beta;
```

```
    if (plottingOn(13) == 1)
        % Plot beta values and interpolated beta curve
        figure; plot(modelData.tclocs+1,origbeta_scaled,'ro',model_xpoints,beta)
        xlabel('$x$ (in)'); ylabel('$\beta$')
        xlim([1 8])
    end

    out.desiredProfile = mean(uopt(:,end-20:end),2) /qin_scale;
    out.qin_scale = qin_scale;
end
```

C.6 Supporting Chapter 3: estimating desired power supporting function

```
function [Cp, beta, alpha, Amat, Bmat, Cmat, Dmat,qin_scale,beta2,beta3,beta4] =...
                getABCD(fitType,xsoln,numtcs,modelpoints,tclocs)
    if nargin < 4
        modelpoints = numtcs;
        tclocs = [];
    end
    %   fitType
    %     1 - alpha and gamma constant
    %     2 - alpha variable, gamma constant
    %     3 - alpha constant, gamma variable
    %     4 - alpha = 0, gamma is constant
    %     5 - alpha and gamma constant
    %         Betas are a linear function of temperature
    qin_scale = (numtcs-1)/(modelpoints-1);
        beta2 = 0;
        beta3 = 0;
    if fitType == 1
        if isempty(tclocs)
            Cp = xsoln(end) * (numtcs-1)/(modelpoints-1);
            betat = xsoln(1:end-2);
            beta = interp1(1:numtcs,betat,1:(numtcs-1)/(modelpoints-1):numtcs)* (numtcs-1)/(modelpoints-1)
                ;
            alpha =  xsoln(end-1)*((modelpoints-1)/(numtcs-1));
            qin_scale = (numtcs-1)/(modelpoints-1);
        else
            Cp = xsoln(end) * (7/(modelpoints-1))/(tclocs(2)-tclocs(1));
            betat = xsoln(1:end-2);
            beta3 = betat * (7/(modelpoints-1))/(tclocs(2)-tclocs(1));
            beta = interp1(tclocs,betat,0:7/(modelpoints-1):7,'pchip','extrap') * ...
                            (7/(modelpoints-1))/(tclocs(2)-tclocs(1));
            beta4 = interp1(tclocs,betat,0:7/(modelpoints-1):7,'spline','extrap') * ...
                            (7/(modelpoints-1))/(tclocs(2)-tclocs(1));

            gprR = fitrgp([1:numtcs]',betat, 'FitMethod','exact','PredictMethod','exact', ...
                      'KernelFunction','squaredexponential');
            beta2 = predict(gprR,[1:(numtcs-1)/(modelpoints-1):numtcs]')* (numtcs-1)/(modelpoints-1);

            alpha =  xsoln(end-1)*(tclocs(2)-tclocs(1))/(7/(modelpoints-1));
            qin_scale = (7/(modelpoints-1))/(tclocs(2)-tclocs(1));
        end
```

```
    Amat = [(-beta(1)-alpha)/Cp, alpha/Cp, zeros(1,modelpoints-2)];
    for idx = 2:modelpoints-1
        Amat = [Amat;...
                zeros(1,idx-2), alpha/Cp, ...
                (-beta(idx)-2*alpha)/Cp, alpha/Cp,...
                zeros(1,modelpoints-idx-1)];
    end
    Amat = [Amat;...
             zeros(1,modelpoints-2), alpha/Cp, ...
             (-beta(end)-alpha)/Cp];
    Bmat = 1/Cp*ones(modelpoints,1);
    Cmat = eye(modelpoints);
    Dmat = zeros(modelpoints,1);
end
%     2 - alpha variable, gamma constant
if fitType == 2
    Cp = xsoln(end)*(numtcs-1)/(modelpoints-1);
    beta = xsoln(1:2:end-1);
    beta = interp1(1:numtcs,beta,1:(numtcs-1)/(modelpoints-1):numtcs)* (numtcs-1)/(modelpoints-1);
    alpha = xsoln(2:2:end-1);
    alpha = interp1(1:numtcs-1,alpha,1:(numtcs-2)/(modelpoints-2):numtcs-1) * (numtcs-1)/(modelpoints
        -1);

