NUMERICAL MODELING AND SIMULATING THERMAL PERFORMANCE OF PRINTED CIRCUIT BOARDS

by

Paul R. Ziegenfelder

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

of

MASTER OF SCIENCE

in

Electrical Engineering

UTAH STATE UNIVERSITY
Logan, Utah

2022

ABSTRACT

Numerical Modeling and Simulating Thermal Performance of Printed Circuit Boards

by

Paul R. Ziegenfelder, Master of Science

Utah State University, 2022

Major Professor: Hongjie Wang, Ph.D.
Department: Electrical and Computer Engineering

Increasing power requirements for printed circuit boards (PCBs) require more holistic modeling of their thermal performance. This can be difficult without the use of expensive and complicated software. This research developed an open-source software solution that can be used for quick evaluation of the thermal performance for a given PCB design using common PCB design files such as gerber and NC drill files. The intent was not to design a full featured analysis tool that can completely replace existing software solutions, many of which can perform very detailed and complex simulations but instead to give a designer confidence in their PCB design and give indications of areas of concern that could require revision.

The steady state solution was based on heat transfer theory and finite element theory. A model was developed for analyzing rectangular 3D bodies of known material properties and boundary conditions. Physical verification led to adjustments of the model with the final output on average of 8% error compared to physical results.

(195 pages)

PUBLIC ABSTRACT

Numerical Modeling and Simulating Thermal Performance of Printed Circuit Boards

Paul R. Ziegenfelder

This research investigates how heat transfer theory can be applied to evaluate the thermal performance of printed circuit board (PCB) designs. A model was developed for correlating theory to the physical parameters relating to the construction of the PCB and a computer application was developed to perform the thermal analysis. The application allows the importation of common PCB design files for a straightforward and quick analysis of a PCB design's thermal performance. The application is open-source to allows for further development and free access from the public.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

DC	direct current
FEA	finite element analysis
Gr	Grashof number
GUI	graphic user interface
htc	heat transfer coefficient
IC	integrated circuit
IPC	Institute of Printed Circuits
PCB	printed circuit board
Pr	Prandtl number
Ra	Raleigh number

CHAPTER 1
INTRODUCTION

The use of the printed circuit board (PCB) in electronic equipment is ubiquitous across industries and has been an enabling technology for many of the advances made in the last half century due to its low cost, high yield in manufacturing, and ability to connect increasingly complex integrated circuits (IC) and components. With increasing power requirements in many applications, modeling and accounting for the dissipation of heat generated due to losses on a PCB can be difficult to model without the use of expensive and complicated software [4].

Standards such as IPC 2152 have been developed for sizing individual traces along a PCB and design aides have been developed to simulate these results [5]. While these solutions are a great start for a PCB design, they don't take into account the impact to thermal performance of neighboring features of the design. More specific conditions can be analysed using thermal networks to model single areas of a PCB design [6, 7]. These solutions are often so specific to a problem that they can require significant rework to perform analysis of new designs. Other software solutions can analyze an entire PCB but often require expensive licenses and extensive knowledge in that software suite. Additionally, this type of analysis is often performed by an engineer specializing in simulation and not the PCB designer. This requires that the design of the PCB and analysis of its performance be performed by different individuals and thus slowing down the entire design process.

The mission of this research is to develop an open-source software solution for analyzing thermal performance of a PCB design in a manner that is simple to use and can perform its analysis relatively quickly. The intent is not to design a fully featured analysis tool that can completely replace the existing software solutions, many of which can perform very detailed and complex simulations but instead to give a designer confidence in their PCB design and give indications of areas of concern that could require revision.

1.0.1 Objective

This research has four main objectives. First, to develop a means of converting common PCB design files (gerber and drill files) into a mesh for finite element analysis (FEA). Second, to find conduction losses in traces based on user inputted electric loads. Third, complete a steady state heat transfer analysis of the PCB design. Finally, to present the input and output of this analysis in a graphic user interface (GUI).

1.0.2 Organization

This thesis contains subsequent chapters as follows: Chapter 2 will discuss current methods of thermal analysis of PCBs and several of the limitations many designers face in analyzing a full PCB. The methods and design used to develop a computer application to perform the thermal analysis are found in chapter 3. Chapter 4 contains simulation results and correlation to physical testing of a test PCB, as well as correlation to the common design standard IPC 2152.

CHAPTER 2
LITERATURE REVIEW

2.1 PCB Construction

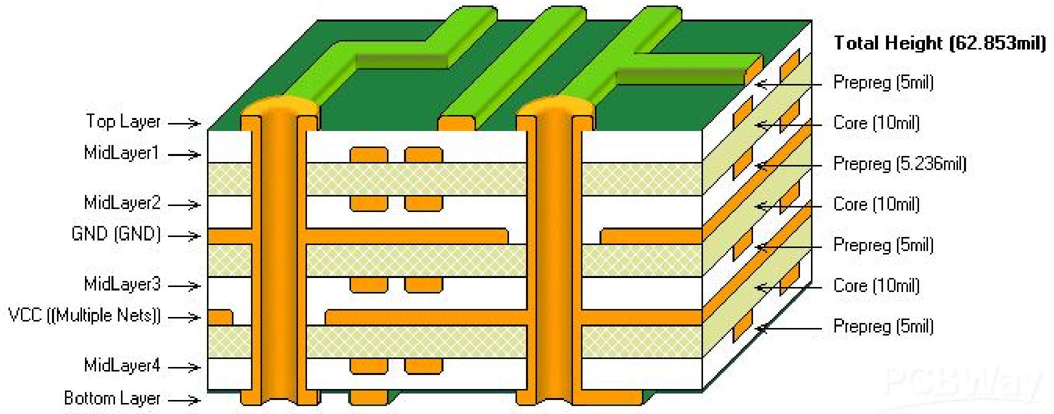


Fig. 2.1: Example PCB of 8 conductive layers [3].

While the construction of printed circuit boards varies from board to board and manufacturer to manufacturer, there are some general methods which are used nearly universally. The function of a PCB is to connect circuits from different components and connectors through the use of conductive traces. These traces are typically made of annealed copper but can be made from a number of different electrically conductive materials such as aluminum. The area of the PCB that is not part of the conductive trace is a dielectric material such as FR-4. This dielectric material is used primarily in two different layers of the PCB: the core layer in which the raw conductor sheets are fixed to, and the prepreg layer which is used to insulate one cut layer of conductor to another. While these two layers are similar in material properties, they vary significantly in thickness.

Figure 2.1 illustrates how an eight layered PCB may be constructed. In this example, three sheets of core layer with copper bonded on either side first have the copper layers cut to match the appropriate trace pattern for each layer. The three sheets are then bonded together using prepreg to insulate the intermediary copper layers. Holes are then drilled and the entire PCB is plated where appropriate.

The PCB has several different materials that are used in its construction. These include aluminum, copper, epoxy, FR-4, nickel, and several others. Finding appropriate material properties is crucial for modeling the thermal performance of the PCB. Since these materials are commonly used, several sources are available for determining appropriate values.

2.2 Heat Transfer

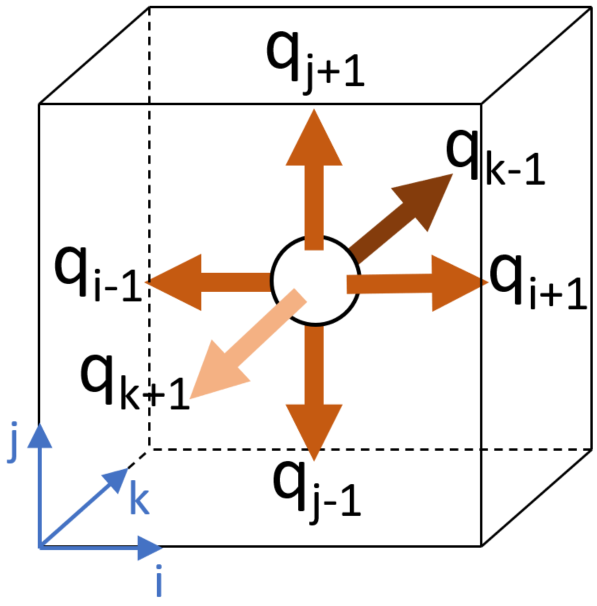


Fig. 2.2: Heat balance for a control volume.

2.2.1 Heat Equation - Steady State

The most basic form of heat transfer for a given volume is based on the principle of energy balance. For a steady state solution, this looks as simple as the heat generated inside a volume is equal to the heat transferred out as seen in figure 2.2. For a given control volume, this is described by equations 2.1 and 2.2 [1].

$$q_{out} = q_{gen} \tag{2.1}$$

$$q_{i-1} + q_{i+1} + q_{j-1} + q_{j+1} + q_{k-1} + q_{k+1} = q_{gen} \tag{2.2}$$

The heat transferred through the Cartesian directions as indicated in figure 2.2 can be characterized as thermal resistances which in turn form a thermal resistor network[1]. This is common practice in most analytical heat transfer problems of known material properties and boundary conditions. These thermal resistances will either be through a solid medium (conduction), through a fluid medium (convection) or through no medium (radiation). Conduction can also be modeled between solid materials that interface across a surface.

2.2.2 Conduction

Fourier's law denotes that the 1-D heat transfer through a solid medium can be characterized as conduction and is represented by equation 2.3[2] where q is the heat transfer rate in watts, A is the perpendicular area through which heat is being transferred, L is the linear length in which the transfer is occurring in the medium, and ΔT is the temperature difference of the boundaries of the conduction medium. The constant k denotes the thermal conductivity of the material. This value is determined through physical experimentation and its units are $\frac{W}{in^{\circ}C}$ or $\frac{W}{m^{\circ}C}$ [8].

$$q_{cond} = \frac{kA}{L}\Delta T \tag{2.3}$$

The true thermal conductivity depends on the temperature of the medium but for this research, all values are estimated at 300 K. This approximation is appropriate given that the operational temperature of most PCBs is close to this value. Table A.1 contains the thermal conductivity of common materials used in PCB construction.

[1]This assumes that any heat generated in the volume is from a point source at the center of the volume.
[2]This assumes that neither A or k vary over L.

2.2.3 Convection and Radiation

The heat transfer rate normal to a surface interfacing with a fluid, such as air, can be expressed by equation 2.4 where q is the heat transfer rate in watts, h is the heat transfer coefficient, A is the area of the surface, and ΔT is the temperature difference of the surface temperature and the ambient air temperature [1] [3]. While the form of this equation is straightforward, determining h can be difficult and is explained in the below sections.

$$q_{conv,rad} = hA\Delta T \tag{2.4}$$

Heat Transfer Coefficient - Convection

There are many factors in how a surface exposed to a fluid medium will transfer heat but for the purposes of this analysis, the focus will be on free convection whose primary forcing factor is gravity and the buoyancy force of lower density fluids moving vertically with relation to higher density fluids. There are several dimensional-less parameters which can be used to describe the behavior of convection. Two parameters that are particularly useful for free convection are the Grashof (Gr) number and Prandtl (Pr) number. The Grashof number is described as the ratio between buoyancy and friction forces where the Prandtl number is described as the ratio between the expansion of the laminar flow boundary layer to the expansion of the temperature boundary layer [8]. For ideal gases such as air, this number is consistently close to 0.71 [1]. The Grashof number can be expressed as equation 2.5 where the first term is the ratio of the gravity constant and the kinematic viscosity of the fluid ν, β is the thermal expansion coefficient, ΔT is the temperature difference between the surface temperature and the ambient air temperature, and P is the characteristic length for the surface. For an ideal gas such as air, the thermal expansion coefficient can be expressed as the reciprocal of the ambient air temperature in Kelvin shown in equation 2.6 [1,8].

$$Gr = \left(\frac{g}{\nu^2}\right)\beta\Delta T P^3 \tag{2.5}$$

[3]This assumes that the surface temperature and convective forces are uniform across the surface.

$$\beta = \frac{1}{T_A}\,[K] \tag{2.6}$$

The characteristic length for a surface is dependent on the orientation of the surface to the gravity force direction. For the purposes of this analysis, the orientations which will be examined are kept to vertically and horizontally facing surfaces. Values can be determined according to table 2.1. One final dimensional-less parameter needed is the Rayleigh (Ra) number which is the product of the Grashof and Prandtl numbers. Depending on the value of the Rayleigh number and the orientation of the surface, the parameters C and n can be determined as seen in table 2.1. Finally, the heat transfer coefficient for convection can be calculated using equation 2.7.