    Amat = [(-beta(1)-alpha(1))/Cp, alpha(1)/Cp, zeros(1,modelpoints-2)];
    for idx = 2:modelpoints-1
        Amat = [Amat;...
                zeros(1,idx-2), alpha(idx-1)/Cp, ...
                (-beta(idx)-alpha(idx-1)-alpha(idx))/Cp, alpha(idx)/Cp,...
                zeros(1,modelpoints-idx-1)];
    end
    Amat = [Amat;...
             zeros(1,modelpoints-2), alpha(end)/Cp, ...
             (-beta(end)-alpha(end))/Cp];
    Bmat = 1/Cp*ones(modelpoints,1);
    Cmat = eye(modelpoints);
    Dmat = zeros(modelpoints,1);
end
%     3 - alpha constant, gamma variable
if fitType == 3
    beta = xsoln(1:2:end-1);
    beta = interp1(1:numtcs,beta,1:(numtcs-1)/(modelpoints-1):numtcs)* (numtcs-1)/(modelpoints-1);
    Cp = xsoln(2:2:end-1);
    Cp = interp1(1:numtcs,Cp,1:(numtcs-1)/(modelpoints-1):numtcs)* (numtcs-1)/(modelpoints-1);
    alpha =  xsoln(end)*((modelpoints-1)/(numtcs-1));

    Amat = [(-beta(1)-alpha)/Cp(1), alpha/Cp(1), zeros(1,modelpoints-2)];
    for idx = 2:modelpoints-1
        Amat = [Amat;...
                zeros(1,idx-2), alpha/Cp(idx), ...
                (-beta(idx)-2*alpha)/Cp(idx), alpha/Cp(idx) ,...
                zeros(1,modelpoints-idx-1)];
    end
    Amat = [Amat;...
             zeros(1,modelpoints-2), alpha/Cp(idx), ...
```

```
                (-beta(end)-alpha)/Cp(idx)];
        Bmat = 1./Cp;
        Cmat = eye(modelpoints);
        Dmat = zeros(modelpoints,1);
    end
    %     4 - alpha = 0, gamma is constant
    if fitType == 4
        Cp = xsoln(end) * (numtcs-1)/(modelpoints-1);
        beta = xsoln(1:end-1);
        beta = interp1(1:numtcs,beta,1:(numtcs-1)/(modelpoints-1):numtcs)* (numtcs-1)/(modelpoints-1);
        alpha = 0;

        Amat = [(-beta(1))/Cp, zeros(1,modelpoints-1)];
        for idx = 2:modelpoints-1
            Amat = [Amat;...
                    zeros(1,idx-1), ...
                    (-beta(idx))/Cp,...
                    zeros(1,modelpoints-idx)];
        end
        Amat = [Amat;...
                 zeros(1,modelpoints-1), ...
                 (-beta(end))/Cp];
        Bmat = 1/Cp*ones(modelpoints,1);
        Cmat = eye(modelpoints);
        Dmat = zeros(modelpoints,1);
    end
end
```

C.7 Supporting Chapter 3: optimal multiple heater solution

```
clc; close all; clear all
%%
out = estimatingDesiredPower(1,1);
out1 = estimatingDesiredPowerRefined(out,500);
close all;
%%
desiredProfile = out1.desiredProfile; %out1.uopt(:,end);

figure; plot(desiredProfile);

npoints = length(desiredProfile);

Am = out1.Amat;
Bm = diag(out1.Bmat);

x = {};
sqerror = [];
sqerror_c = [];
tabits = 1:10;
colors = jet(length(tabits));
figure; plot(out1.model_xpoints,desiredProfile,'k'); hold on

options = optimoptions('fmincon','Display','iter','Algorithm','sqp');
tempErr = zeros(size(tabits));
```