Table 2.1: Convection Parameters [1]

Orientation	C	n	Ra	Flow	P $[in]$
Vertical	0.59	1/4	$10^4 < Ra < 10^9$	Laminar	H
	0.13	1/3	$10^9 < Ra < 10^12$	Turbulent	
Horizontal ↑	0.54	1/4	$2.2 \times 10^4 < Ra < 8.0 \times 10^6$	Laminar	$\frac{WL}{2(W+L)}$
	0.15	1/3	$8.0 \times 10^6 < Ra < 1.6 \times 10^9$	Turbulent	
Horizontal ↓	0.27	1/4	$3.0 \times 10^5 < Ra < 3.0 \times 10^{10}$	Laminar	$\frac{WL}{2(W+L)}$

$$h_{conv} = C k_{air} Ra^n / P \tag{2.7}$$

Heat Transfer Coefficient - Radiation

Heat transfer through a surface due to radiation is given as equation 2.8 where ϵ is the emissivity of the surface (typically 0.8 for a painted or epoxy coated surface [1]), σ is the Stefan-Boltmann constant of $5.67 \times 10^8 \left[Wm^{-2}K^{-4}\right]$, A is the area of the surface, T_S is the surface temperature in Kelvin, and T_A is the ambient air temperature in Kelvin [4]. In order to better match the format for thermal resistance of conduction and convection, this expression can be rewritten as equation 2.9 [9]. This makes the heat transfer coefficient for radiation equivalent to equation 2.10. Now that both convective and radiative heat transfers follow the same form, the total heat transfer coefficient for convection and radiation is simply the sum of the two as expressed in equation 2.11.

$$q_{rad} = \epsilon \sigma A \left(T_S^4 - T_A^4\right) \tag{2.8}$$

$$q_{rad} = \epsilon \sigma A \left(T_S^2 + T_A^2\right) \left(T_S + T_A\right) \Delta T \tag{2.9}$$

$$h_{rad} = \epsilon \sigma \left(T_S^2 + T_A^2\right) \left(T_S + T_A\right) \tag{2.10}$$

$$h = h_{conv} + h_{rad} \tag{2.11}$$

2.3 Thermal Resistances and Networks

For the heat transfer between two points where the conditions between those points are consistent, the heat transfer can be expressed as equation 2.12 where q is the heat transfer rate in watts, ΔT is the temperature difference between the two points, and R is the thermal resistance [10]. This resistance is analogous to the electrical resistance in a electric circuit as seen in Ohm's Law. The thermal resistance can be expressed as seen in equation 2.13. It is often useful to instead express its reciprocal, the thermal conductance as seen in equation 2.14.

$$q = \frac{\Delta T}{R} \tag{2.12}$$

[4]This assumes that the temperature and radiative conditions are uniform across the surface and that the surrounding surfaces, which is what T_A truly represents, match the ambient air temperature.

$$R = \frac{\Delta T}{q} \tag{2.13}$$

$$q = C\Delta T \tag{2.14}$$

The same methods of evaluating electrical circuits with multiple impedance elements can be applied to thermal resistances as well. For example, two solid plates places against each other can be expressed as two thermal resistances in series as illustrated in figure 2.3. More complex arrangements can be expressed as series and parallel thermal resistances. For example, the composite thermal resistance of the configuration shown in fig 2.4 would be expressed as R_1 in series with R_2 and R_3 in parallel.

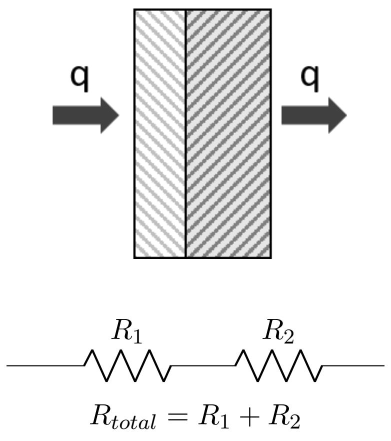
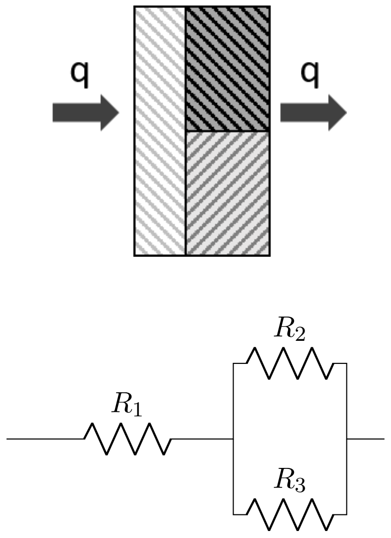


Fig. 2.3: Series thermal resistance.



$$R_{total} = R_1 + R_2||R_3 = R_1 + \frac{R_2 R_3}{R_2+R_3}$$

Fig. 2.4: Series and parallel thermal resistances.

2.3.1 Conduction and Convection/Radiation Thermal Resistances

Applying equation 2.13 to equation 2.3 and equation 2.4, the thermal resistance for conduction given as R and its reciprocal conductance C can be expressed as equations 2.15 and 2.16. Likewise for convection and radiation, the thermal resistance R and conductance C are given by equations 2.17 and 2.18.

$$R_{cond} = \frac{L}{kA} \tag{2.15}$$

$$C_{cond} = \frac{kA}{L} \tag{2.16}$$

$$R_{conv,rad} = \frac{1}{htcA} \tag{2.17}$$

$$C_{conv,rad} = htcA \tag{2.18}$$

2.4 Thermal Analysis of PCB Design

Given that the PCB is constructed in layers of well controlled thickness and material properties, creating a thermal network to evaluate specific portions of a PCB design can be readily accomplished. One area that continues to be examined is the use of vias for increasing the diffusion of heat from a heat source mounted to the PCB [9, 11–13]. Other areas that thermal networks are used in recent papers are modelling a specific component mounted to the PCB [14] and even evaluating a motor stator design [7]. In all of these examinations, the approach is fairly similar.

Take for example a heat source mounted on a two layer PCB with vias used to diffuse the heat as seen in figure 2.5. Assuming the majority of heat transfers vertically and that the temperature of the component is homogeneous, each layer can be interpreted as an equivalent thermal resistance. The resistances can then be placed in a thermal network as seen in 2.6. At this point, several different analyses could be performed depending on the desired piece of information. If its desired to know the component temperature given a heat output and ambient air temperature, equations 2.12, 2.15 and 2.18 could be used in an energy balance as seen in equation 2.1.

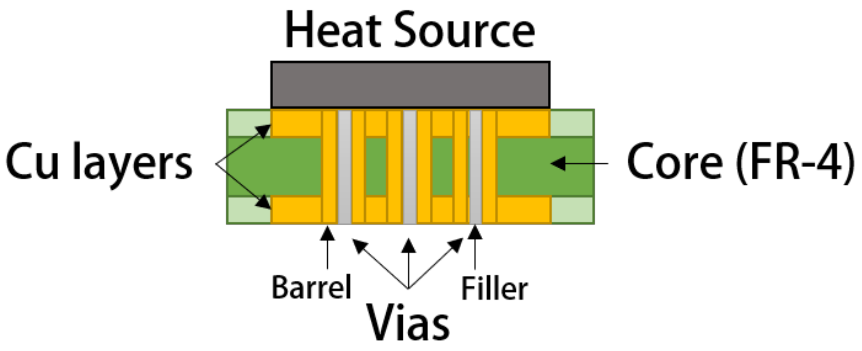


Fig. 2.5: PCB example - heat source and vias for heat diffusion.

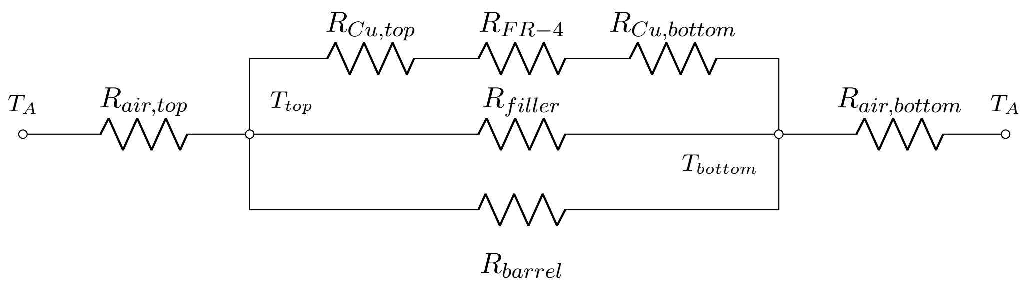


Fig. 2.6: Thermal network of PCB example.

2.5 Limitations to Analytical Solutions

The approach mentioned above is useful for the design of specific design features and should be used for generating the initial parameters around that feature. However, it does not take into account what neighboring features of a PCB design might contribute to its performance. In the example above, if multiple of these components were placed close together, their thermal performance could be considerably worse, especially for components that have less isolation in their placement.

This gives rise to the need of more holistic analyses. While it may make sense to simply expand a given thermal network analysis to include more neighboring details, this can quickly become a herculean task given the number of features on a PCB and their proximity to each other. This is where computer algorithms are best utilized to perform more laborious calculations. The remainder of this research explores the design and means to develop such an algorithm and computer application.

CHAPTER 3
RESEARCH AND DESIGN METHODS

3.1 Conversion to Elements

To perform a finite element analysis on a PCB design, the board design needs to first be interpreted into a mesh of elements. To perform this action, the different draw features of a gerber file need to be understood. While the design of a PCB can be quite complex, the draw callouts that define the design are made from simple shapes and will be referred to as features. It should be noted that describing complex geometry with simple shapes and directions will mean that even a relatively simple PCB design can have a gerber file with hundreds of lines of draw callouts and a complex PCB will have thousands of lines of draw callouts! While each line is not difficult to understand, the volume of lines necessitates the use of a computer algorithm to interpret a gerber file into a geometry. This holds true for both the manufacture of the board as well as this research which will interpret the design into a mesh of elements used for analysis.

3.1.1 Gerber Basic Format

The gerber file is comprised of three sections. The first section is used to determine the precision of all dimensions in the file and what units those dimensions will be. The second section contains the feature callouts which defines all standard features (circle, rectangle, oval) that will be used. Each feature definition start with $\%AD$ followed by the feature ID (starts with 10 and increments by one for each feature), the feature type, and then any feature dimensions. For example, a circle of diameter 10 mil would have a feature definition of $\%ADD10C, 0.01000 * \%$. If another circle feature with a diameter of 5 mil were to be used in the same gerber file, its feature definition might look like $\%ADD11C, 0.00500 * \%$. Figure 3.1 shows an example gerber file and demonstrates how each line contains a new instruction. Comments for the significance of each instruction can be found on the right-hand side denoted with $***$.

The third section contains the draw callouts. If there are polygon pours in the layer file, they will typically be immediately after the feature callouts and is explained in more detail in section 3.1.4. Following the polygon pours, the standard features will be implemented. This is initiated by a line containing one of the feature IDs. For example, if the circle example above were to begin being drawn, a new line with $D10*$ would denote the start of this feature's draw callouts. The next lines will contain the draw callout locations and will place that feature's geometry according to the X and Y coordinates indicated. Since many features are repeated along a single dimension, each line will use the previously specified X or Y coordinate if no coordinate is used in that line. That even holds true from one feature to another. The end characters of a draw callout will indicate whether the draw callout is a new instance of that feature or a continuation of that feature from the previously given coordinates. This is explained in more detail in the section 3.1.2.

3.1.2 Circle and Line

The most basic feature in a gerber file is the circle. This is denoted with a C in the feature callout after the feature ID and only requires a diameter dimension. If the feature is to be applied as a single point, it will have the characters $D02*$ at the end of the draw callout. An example of a circle feature callout would be $\%ADD10C, 0.01000 * \%$ and an example of it's draw callout would be $X90000Y70000D02*$.

The line feature is identical to the circle feature in the feature callout with the only difference being how is it used in the draw callout: the end characters of $D01*$ will denote a continuation from the previous draw callout. This will create a string of continuous circles from a start point to an end point; the start point being the previous draw callout's X and Y coordinates and the end point being the new draw callout's coordinates. For example, if a line of the above circle example feature were to be be drawn from $(900, 700)$ to $(1200, 700)$, the first line would be identical to how the circle was called out: $X90000Y70000D02*$ and the subsequent line would be $X120000D01*$.

```
G04*
G04 @! TF.GenerationSoftware,Altium Limited,Altium Designer,21.5.1 (32)*
G04*
G04 Layer_Color=16711935*
%FSLAX25Y25*%          *** Precision of all numbers used
%MOIN*%                *** Units used MOIN=in, MOMM = mm
G70*
G04*
G04 @! TF.SameCoordinates,B9470C18-1CFA-493D-80FA-71023878962F*
G04*
G04*
G04 @! TF.FilePolarity,Positive*
G04*
G01*
G75*
%ADD10C,0.01000*%      *** All AD* call out a new feature of a certain size
D10*                   *** starts location(s) for feature D10
X90000Y70000D02*       *** D02 is starting point, starts at X= 900 mil, Y=700 mil
X120000D01*            *** D01 continues from last point, X= 1200 mil
Y0D02*                 *** D02 starts new location, Y = 0 [x still =1200]
Y70000D01*             *** D01 continues from last point, Y = 700 [x still = 1200]
X0Y0D02*               *** D02 starts new location, x = 0, y = 0
X120000D01*            *** D01 continues from last point, x = 1200
X0D02*                 *** D02 starts new location, x = 0, [y still = 0]
Y70000D01*             *** D01 continues from last point, Y = 700 [x still = 0]
M02*
```

Fig. 3.1: Example Gerber File

3.1.3 Rectangle and Oval

The next most basic feature in a gerber file is the rectangle. This is denoted with a R in the feature callout after the feature ID and requires a width and height dimension. Typically, this feature is not drawn as lines but at single points. An example of a rectangle feature callout would be $\%ADD11R, 0.05000X0.02800 * \%$ which would be a $50X28$ mil rectangle.

The oval feature is very similar to the rectangle feature and is denoted with an O in the feature callout after the feature ID. This feature also requires a width and height dimension. The difference between a rectangle and oval is that for an oval, the four corners are rounded such that two side are half-circles. It is not specified which sides are rounded but instead inferred to be the shorter of the two sides. An example of an oval feature callout would be $\%ADD11O, 0.05000X0.02800 * \%$ which would be a $50X28$ mil oval with the left and right sides being two half circles - inferred from the the smaller dimension.

3.1.4 Polygon Pour

The polygon pour is a polygon defined by a series of points with the volume inside those points as being the drawn area. This feature is first identified by the line $G36*$ in the draw callout section. Subsequent lines are used to identify the control points of the polygon and give the X and Y coordinates of those control points. The feature is finished by the line $G37*$. If multiple polygon pours are used, each one will start with $G36*$ and end with $G37*$.

3.1.5 Drill File

The drill file is very similar to a gerber file with some minor differences in formatting. Figure 3.2 demonstrates a drill file with each line of instruction. Again, comments for the significance of each line can be found on the right hand side denoted with $***$.

```
M48
;Layer_Color=9474304
;FILE_FORMAT=2:4          *** Precision of all numbers
INCH,LZ                   *** Units used (inch in this case)
;TYPE=PLATED              *** Indicates hole type (plated holes is this case)
T1F00S00C0.0276           *** Call out for first hole tool (circle, diameter 0.0276 in)
T2F00S00C0.0402           *** Call out for second hole tool (circle, diameter 0.0402 in)
T3F00S00C0.0472           *** Call out for third hole tool (circle, diameter 0.0472 in)
%                         *** indicates tool callout complete
T01                       *** switch to tool 'T1'
X054744Y08285             *** Place hole at location (X=5474.4mil , Y= 8285.0 mil)
X055532                   *** Place hole at location (X=5553.2, Y= 8285.0 still)
X056319                   *** Place hole at location (X=5631.9, Y= 8285.0 still)
X057106                   *** Place hole at location (X=5710.6, Y= 8285.0 still)
Y083638                   *** Place hole at location (X=5710.6 still, Y= 8363.8)
X056319                   *** Place hole at location (X=5631.9, Y= 8363.8 still)
```

Fig. 3.2: Example Drill File

3.2 Mesh Conversion

To perform the heat transfer analysis using the finite element method, the gerber files used to describe a PCB need to be interpreted into a mesh. This is performed by reading each set of instructions of the gerber file and tracing over a grid of a set resolution and denoting the traced cells in the grid as the conductor material. It is implied that the cells that are not traced are the dielectric material for that layer in the PCB. This method is shown in figure 3.3 where the shaded grey area is the true feature dimensions and the cells in which center lies within the feature dimension are designated as the conductor material – marked by Xs in the figure [15]. This action is performed for each conductor layer within the PCB. The drill file can then be read and holes applied where indicated. In figure 3.3, this is shown by the orange circle demonstrating the drill feature callout's true dimensions and the red Xs denoting the cells that are removed from the hole. If the holes are plated, the indicated plating thickness and material can be applied to the outside of any holes created. Figure 3.4 shows the results of this method in interpreting a single layer of a PCB into a mesh. The specified layer thickness is used to set the thickness of the elements for that layer based on the 2D map of cells. This research will refer to cells as 2D squares specific to a layer and elements as 3D rectangular prisms.

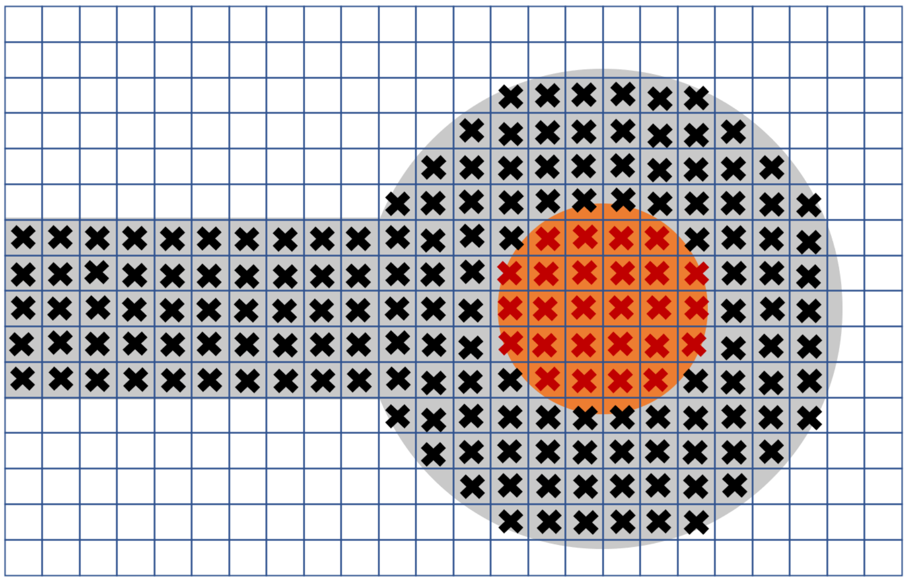


Fig. 3.3: Gerber file drawing features interpreted into an element mesh.

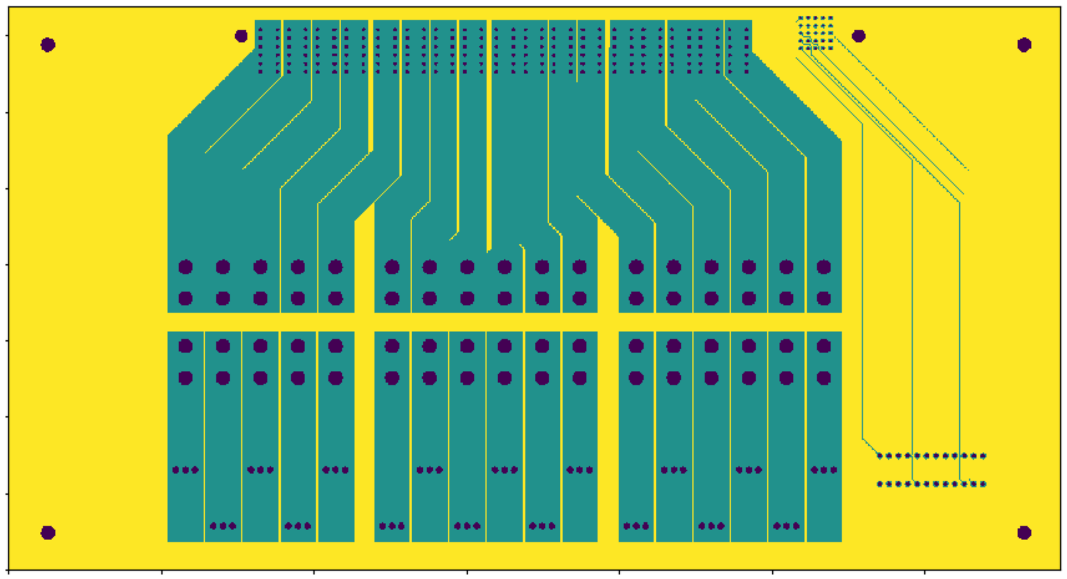


Fig. 3.4: A gerber file interpreted into a mesh. Yellow denote the dielectric material, teal denotes the conductor material or traces, and purple denotes the holes created from the drill file.

3.3 Calculation of Conduction Losses

The conduction loss of a conductor under DC load is given by equation 3.1 where q_{loss} is the heat generated, I is the current passing through the conductor and R is the electrical resistance of the portion of the trace that current flows [11]. The electrical resistance of a trace can be found using equation 3.2 where ρ[1] is the resistivity of the material, A is the cross-sectional area and L is the length through which the current passes. The cross-sectional area A can be broken down into the width and height of the trace. Where the height of the trace is simply the layer height, the width is more difficult to determine from the traced PCB files. To determine the width, the use of path finding algorithms are used.

$$q_{loss} = I^2 R \tag{3.1}$$

$$R = \frac{\rho}{A} l = \frac{\rho l}{wh} \tag{3.2}$$

3.3.1 Identification of Networks

Before a path finding algorithm can be used, the allowed material for a path needs to be determined. This is equivalent to finding the electrical networks of a PCB layer and is determined by looking at surrounding cells. If a surrounding cell of a conductor element is also a conductor, they are part of the same network. This is repeated for all conductor elements in the mesh. Figure 3.5 shows the result of this method as the different tones denote different networks along a layer. This research is limited to finding networks in a single layer and not across layers as can be accomplished by the use of vias or across components. Additional algorithms could be developed to identify cross layer networks.