```
tempErr_rms = zeros(size(tabits));
tempErr_max = zeros(size(tabits));
gprProfileFit = fitrgp([1:length(desiredProfile)]',desiredProfile, 'FitMethod','exact','PredictMethod','
    exact', ...
                    'KernelFunction','squaredexponential');
x_solns = {};
cut_profiles = [];
for numtabs = tabits
    numtabs
    xinit = (npoints)/numtabs:(npoints)/numtabs:npoints;
    xinit = round(xinit);

    % Define constraints
    A = -1*eye(numtabs);
    A = [A;eye(numtabs)];
    b = zeros(numtabs,1);
    b = [b;npoints*ones(numtabs,1)];
    Aeq = zeros(1,numtabs);
    Aeq(end) = 1;
    beq = npoints;
    [x{end+1},sqerror_c(end+1),exitflag,output] = ...
        fmincon(@(x)cutHeaterFun(x,gprProfileFit),xinit,A,b,Aeq,beq,[],[],[],options);
    x1 = round(x{numtabs});
    x1
    % sort tabs
    x1 = sort(x1);
    % remove zero tabs
    while x1(1) == 0
        if length(x1) < 2
            error('Solution invalid!!');
        end
        x1 = x1(2:end);
    end

    % remove repeated tabs
    idx = 2;
    while idx <= length(x1)
        if x1(idx) == x1(idx-1)
            x1(idx) = [];
        else
            idx = idx + 1;
        end
    end
    x1
    x_solns{end+1} = x1;
    tabp1 = 1;
    cut_profile = zeros(size(desiredProfile));
    for idx = 1:length(x1)
        profilePiece = desiredProfile(tabp1:x1(idx));
        meanInSection = mean(profilePiece);
        cut_profile(tabp1:x1(idx)) = meanInSection;
        tabp1 = x1(idx)+1;
    end
    if numtabs == 3
```

```
        cut_profile3 = cut_profile;
        plot(out1.model_xpoints,cut_profile3);
        xlim([1 8]);
        xlabel('$x$ (in)'); ylabel('Power $(V^2/m)$');
        legend('Desired profile','Optimal profile with 3 heaters')
%         figure; plot(out1.model_xpoints,cut_profile-desiredProfile)
    end

    if numtabs ==5
        cut_profile5 = cut_profile;
        plot(out1.model_xpoints,cut_profile5);
        xlim([1 8]);
        xlabel('$x$ (in)'); ylabel('Power $(V^2/m)$');
        legend('Desired profile','Optimal profile with 3 heaters','Optimal profile with 5 heaters')
    end



    sqerror(end+1) = rms(abs(cut_profile-desiredProfile));
    cut_profiles(end+1,:) = cut_profile;
    xss = -(Am'*Am)\(Am'*Bm*cut_profile*out1.qin_scale);
    tempErr(numtabs) = max(xss)-min(xss);
    tempErr_rms(numtabs) = rms(abs(xss-mean(xss)));
    tempErr_max(numtabs) = max(abs(xss-mean(xss)));
end


figure; plot(tabits,sqerror);
xlabel('Number of tabs'); ylabel('RMS error in Power $(V^2/m)$')


%% Analytical solution to cut heater error for a line y = mx
% desiredProfile = desiredProfile(1) +...
%     mean(abs((diff(out1.desiredProfile))))*[1:length(desiredProfile)]';
%
% Solution is as follows:
%  rms = sqrt(1/L * Int( (f(x))^2)dx,0,L) )
%
%  f(x) = H/(2n) - mx
%  f(x) is dicontinuous repeating, so integral goes to
%       n*Integral( f(x)^2 dx from 0 to L/n)
return
H = desiredProfile(end)-desiredProfile(1);
L = length(desiredProfile)-1;
m_s = H/L;
n = [1:100]';
% errorTot = H./(2.*n).*L - m_s./(4.*n).*L.^2;
rmsinner = n./(3*m_s*L).*(-(H./(2.*n)-m_s*L./n).^3 + (H./(2.*n)).^3);
rms_n = sqrt(rmsinner);
figure; plot(tabits,sqerror,n,rms_n,'ro')
```

C.8 *Supporting Chapter 3: optimal multiple heater solution supporting function*