[1] Common values for electrical resistivity can be found in table A.2.

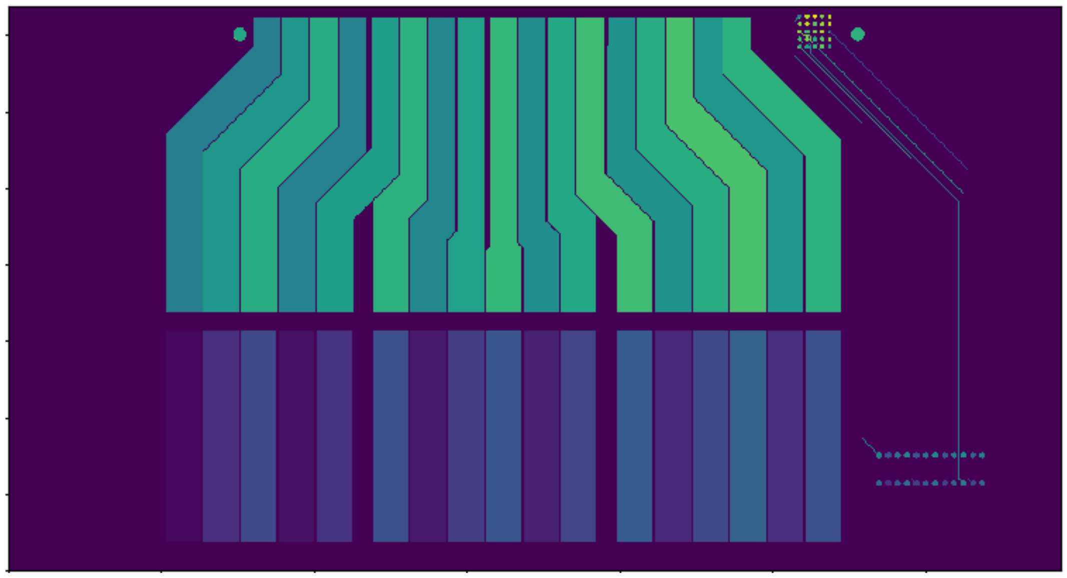


Fig. 3.5: Board conductor separated into networks. Each color tone denotes a difference network along that layer.

3.3.2 Path Finding

Current will travel from a higher electrical potential to a lower one along the path of least resistance. For a PCB trace design, this typically means the shortest path possible. In figure 3.6, a simple arch shaped trace can be seen by the individual cell blocks. An S denotes the start of that current path and an F denotes the finish of the path. The cells with a red square show the ideal path that the current would flow through the trace.

Through the development of artificial intelligence applications, many algorithms have already been developed to find a path through a 2-D matrix of allowed path cells (conductor) and not allowed cells (dielectric) as defined in section 3.3.1. One of the most prominent algorithms is the A* method. The A* method works by defining a start condition, a desired end condition and finding a series of steps to get from the start to the end condition that results in the lowest total step "cost". In a path finding case, the starting condition would be the start location, the end condition the end location and the cost would be the distance traveled as defined by each individual step required. This method is very powerful in finding paths requiring diversion around many obstacles along the way.

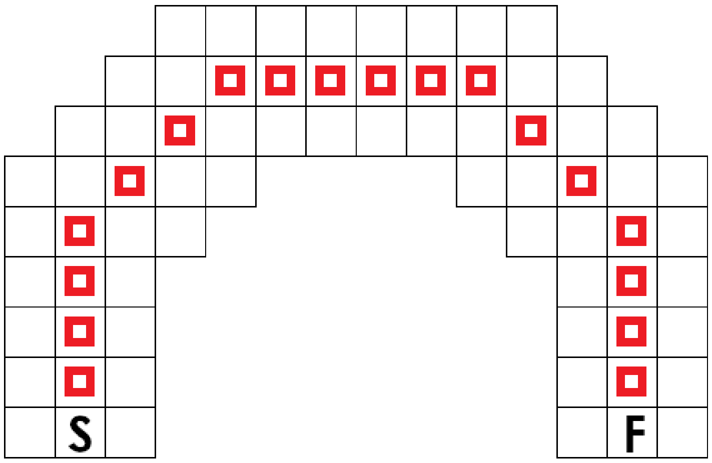


Fig. 3.6: Simple path finding example. Cells indicate allowed area for path. *S* and *F* denote the start and finish of path. The red squares show the path through the area.

In the case of this research, a premade library with the A* method was used with its output being a list of each position along the path. From each position, individual steps along the path are found by calculated the difference between each position. For each step calculated, an orthogonal direction can be found by swapping the X and Y components and inverting one of them. For this algorithm, inverting the Y component after the swap was chosen. Table 3.1 shows the step directions and vectors along with their respective orthogonal directions and vectors.

Table 3.1: Step Direction and Orthogonal Direction

Step Direction	Step Vector $[X,Y]$	Orthogonal Direction	Orthogonal Vector $[X,Y]$
$\leftarrow$	[-1, 0]	$\uparrow$	[0, 1]
$\nwarrow$	[-1, -1]	$\nearrow$	[1, 1]
$\uparrow$	[0, 1]	$\rightarrow$	[1, 0]
$\nearrow$	[1, 1]	$\searrow$	[1, -1]
$\rightarrow$	[1, 0]	$\downarrow$	[0, -1]
$\searrow$	[1, -1]	$\swarrow$	[-1, -1]
$\downarrow$	[0, -1]	$\leftarrow$	[-1, 0]
$\swarrow$	[-1, -1]	$\nwarrow$	[-1, 1]

With the basics of how path finding operates, the process of finding the trace width can begin. To find the width at each step, an algorithm can march in the negative orthogonal direction until it meets the edge of conductor material, then march in the positive orthogonal direction until it meets the opposite edge of material. Counting the number of cells marched from one edge to the other will give the width of the trace at that point. This action is repeated for each position along the path.

The resistance along that width can then be found using equation 3.3 where n is trace width in number of cells. It should be noted that cells that have uniform length and width, those terms cancel out to give the formula expressed in equation 3.4. Further, since the resistance is distributed along the path, the individual cell resistance can be found in equation 3.5. To find the heat generated due to conduction loss, equation 3.5 is then used in equation 3.1. This is applied for all cells that are identified as having conduction loss given a load.

$$R_{width} = \frac{\rho l}{(nw)\,h} \tag{3.3}$$

$$R_{width} = \frac{\rho}{nh} \tag{3.4}$$

$$R_{cell} = \frac{\rho}{hn^2} \tag{3.5}$$

Following each step of the path and finding its width and then respective resistance, the resistance of each cell in that trace can be determined. This method assumes that the total trace resistance can be modeled as a series of individual trace widths sliced along each step of the path. Figure 3.7 depicts the path found (red squares) and the width at each step of the path (orange lines). Using this approach leaves some cells that do not have a resistance found because they are not part of a step's width (denoted by blue dots). For simplification, these cells can have their resistance match the closest neighboring cell with a resistance value.

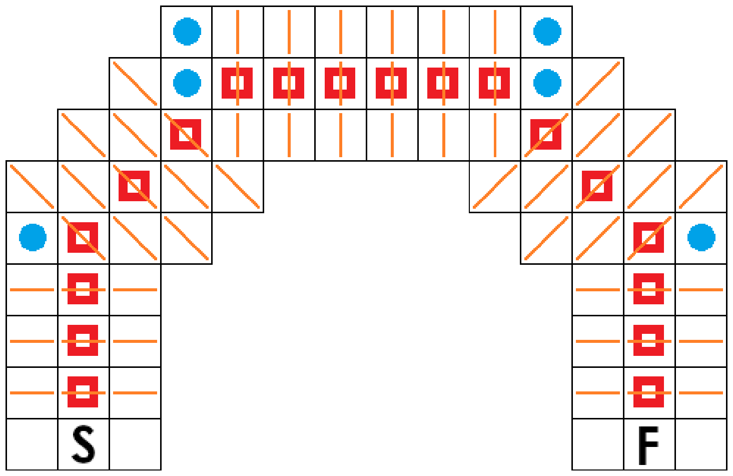


Fig. 3.7: Detailed Pathfinding example. Cells indicate allowed area for path. *S* and *F* denote the start and finish of path. The red squares show the path through the area. Orange lines indicate the orthogonal step direction and width at that step. Blue dots indicate cells not implicitly calculated.

3.3.3 Path Finding Issues

Using the path finding algorithm without any modification will find a path as found in figure 3.8. There are several issues with this including the possibility of more cells not having a cell resistance implicitly calculated (the more that are implicitly calculated the better), and erroneous trace width calculations along starting and finishing points due to the path's tendency to the inside of the path area.

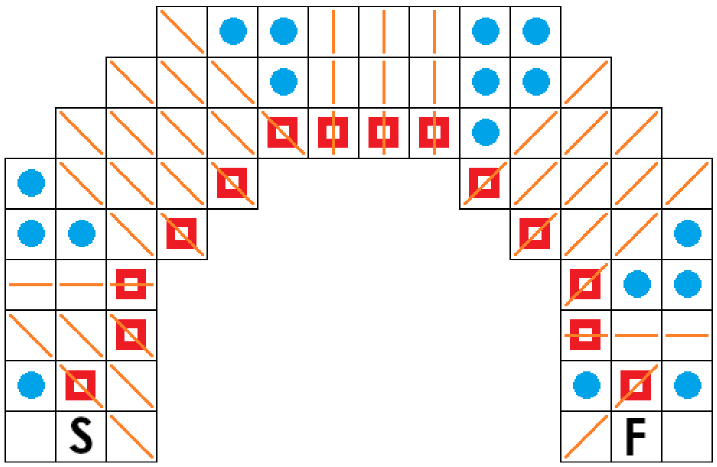


Fig. 3.8: Path finding and resistance algorithm without modifications.

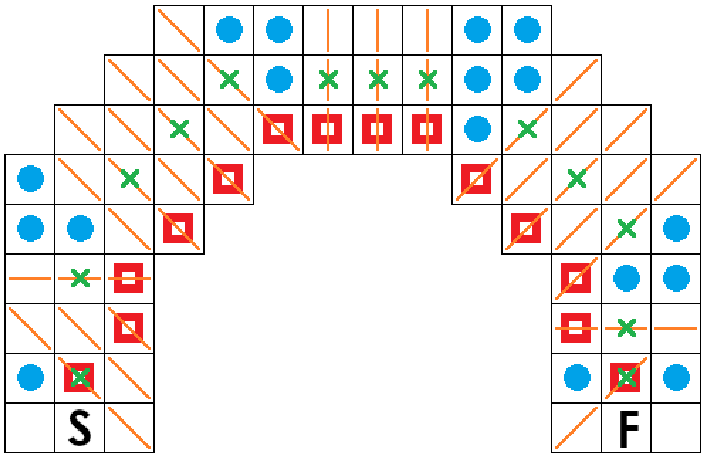


Fig. 3.9: Path finding corrected using center width locations as way-points.