```
function sqerror = cutHeaterFun(xvec,gprProfileFit)
%CUTHEATERFUN Takes in either a machine learned profile, or vector profilt,
%and a vector of achieved profile, and returnes the error between the two
%profiles.
%
% sqerror = cutHeaterFun(xvec,gprProfileFit)
%
% Input -
%   xvec - vector of achieved power in bondline
%   gprProfileFit - profile (vector or machine learning structure) for
%                   desired profile
% Output -
%   sqerror - squared error between the two profiles

if isnumeric(gprProfileFit)  % Passed in a vector
    xvec = round(xvec);
    tabp1  = 1;
    sqerror = zeros(length(xvec),1);
    numpoints = length(gprProfileFit);
    for idx = 1:length(xvec)
        profilePiece = gprProfileFit(tabp1:xvec(idx));
        Atemp = ones(size(profilePiece));
        lambda = (Atemp'*Atemp)\(Atemp'*profilePiece);
        sqerror(idx) = sum((profilePiece-lambda).^2);
        tabp1 = xvec(idx)+1;
    end
    sqerror = sqrt(1/numpoints*sum(sqerror));

else
    % Calculate optimal leval and then error^2 by tab
    tabp1 = 1;
    sqerror = zeros(length(xvec),1);
    numpoints = 2*xvec(end)/length(xvec);
    for idx = 1:length(xvec)
        profilePiece = predict(gprProfileFit,[tabp1:(xvec(idx)-tabp1)/numpoints:xvec(idx)]');
        Atemp = ones(size(profilePiece));
        lambda = (Atemp'*Atemp)\(Atemp'*profilePiece);
        sqerror(idx) = sum((profilePiece-lambda).^2);
        tabp1 = xvec(idx)+(xvec(idx)-tabp1)/numpoints;
    end
    sqerror = sqrt(1/numpoints*sum(sqerror));
end
```

C.9 Supporting Chapter 3: optimal multiple tabbed heater

```
function Results = tabbedOptSolver(m_inches)

    Results = {};
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    load('cutheatersoln_step_ramp_fixed.mat');

    opts.tabthickness = 5;
    opts.tabgap = 5;
    opts.n = 140;
```

```
    opts.m = m_inches*20;
    opts.n_inches = 7;
    opts.m_inches = m_inches;

    opts.refinedSimPoints = 500;

    rTabgap = ceil(opts.tabgap*opts.refinedSimPoints/(opts.n+1));
    mintab = ceil(0.5*opts.refinedSimPoints/opts.n_inches);

    %% Find best solution for 2 tabs
    numtabs = 2;
    besttabs = zeros(1,numtabs);
    lowestmaxerror = 10000;

    for tab1 = 1:opts.refinedSimPoints-opts.tabgap-1
        tab1
        opts.tablocs = [tab1 opts.refinedSimPoints];
        tabson = ones(1,numtabs);
        [x,y,T,xpoints,graddata,BCidx_out]= ...
          LaplaceExplicitGapLocs(opts.n,opts.m,opts.tablocs,tabson,opts.tabthickness,opts.tabgap,1,1);
%         plot(BCidx_out); hold on

        maxerror = getMaxError(opts,gprProfileFit,out1,Am,Bm);


        if maxerror < lowestmaxerror
            besttabs = opts.tablocs
            lowestmaxerror = maxerror
        end