To help the path stay closer to the center of the trace, an additional centering algorithm can be employed. To do this, the first path as depicted in figure 3.9 can be found (red squares), then the center cell along that width can be found (green X). This center is found using a similar method for finding the width: marching in the negative orthogonal direction until the edge of the conductor is met, and marching back in the positive orthogonal direction half the distance of the width. Once the center cell is found, it is used as a way-point for a new path to be found. This method is repeated for every step along the original path with the center cells compiled into a list of way-points. Once all the way-points are found, the center path is found by finding smaller paths between way-points. This results in a path (and cell resistances) that closely match the intended path seen in figure 3.6. Even with this corrected path finding method, some granularity will exist in the trace for resistances. To help correct this effect, a smoothing function can be applied to average out the resistance value of the neighboring cells within some radius.

3.3.4 Conductor Material Not in Conduction Loss Path

If the above algorithm is used to set the resistances of the cells along a current path, there are some cells which do not belong to a cell's current path but are part of the conductor material along that trace. While these cells act as a good thermal conductor to help wick heat away from where it is being generated, they should not be considered to generate heat due to conduction losses. Therefore, these extra cells need to be trimmed from the resistance values found as to not erroneously add more heat than is actually being generate. These trimmed cells are shown in figure 3.10 denoted by a grey triangle.

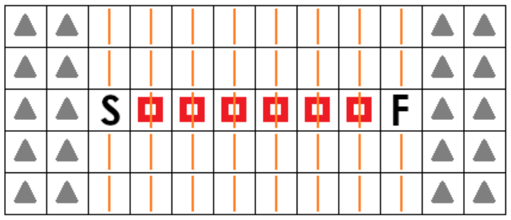


Fig. 3.10: Extra conductor material not part of the current path.

3.4 Heat Transfer Solution

Following the equations from section 2.2, each element in the grid of the PCB mesh will have a heat equation consisting of its neighboring temperatures and conductances as seen in equation 3.6. This equation can be rewritten as equations 3.7 and 3.8[2].

[2]This assumes that the heat generated is a single point source at the center of the element.

$$\begin{aligned} & C_{i-1}\left(T_{i,j,k}-T_{i-1,j,k}\right)+C_{i+1}\left(T_{i,j,k}-T_{i+1,j,k}\right) \\ +& C_{j-1}\left(T_{i,j,k}-T_{i,j-1,k}\right)+C_{j+1}\left(T_{i,j,k}-T_{1,j+1,k}\right) \\ +& C_{k-1}\left(T_{i,j,k}-T_{i,j,k-1}\right)+C_{k+1}\left(T_{i,j,k}-T_{i,j,k+1}\right)=q_{gen} \end{aligned} \tag{3.6}$$

$$\begin{aligned} C_{i,j,k}T_{i,j,k} & \\ -C_{i-1}T_{i-1,j,k}-C_{i+1}T_{i+1,j,k} & \\ -C_{j-1}T_{i,j-1,k}-C_{j+1}T_{i,j+1,k} & \\ -C_{k-1}T_{i,j,k-1}-C_{k+1}T_{i,j,k+1} & \\ & =q_{gen} \end{aligned} \tag{3.7}$$

$$C_{i,j,k}T_{i,j,k}=C_{i-1}+C_{i+1}+C_{j-1}+C_{j+1}+C_{k-1}+C_{k+1} \tag{3.8}$$

$$\mathbf{G}\vec{T}=\vec{S} \tag{3.9}$$

$$\vec{T}=\mathbf{G}^{-1}\vec{S} \tag{3.10}$$

Using this formulation of the heat equation, the full series of heat equations for n elements can be formulated into equation 3.9. **G** (equation 3.11) is the tensor matrix describing the interaction between neighboring elements and is found through the calculation of each conductance value. $\vec{T}$ (equation 3.13) is the list of the temperature at each element and is the unknown in this equation. $\vec{S}$ (equation 3.14) is the remaining known quantities in each heat equation such as heat generated at each cell and the ambient air's contribution to heat transfer through convection and radiation. Once **G** and $\vec{S}$ and determined, $\vec{T}$ can be calculated using equation 3.10.

$$\mathbf{G} = \begin{matrix} \begin{bmatrix} C_1 & -C_{1\to i+1} & 0 & 0 & 0 & \dots \\ -C_{2\to i-1} & C_2 & -C_{2\to i+1} & 0 & 0 & \dots \\ 0 & -C_{3\to i-1} & C_3 & -C_{3\to i+1} & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \\ \begin{bmatrix} 0 & -C_{1\to j+1} & 0 & 0 & 0 & \dots \\ 0 & 0 & -C_{2\to j+1} & 0 & 0 & \dots \\ 0 & 0 & 0 & -C_{3\to j+1} & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \\ \begin{bmatrix} 0 & -C_{1\to k+1} & 0 & 0 & 0 & \dots \\ 0 & 0 & -C_{2\to k+1} & 0 & 0 & \dots \\ 0 & 0 & 0 & -C_{3\to k+1} & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \\ \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \iddots \\ 0 & -C_{n-2\to k-1} & 0 & 0 & 0 & \dots \\ 0 & 0 & -C_{n-1\to k-1} & 0 & 0 & \dots \\ 0 & 0 & 0 & -C_{n\to k-1} & 0 & \dots \end{bmatrix} \\ \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \iddots \\ 0 & -C_{n-2\to j-1} & 0 & 0 & 0 & \dots \\ 0 & 0 & -C_{n-1\to j-1} & 0 & 0 & \dots \\ 0 & 0 & 0 & -C_{n\to j-1} & 0 & \dots \end{bmatrix} \\ \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & -C_{n-2\to i-1} & C_{n-2} & -C_{n-2\to i+1} & 0 \\ 0 & 0 & -C_{n-1\to i-1} & C_{n-1} & -C_{n-1\to i+1} \\ 0 & 0 & 0 & -C_{n\to i-1} & C_n \end{bmatrix} \end{matrix} \tag{3.11}$$

$$C_X = C_{X\to i-1} + C_{X\to i+1} + C_{X\to j-1} + C_{X\to j+1} + C_{X\to k-1} + C_{X\to k+1} \tag{3.12}$$

$$\vec{T} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_{n-1} \\ T_n \end{bmatrix} \tag{3.13}$$

$$\vec{S} = \begin{bmatrix} q_{gen_1} + \sum C_{1\to air} T_A \\ q_{gen_2} + \sum C_{2\to air} T_A \\ q_{gen_3} + \sum C_{3\to air} T_A \\ \vdots \\ q_{gen_{n-1}} + \sum C_{n-1\to air} T_A \\ q_{gen_n} + \sum C_{n\to air} T_A \end{bmatrix} \tag{3.14}$$

3.4.1 Matrix and Element Formulation

In order to develop an algorithm to form the above matrices, the elements must be arranged in a consistent manner. The manner chosen for this research is to order each cell first in the X or i direction, then along the Y or j direction and finally along the Z or k direction. This is shown in figure 3.11 where a 4x3x2 mesh is devolved into a series of 24 elements. It should be noted that the total number of elements will determine the size of **G** (24 x 24 in this case) as well as $\vec{T}$ and $\vec{S}$ (24 x 1 in this case). It is easy to see how a PCB made of thousands of elements will then require very large matrices to perform all the heat transfer between the elements.

With the elements defined and correlation between rows and columns of **G**, each row of **G** can then be found. Each row represents that element number's interaction with all other

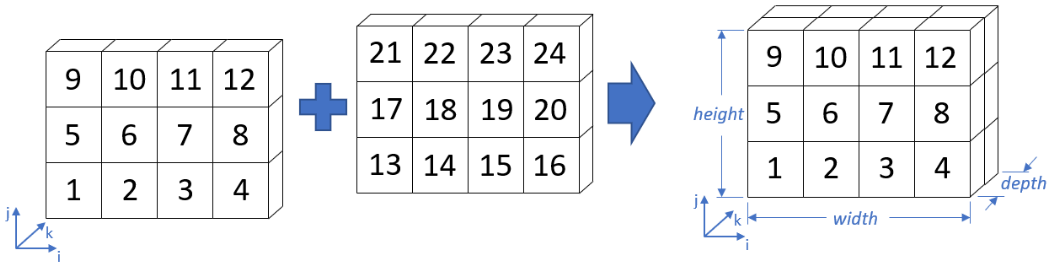


Fig. 3.11: Mesh devolved into elements.

elements. For example, given the mesh in figure 3.11, element 5 interfaces with elements 1 (-j direction), 6 (+i direction), 9 (+j direction), 17 (+k direction), and with the ambient air (-i and -k directions). Assuming no heat is generated in element 5, this would result in a heat equation seen in 3.16 which can be rewritten as equation 3.17. Equation 3.18 shows what this row in **G** would look like, as well as correlating value in $\vec{S}$ (5^{th} row) shown in equation 3.19.

$$C_5 = C_{1\to5} + C_{5\to6} + C_{5\to9} + C_{5\to17} + C_{5\to air_i} + C_{5\to air_k} \tag{3.15}$$

$$\begin{aligned} &C_5T_5 \\ -C_{5\to air_i}T_{air} &- C_{5\to6}T_6 \\ -C_{1\to5}T_1 &- C_{5\to9}T_9 \\ -C_{5\to air_k}T_{air} &- C_{5\to17}T_{17} = 0 \end{aligned} \tag{3.16}$$

$$-C_{1\to5}T_1 + C_5T_5 - C_{5\to6}T_6 - C_{5\to9}T_9 - C_{5\to17}T_{17} = (C_{5\to air_i} + C_{5\to air_k})\,T_{air} \tag{3.17}$$

$$\mathbf{G}_{(5,:)} = [\,\text{-C}_{1\to5}\ \ 0\ 0\ 0\ \text{C}_5\ \ \text{-C}_{5\to6}\ \ 0\ 0\ \text{-C}_{5\to9}\ \ 0\ \ldots\ 0\ \text{-C}_{5\to17}\ \ 0\ \ldots\ 0\,] \tag{3.18}$$

$$\vec{S}_5 = (C_{5\rightarrow air_i} + C_{5\rightarrow air_k})\,T_{air} \tag{3.19}$$

Translating element numbers to coordinates in the 3D mesh is not immediately straight forward. If the formulation of the elements follows figure 3.11, the location of neighboring elements from the 3D mesh into the 2D matrix can be described as in table 3.2. It should be noted that if an element neighbors ambient air, there is no correlating row in **G** for that temperature but is instead captured in $\vec{S}$.

Table 3.2: **G** Row Formulation

Neighboring Mesh Direction	**G** Column $(n^{th} row)$
$i-1$	$n-1$
$i+1$	$n+1$
$j-1$	$n-width$
$j+1$	$n+width$
$k-1$	$n-(width \times height)$
$k+1$	$n+(width \times height)$

3.4.2 Element-to-Element Conductance

Once the PCB design files have been converted into a mesh of elements, the interaction between elements needs to be defined and calculated. The first action is to model each element as a point mass at the center of the element as seen in fig 3.12. Each element is then linked to its neighboring elements through a rod representing the thermal relationship between them as seen in fig 3.13. Each thermal conductance is calculated using equations 2.16 and 2.18. The area will be expressed as the full perpendicular area between elements but the length will be half the length of the element. For example, as a derivation of equation 2.16, the thermal conductance between one element and a neighboring element in the i direction would be expressed as equation 3.21.