    end
    Results{1,1} = 2;
    Results{1,2} = besttabs;
    Results{1,3} = lowestmaxerror;
    %% Find best solution for 3 tabs onward
    for numtabs = 3:7
        tablocs = round(opts.refinedSimPoints/numtabs):round(opts.refinedSimPoints/numtabs):opts.
            refinedSimPoints+15;
        tablocs(end) = opts.refinedSimPoints;
        opts.tablocs = tablocs;
        % Initialize loop with error with even tabs
        %%%%%%%%%%%%%%%%%%
        tabson = ones(1,numtabs);
        [~,~,~,~,~,BCidx_out]= ...
          LaplaceExplicitGapLocs(opts.n,opts.m,opts.tablocs,tabson,opts.tabthickness,opts.tabgap,1,1);
%         figure; plot(BCidx_out); drawnow
        maxerror = getMaxError(opts,gprProfileFit,out1,Am,Bm);

        %%%%%%%%%%%%%%%%%%
        maxerrors = [maxerror; 10000*ones(2*(numtabs-1),1)];
        taboptions = [opts.tablocs;zeros(2*(numtabs-1),length(opts.tablocs))];
        stillSearching = 1;
        stepsize = 40;
        if numtabs > 5
```

```
        stepsize = 20;
    end

    while stillSearching == 1
        % Loop with decreasing step sizes
        for gapnum = 1:numtabs-1
            % Take a step left and get maxerror
            opts.tablocs = taboptions(1,:);
            if (gapnum == 1)
                if (taboptions(1,1)-stepsize) >  (mintab + round(rTabgap/2))
                    opts.tablocs = taboptions(1,:);
                    opts.tablocs(1) = taboptions(1,1)-stepsize;
                    maxerrors(2) = getMaxError(opts,gprProfileFit,out1,Am,Bm);
                    taboptions(2,:) = opts.tablocs;
                    [~,~,~,~,BCidx_out]= ...
                      LaplaceExplicitGapLocs(opts.n,opts.m,opts.tablocs,tabson,opts.tabthickness,opts.
                          tabgap,1,1);
%                      plot(BCidx_out); drawnow
                end
            else
                if (taboptions(1,gapnum)-stepsize) >  (taboptions(1,gapnum-1) + mintab + rTabgap)
                    opts.tablocs = taboptions(1,:);
                    opts.tablocs(gapnum) = taboptions(1,gapnum)-stepsize;
                    maxerrors(2*gapnum) = getMaxError(opts,gprProfileFit,out1,Am,Bm);
                    taboptions(2*gapnum,:) = opts.tablocs;
                    [~,~,~,~,BCidx_out]= ...
                      LaplaceExplicitGapLocs(opts.n,opts.m,opts.tablocs,tabson,opts.tabthickness,opts.
                          tabgap,1,1);
%                      plot(BCidx_out); drawnow
                end
            end

            % Take a step right and get max error
            opts.tablocs = taboptions(1,:);
            if (gapnum == (numtabs-1))
                if (taboptions(1,gapnum)+stepsize) <  opts.refinedSimPoints+ceil(1*opts.
                     refinedSimPoints/opts.n)-mintab-floor(rTabgap/2)
                    opts.tablocs = taboptions(1,:);
                    opts.tablocs(gapnum) = taboptions(1,gapnum)+stepsize;
                    maxerrors(end) = getMaxError(opts,gprProfileFit,out1,Am,Bm);
                    taboptions(end,:) = opts.tablocs;
                    [~,~,~,~,BCidx_out]= ...
                      LaplaceExplicitGapLocs(opts.n,opts.m,opts.tablocs,tabson,opts.tabthickness,opts.
                          tabgap,1,1);
%                      plot(BCidx_out); drawnow

                end

            else
                if (taboptions(1,gapnum)+stepsize) <  (taboptions(1,gapnum+1) - mintab - rTabgap)
                    opts.tablocs = taboptions(1,:);
                    opts.tablocs(gapnum) = taboptions(1,gapnum)+stepsize;
                    maxerrors(1+2*gapnum) = getMaxError(opts,gprProfileFit,out1,Am,Bm);
                    taboptions(1+2*gapnum,:) = opts.tablocs;
```

```
                        [~,~,~,~,~,BCidx_out]= ...
                          LaplaceExplicitGapLocs(opts.n,opts.m,opts.tablocs,tabson,opts.tabthickness,opts.
                              tabgap,1,1);
%                          plot(BCidx_out); drawnow
                    end
                end
            end
            [lowest,l_idx] = min(maxerrors);
            maxerrors
            if (l_idx == 1)
                if stepsize == 1
                    disp('Converged!!!!!');
                    maxerrors
                    taboptions
                    stillSearching = 0;
                    Results{numtabs-1,1} = numtabs;
                    Results{numtabs-1,2} = taboptions(1,:);
                    Results{numtabs-1,3} = maxerrors(1);
                    break;
                end
                stepsize = round(stepsize/2)
                % should break this loop and go to the next step size
            else
                taboptions(1,:) = taboptions(l_idx,:);
                maxerrors(1) = maxerrors(l_idx);
                stillSearching = 1;
            end
            taboptions(1,:)
        end
    end

end

function maxerror = getMaxError(opts,gprProfileFit,out1,Am,Bm)
    [maxerror, rmserror,~,~,Pach_avg,xpoints,~] =...
        singleHeaterTabDesignSim(opts,gprProfileFit,0);
    tabbed_profile = interp1(xpoints+1,Pach_avg,out1.model_xpoints,'spline')';
    xss_tabbed = -(Am'*Am)\(Am'*Bm*tabbed_profile*out1.qin_scale);
    maxerror = max(abs(xss_tabbed-mean(xss_tabbed)));

end
```

C.10 Supporting Chapter 3: Solving for single tabbed heater design

```
% This function is designed to find the optimal solution to individualized
% control of a multi-zone embedded heater. This is done by controlling each
% zone to minimize the error between power generated in that zone and the
% desired power output in that zone.
function [maxerror, rmserror,Pdes_out,Pachieved,Pachievedavg,x,tabtimes] = ...
    singleHeaterTabDesignSim(opts,gprProfileFit,isPlotting)

% LaplaceExplicitGapLocs(n,m,...
%                     tablocs,tabCombos(mainIdx,:),tabthickness,tabgap,0);
    tablocs = opts.tablocs;
```

```
    tabthickness = opts.tabthickness;
    tabgap = opts.tabgap;
    n = opts.n;
    m = opts.m;
    n_inches = opts.n_inches;
    m_inches = opts.m_inches;
    refinedSimPoints = opts.refinedSimPoints;

    if nargin < 3
        isPlotting = 1;
    end
    numtabs = length(tablocs);
    % Create data structure to hold simulation data
    %%%%%%%%%%%%%%%%%%
    allcombos = {};
    for idx = 1:numtabs
        combos = combnk([1:numtabs],idx);
        for idx2 = 1:size(combos,1)
            allcombos{end+1} = combos(idx2,:);
        end
    end
    tabCombos = zeros(length(allcombos)+1,numtabs);
    % length(tabCombos)
    for idx = 1:length(allcombos)
        tabCombos(idx,allcombos{idx}) = 1;
    end

    Psim = cell(length(tabCombos),3);
    for mainIdx = 1:length(tabCombos)
        Psim{mainIdx,1} = tabCombos(mainIdx,:);
    end
    %%%%%%%%%%%%%%%%%%%%%%


    % load simulations

    % load target profile and create desired power profile
%     load Pdes_poly % contains information about desired polynomial
    % Form the desired profile by evaluating polynomial at same points as
    % modeled. Need to also remove tails where polynomial is incorrect.
%     x = Psim{1,2};
    x = 0:7/(n):7;

%     bondZoneWidth = size(Psim{1,3},1);
    bondZoneWidth = m/m_inches+1;