Since the link between any two elements is a series of thermal conductances, the total thermal conductance between one element and another is calculated using equation 3.20. This process is repeated for every element in the mesh and evaluated in every Cartesian direction from each element. It should be noted that if one of the thermal conductances is much less than the other, it will dominate the total thermal conductance such that the total thermal conductance could be expressed as simply the smaller value. This is the case with one thermal conductance by conduction and the other by convection and radiation. Even in a dielectric material such as FR-4, the thermal conductance due to conduction is much greater than the thermal conductance by convection and radiation so the total thermal conductance is roughly equivalent to that of convection and radiation.

$$C_{total} = \left[\frac{1}{C_1} + \frac{1}{C_2}\right]^{-1} = \frac{C_1 C_2}{C_1 + C_2} \tag{3.20}$$

$$C_{cond_i} = k_{material} \frac{(element\,height \times element\,length)}{element\,width/2} \tag{3.21}$$

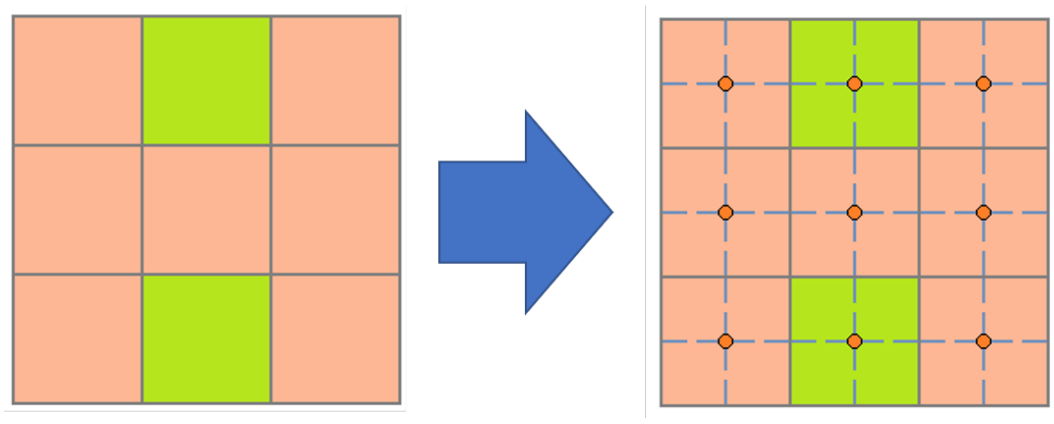


Fig. 3.12: Element equivalent model.

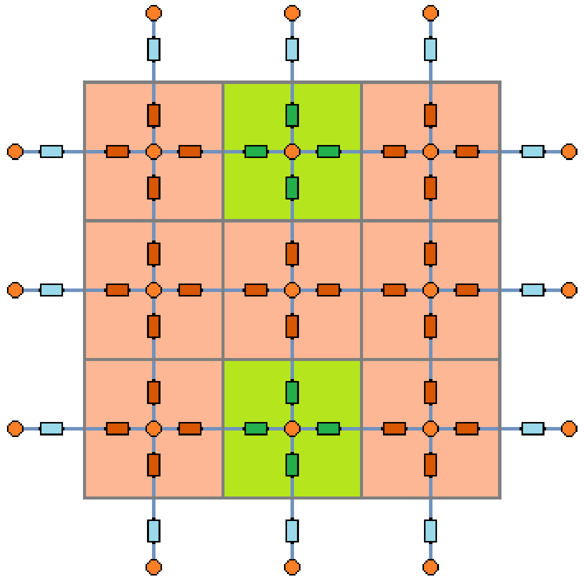


Fig. 3.13: Element-to-element links.

3.4.3 Sparse Matrix Calculations

Representing a full PCB design in a mesh typically requires a very large numbers of elements. If **G** were to be fully defined, the amount of memory used by a personal computer can easily exceed its hardware limitations. This is unfortunate given that most of the matrix has values of zero. This is a fairly common problem with data analytics and thus is called a sparse matrix given that most of the matrix have values of zero. In **G**, any given row has at a maximum of 6 nonzero values and frequently less. Given the commonality of the problem, many existing libraries have been developed to handle extremely large sparse matrices. For this research a pre-existing library was used to handle **G** as a sparse matrix and for the calculation of $\vec{T}$ from equation 3.10.

3.4.4 Mapping Temperature Back To a Mesh

Once the temperature of each element has been found by performing the calculation described in equation 3.10, it then needs to be remapped into the mesh representing the geometry of the PCB design. Again, several existing libraries exist for remapping and resizing matrices to other shapes and were used for this research.

3.5 Graphic User Interface

To develop a graphic user interface (GUI), the user experience needed to be defined. Figure 3.14 illustrates the process the user would carry out to perform the thermal analysis of the PCB. The process is divided into 4 major sections: setup board properties, setup simulation properties, run simulation and show results.

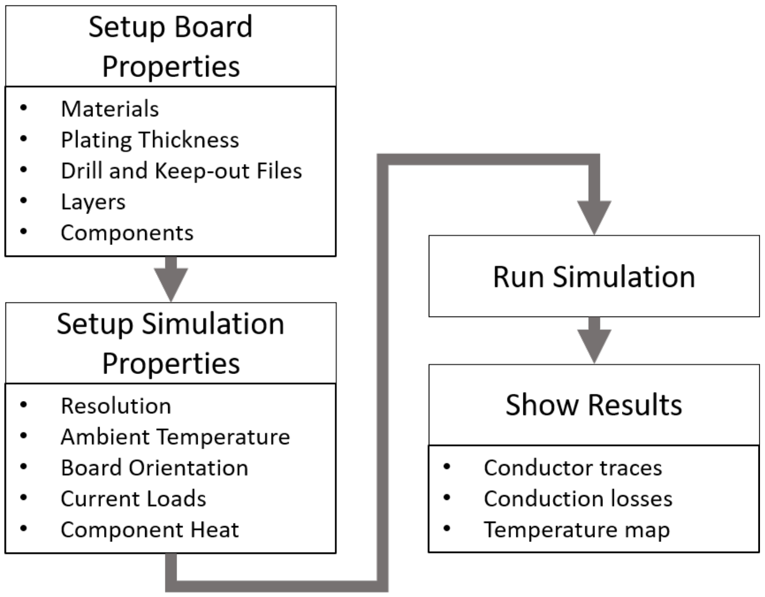


Fig. 3.14: User experience process.

3.5.1 Board Settings

To setup the board properties, first, a PCB board needs to be setup complete with details on the materials used for the conductor and dielectric layers. Second, the plating thickness needs to be specified. Third, the drill and keep-out files need to be identified as these will apply to all layers. These first three settings can be seen in figure 3.15. Fourth, each layer in the PCB design needs to be defined by its type (conductor or dielectric), gerber file and thickness and can be seen implemented in figure 3.16. Fifth, any components that need be examined for heat generation need to be added including their width and length[3] as well as (X, Y) position on the PCB [4]. The board component settings can be seen

[3]Assumed rectangular profile.
[4]This can be determined from pick-and-place file.

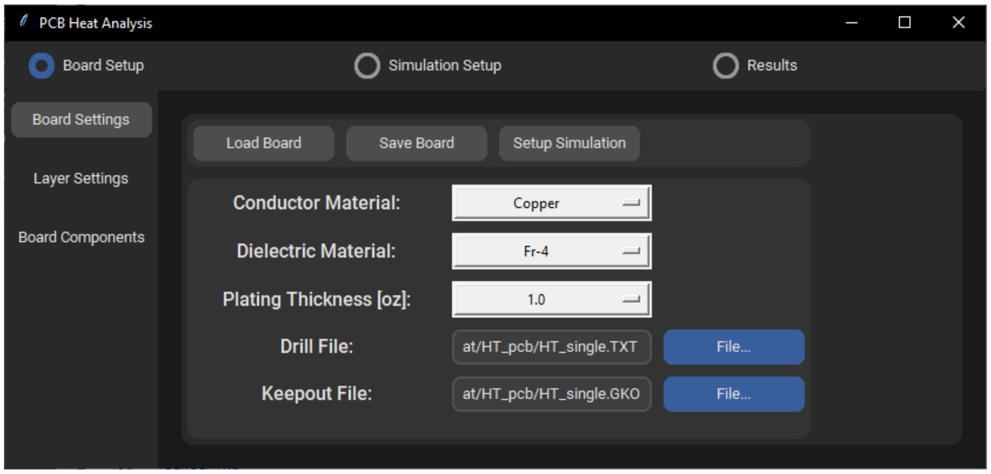
implemented in figure 3.17. With the PCB settings configured, a single board configuration can be run against several different simulation conditions.

3.5.2 Simulation Settings

The simulation properties include the resolution in which the simulation should run, the ambient air temperature in ^oC and the orientation of the board. The simulation resolution will impact the accuracy of the simulation as well as the time needed to solve the heat transfer equations. The board orientation is needed since free convection conditions change depending on a surface's orientation to gravity - the driving force to free convection. For this research, the board orientation was kept to one of five conditions, zero rotation along x-axis and y-axis, $\pm 90^o$ rotation along x-axis, and $\pm 90^o$ rotation along y-axis. These first set of simulation settings can be seen implemented in figure 3.18. To capture conduction losses and the heat generated by them in the PCB, the electrical DC loads need to be identified with their DC current value in amperes along with start and finish (X, Y) coordinates on the board. The simulation load settings can be seen implemented in figure 3.19. Finally, any component which is generating heat for a specific simulation case needs to be specified as seen implemented in figure 3.20.

3.5.3 View Results

With the board and simulation settings defined, the simulation is ready to be run and the heat equations calculated. The primary output for this analysis is the temperature map of the PCB as seen implemented in figure 3.23. The temperature is mapped to the color index on the right-hand side of the plot. This particular color index was chosen to best match the one used by the thermal camera used for hardware validation. Other items useful to examine are the conductor traces along any given layer as can be examined in the implemented screen seen in figure 3.21. Also, examining the conduction losses can be a useful check to see if the simulation is setting up the problem as expected, as well as to interrogate the PCB design for areas that could be improved. The conduction losses can be examined in the screen implemented in fig 3.22.



Conductor Material: Copper

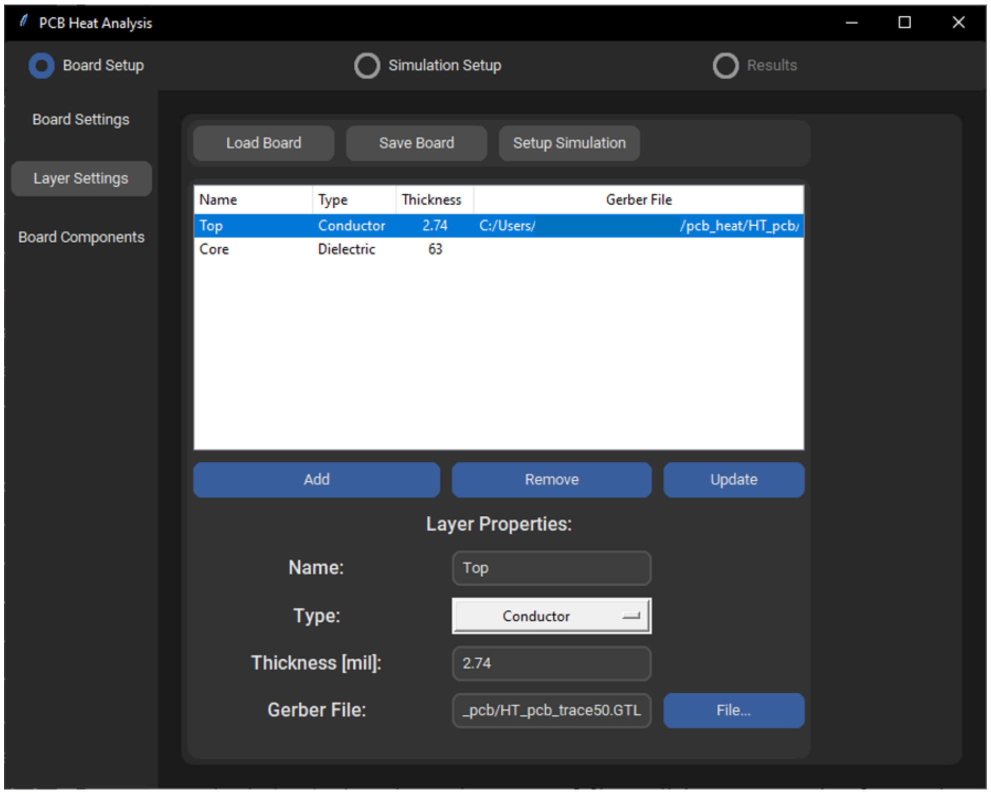
Dielectric Material: Fr-4

Plating Thickness [oz]: 1.0

Drill File: at/HT_pcb/HT_single.TXT

Keepout File: at/HT_pcb/HT_single.GKO

Fig. 3.15: GUI - general board settings.