%     Pdes = polyval(P,x);
%     idxlow = find(x < 1);
%     idxhigh = find(x > 8);
%     Pdes(idxlow) = Pdes(max(idxlow)+1);
%     Pdes(idxhigh) = Pdes(min(idxhigh)-1);
%
%     [b,a] = butter(2,.1); % filter to clean up head and tail
%     Pdes = filtfilt(b,a,Pdes);
```

```
    pointInFit = length(gprProfileFit.Alpha);
%     pointInFit =500;
%     Pfun=@(x) 800 - 100*cos(2*pi*(x+10)/500);
%     Pdes = Pfun(x'*pointInFit/x(end));
    Pdes = predict(gprProfileFit,x'*pointInFit/x(end));
    Pdes = Pdes';
    Pdes_out = Pdes;
    %%%%%%%%%%%%%%%%
    if isPlotting
        figure; plot(x+1,Pdes,'k');
        set(gca,'FontSize',18);
        xlabel('$x$ (in)')
        ylabel('Target power $(V/m)^2$');
    end
    %%%%%%%%%%%%%%%%
    Pdes = repmat(Pdes,bondZoneWidth,1);

%     tab_gap = 1;


    Kpd = 1e-04;
    Kid = 1e-05;

    tabtimes = zeros(numtabs,1);

    zpoints = round(tablocs * length(x)/tablocs(end));
    if zpoints(1) == 0
        zpoints(1) = 1;
    end
    zpoints = [1 zpoints];
    if (isnan(zpoints(2)))
       blah = 1;
    end
    zoneAvgTarget = getZoneAverages(Pdes,numtabs,2,zpoints);
    zoneErrors_old = zeros(numtabs,1);

%     [~,y,~,~,~] = LaplaceExplicitGapLocs(90,60,...
%                   tablocs,zeros(1,numtabs),3,3,1);
%     [~,y,~,~,~] = LaplaceExplicitGapLocs(n,m,...
%                   tablocs,zeros(1,numtabs),tabthickness,tabgap,1);
    heaterlength = m;
    heateredge1 = (round(heaterlength/2)) - floor(bondZoneWidth/2);
    heateredge2 = (round(heaterlength/2)) + floor(bondZoneWidth/2);

    eps = 1e-6;
    maxits = 3000;

    if isPlotting
        figure;
        colors = jet(maxits);
    end

    for idxcount = 1:maxits
```

```
        [graddata,Psim] = getHeaterSolution(opts,tabtimes,Psim);
        zoneAvgSim = getZoneAverages(graddata, numtabs,2,zpoints);
        zoneErrors = zoneAvgTarget - zoneAvgSim;
        % Calculate new tabtimes based on error in zones

        tabtimes = tabtimes + Kpd*zoneErrors-Kid*zoneErrors_old;
        maxtimechange = max(abs(zoneErrors-zoneErrors_old));
        if(maxtimechange < eps)
            fprintf('Converged after %d iterations!\n',idxcount);
            break;
        end
        zoneErrors_old = zoneErrors;
        % Handle saturation
        for idx = 1:length(tabtimes)
            if (tabtimes(idx)>1) tabtimes(idx)= 1; end
            if (tabtimes(idx)<0) tabtimes(idx)= 0; end
            if (zoneErrors_old > zoneAvgTarget)
                zoneErrors_old = zoneAvgTarget;
            end
        end
        if isPlotting
            plot(tabtimes,'*','Color',colors(idxcount,:)); hold on
        end
    end

    if idxcount == maxits
        fprintf('Didn''t converge after %d iterations!\n',idxcount);
    end

    if isPlotting
        set(gca,'FontSize',14);
        title('Times for each zone converging');
        xlabel('$t_n$'); ylabel('time (s)');
%         [X,Y] = meshgrid(x,y);
        [X,Y] = meshgrid(x,y(heateredge1:heateredge2));
%         surf_from_scatter(X,Y,graddata);
%         surf_from_scatter(X,Y,graddata);
        surf_from_scatter(X,Y,(graddata-Pdes));
%         surf_from_scatter(X,Y,(graddata));

        set(gca,'FontSize',14);
        xlabel('X (in)');
        ylabel('Y (in)');
        zlabel('Error  in power generated $(V/m)^2$');
    end