Name	Type	Thickness	Gerber File
Top	Conductor	2.74	C:/Users/ /pcb_heat/HT_pcb/
Core	Dielectric	63	

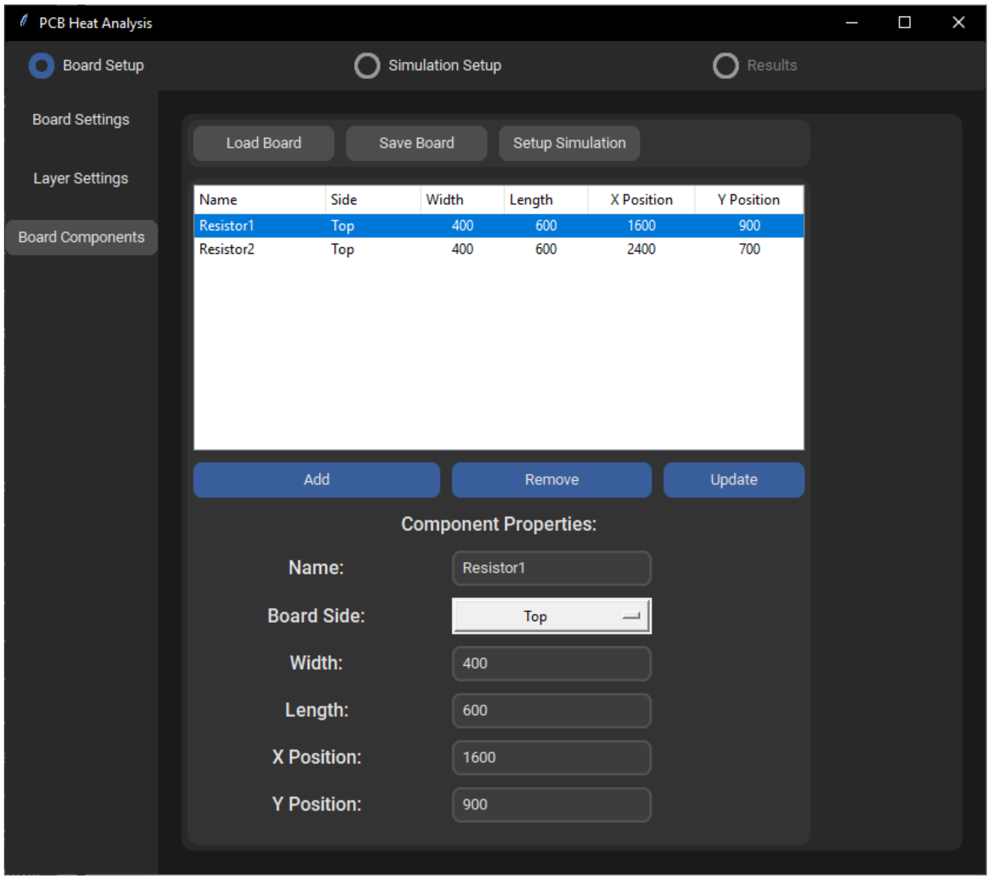
Name: Top

Type: Conductor

Thickness [mil]: 2.74

Gerber File: _pcb/HT_pcb_trace50.GTL

Fig. 3.16: GUI - board layer settings.

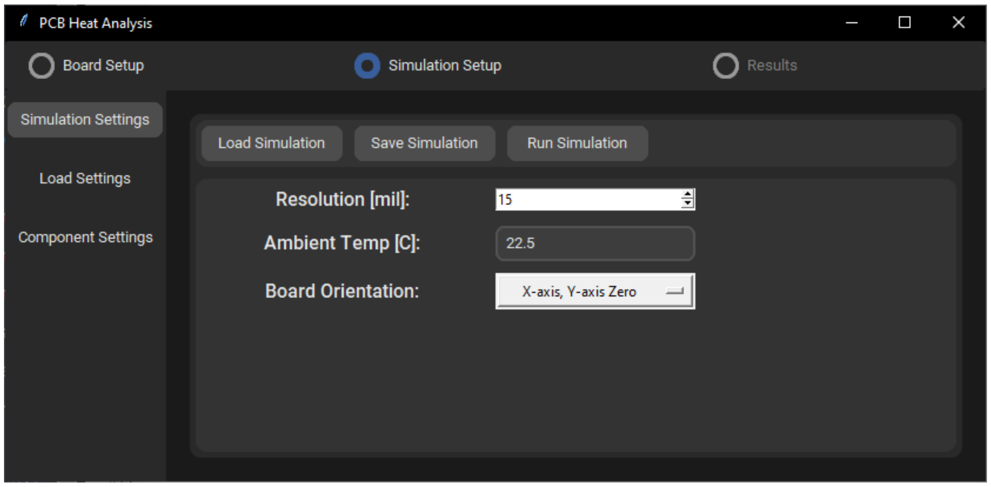


Name	Side	Width	Length	X Position	Y Position
Resistor1	Top	400	600	1600	900
Resistor2	Top	400	600	2400	700

Component Properties:

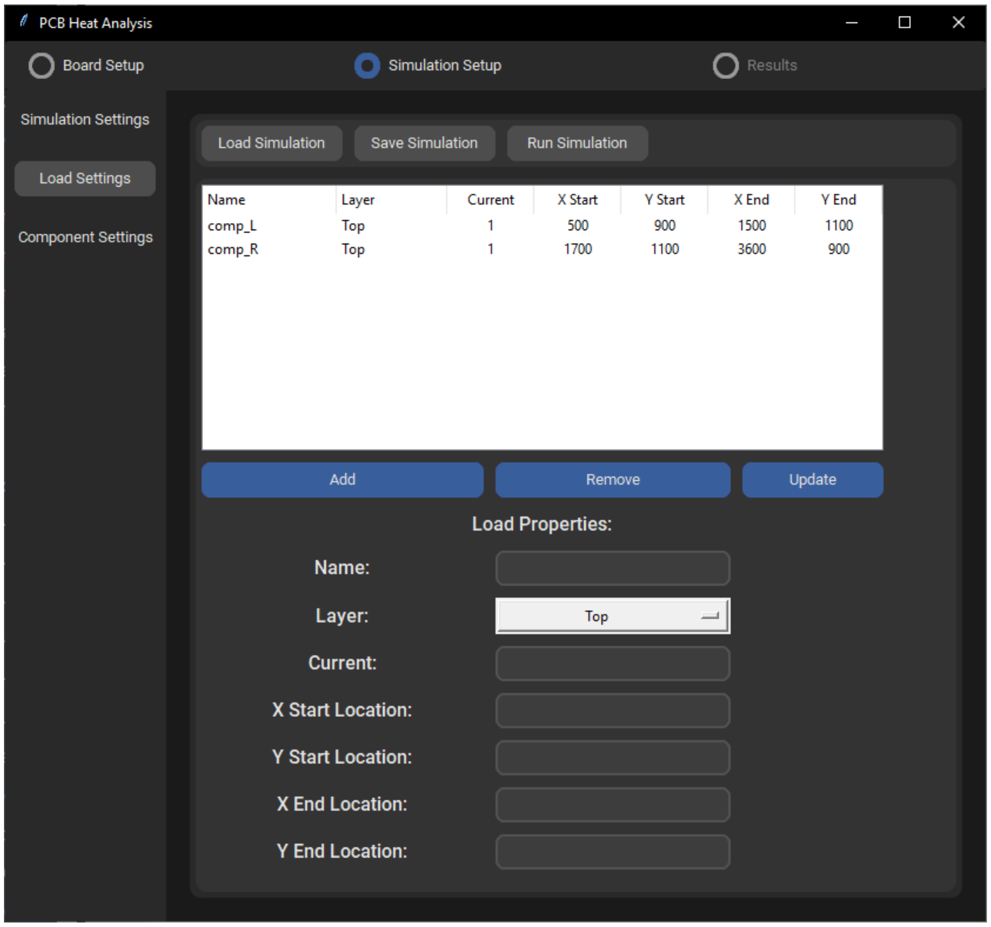
Name:	Resistor1
Board Side:	Top
Width:	400
Length:	600
X Position:	1600
Y Position:	900

Fig. 3.17: GUI - board component settings.



Resolution [mil]:	15
Ambient Temp [C]:	22.5
Board Orientation:	X-axis, Y-axis Zero

Fig. 3.18: GUI - general simulation settings.



Name	Layer	Current	X Start	Y Start	X End	Y End
comp_L	Top	1	500	900	1500	1100
comp_R	Top	1	1700	1100	3600	900

Load Properties:

Name:

Layer: Top

Current:

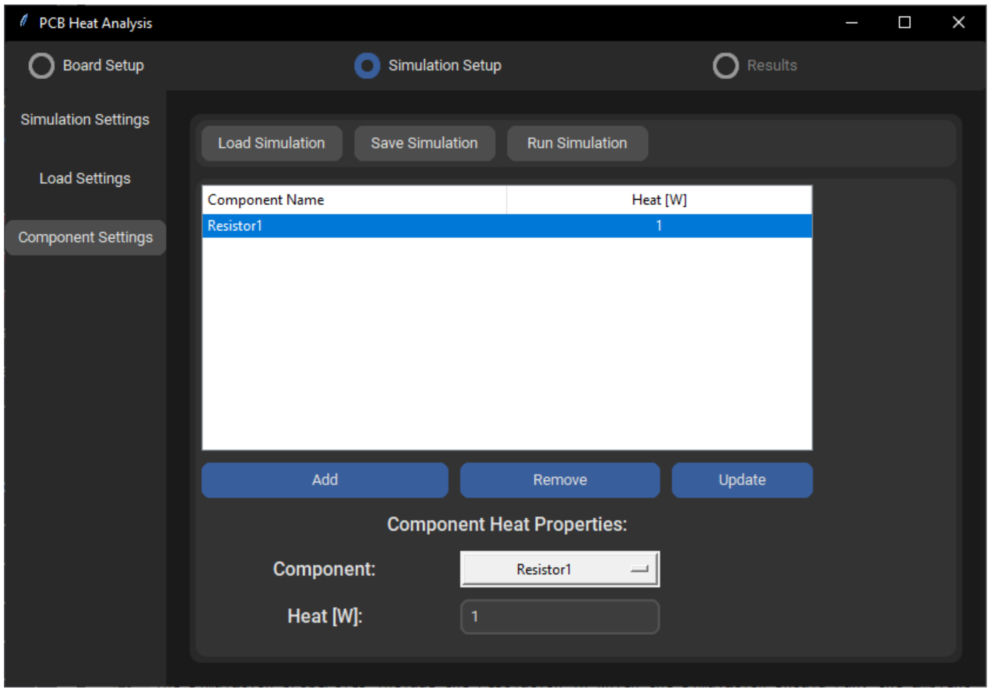
X Start Location:

Y Start Location:

X End Location:

Y End Location:

Fig. 3.19: GUI - simulation load settings.