%     maxerror = max(max(abs((graddata-Pdes)./Pdes)))
%     rmserror = rms((graddata(:)-Pdes(:))./Pdes(:))
    Pachieved = graddata(round(size(graddata,1)/2),:);
    Pachievedavg = mean(graddata,1);
    maxerror = max(max(abs(graddata-Pdes)));
    rmserror = rms(abs(graddata(:)-Pdes(:)));
end
```

```
% break data into equal zones and return average value for each zone
% Input
%   data - 2D array of data
%   zone - number of zones to get averages on
%   dim2split - dimension to cut zones across
% Output
%   zoneAvgs - 1D array of average value of each zone
function zoneAvgs = getZoneAverages(data,zones,dim2split,boundaries)
    if (nargin < 4)
        boundaries = 1:((size(data,dim2split)-1)/zones):size(data,dim2split)+1;
        boundaries = round(boundaries);
        boundaries(end) = size(data,dim2split);
    end

    zoneAvgs = zeros(zones,1);

    for idx = 1:length(zoneAvgs)
        if dim2split == 1
            zoneAvgs(idx) = ...
                mean(mean(data(boundaries(idx):boundaries(idx+1),:)));
        end
        if dim2split == 2
            try
            zoneAvgs(idx) = ...
                mean(mean(data(:,boundaries(idx):boundaries(idx+1))));
            catch
                blah = 1;
            end
        end
    end
end

% Convert tab times and run simulation on those times
% Input
%   tabtimes - 1D vector of time that each tab is active
function [graddata,Psim] = getHeaterSolution(opts,tabtimes,Psim)
    tablocs = opts.tablocs;
    tabthickness = opts.tabthickness;
    tabgap = opts.tabgap;
    n = opts.n;
    m = opts.m;
    n_inches = opts.n_inches;
    m_inches = opts.m_inches;
    bondZoneWidth = m/m_inches+1;
    xpoints = length(0:n_inches/(n):n_inches);

% Set number of tabs based on number of tabtimes
    numtabs = length(tabtimes);

    if isempty(find(tabtimes )) % times are all zeros
        graddata = zeros(bondZoneWidth,xpoints);% 0*Psim{1,3};
    else
```

```
        graddata = zeros(bondZoneWidth,xpoints);
        currentActiveTabs = zeros(1,numtabs);
        currentActiveTabs(find(tabtimes)) = 1;
        lastMinTime = 0;
        while (length(find(currentActiveTabs)) >= 1)
            minTimeLeft = min(tabtimes(find(currentActiveTabs)));
            idxActiveTabs = 0;
            for idx = 1:size(Psim,1)
               if (isequal(Psim{idx,1},currentActiveTabs))
                   idxActiveTabs = idx;
               end
            end
            % if Psim is empty, run sim and save it
            if(isempty(Psim{idxActiveTabs,3}))
                [x2,y2,T,xpoints,graddata_temp] = LaplaceExplicitGapLocs(n,m,...
                  tablocs,Psim{idxActiveTabs,1},tabthickness,tabgap,0);
                graddata_temp = graddata_temp(round(m/2-m/m_inches/2):round(m/2+m/m_inches/2),:);
                Psim{idxActiveTabs,2} = xpoints;
                Psim{idxActiveTabs,3} = graddata_temp;
            end

            graddata_temp = Psim{idxActiveTabs,3};
            graddata = graddata + graddata_temp*(minTimeLeft-lastMinTime);
            lastMinTime = minTimeLeft;
            tabsToRemove = find(tabtimes == minTimeLeft);
            currentActiveTabs(tabsToRemove) = 0;
        end
    end
end
```