Component Name	Heat [W]
Resistor1	1

Component Heat Properties:

Component: Resistor1

Heat [W]: 1

Fig. 3.20: GUI - simulation component heat settings.

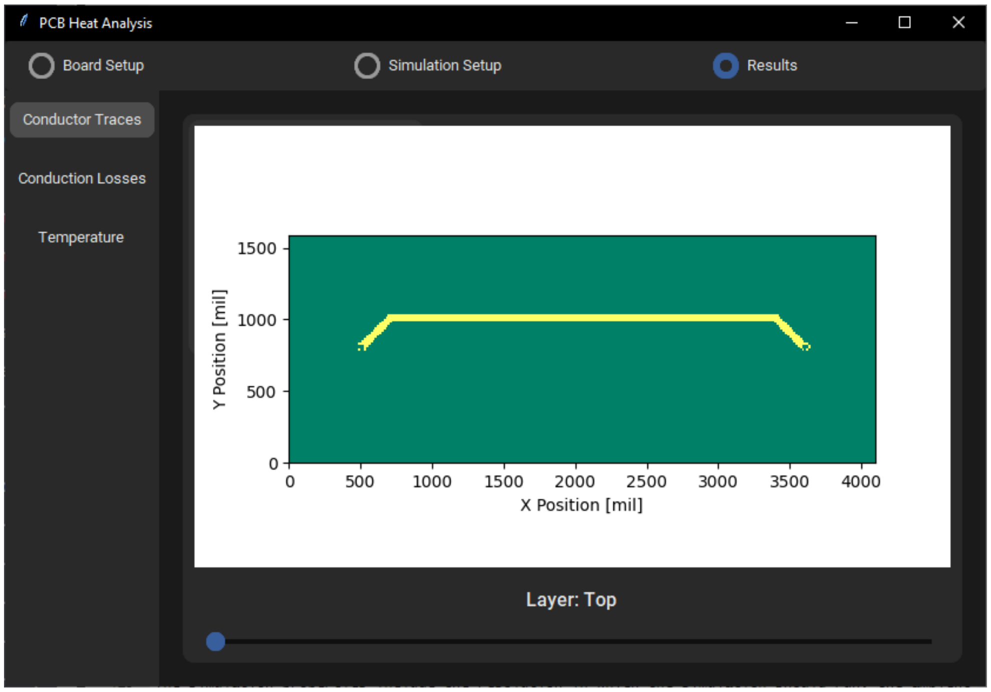


Fig. 3.21: GUI - conductor trace results.

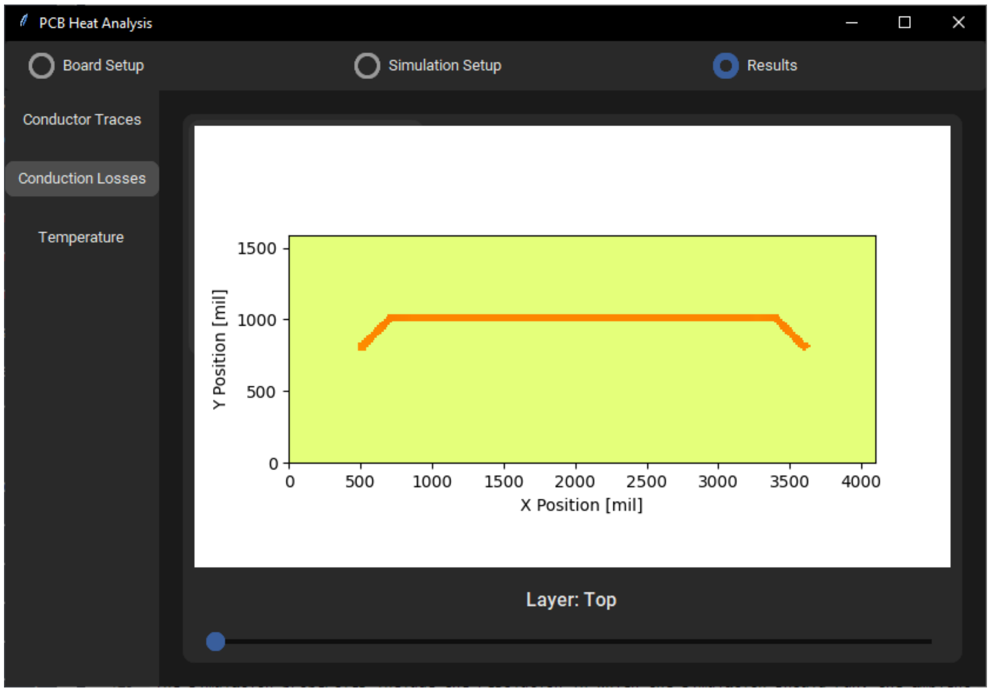


Fig. 3.22: GUI - conduction loss results.

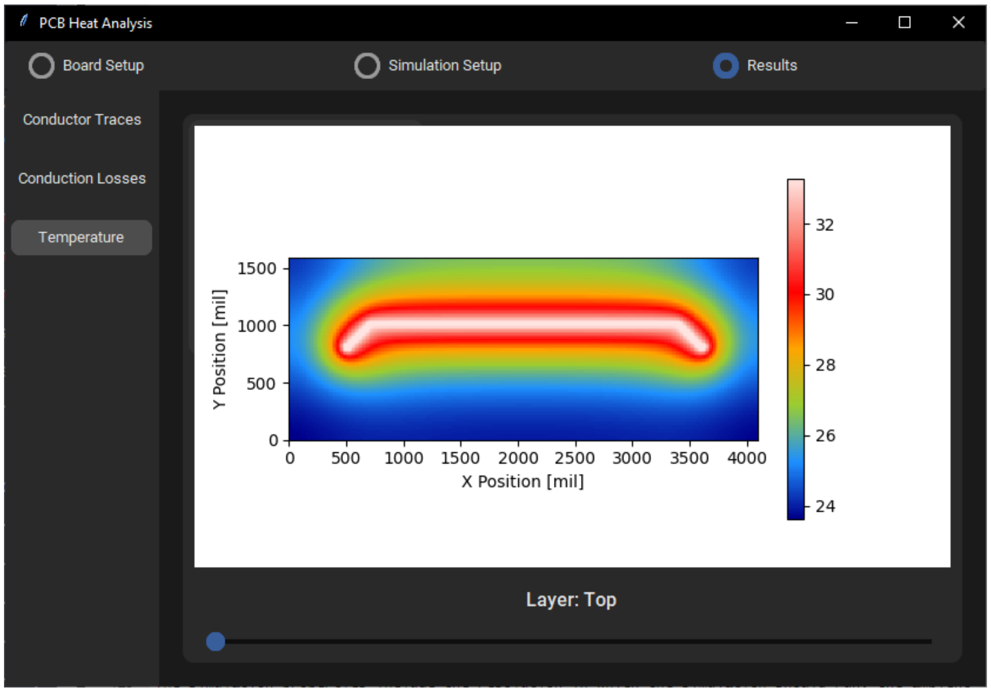


Fig. 3.23: GUI - temperature results.

CHAPTER 4
SIMULATION VERIFICATION

4.1 Physical Testing

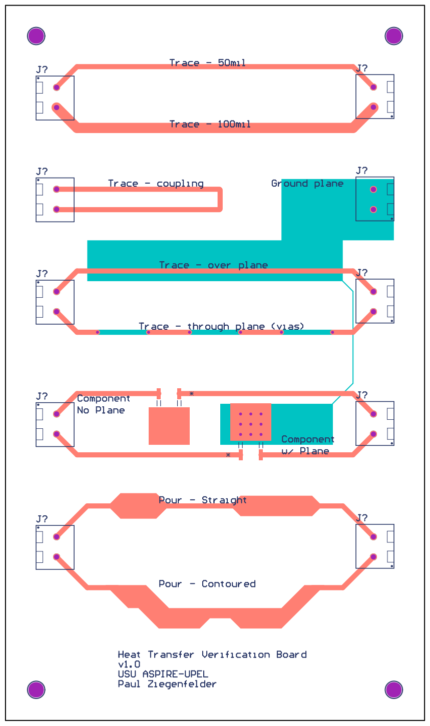
In order to both verify and tune the algorithm as explained in chapter 3, the results need to be compared to some baseline. Using the IPC 2152 standard as a baseline is a possibility but as explored by other researchers, is often over conservative in its temperature rise estimation [5]. As physical testing would make the ideal comparison, a test plan was created for the purposes of tuning the simulation as well as investigate several features or effects common in PCB design that are not covered in IPC-2152 or other design aides. Table 4.1 outlines the testings for each trace that would be needed as well the different current loads.

Table 4.1: Board test plan

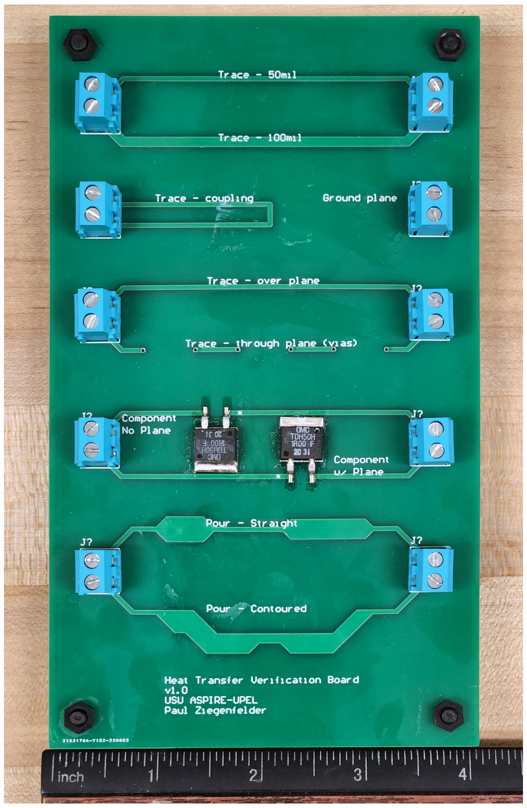
			DC Current [A]					
Test No	Trace Name	Purpose	1	1.5	4	6	8	10
1.1 - 1.4	50 mil	tuning			X	X	X	X
2.1 - 2.4	100 mil	tuning			X	X	X	X
3.1 - 3.4	thermal coupling	effect			X	X	X	X
4.1 - 4.4	over plane	effect			X	X	X	X
5.1 - 5.4	inter-layer	effect			X	X	X	X
6.1 - 6.2	component (isolated pad)	effect	X	X				
7.1 - 7.2	component (multi-layer pad)	effect	X	X				

4.1.1 Test PCB

From the test plan shown in table 4.1, the board shown in figure 4.1 was designed and created using 2 oz. copper for both top and bottom layers with a total board thickness around 1.6 mm (63 mil). A single board with multiple traces was chosen to minimize the cost of acquisition. 500 mil spacing between test traces was used to give adequate thermal isolation from one trace to another. A description of each trace and its purpose in the test plan is described below:



(a) Board design: top layer in red, bottom layer in cyan, plated holes in violet



(b) Fabricated board

Fig. 4.1: Test board design and implementation.

50 mil trace

To better correlate between physical testing, simulation and IPC 2152, a trace width at 50 mil was used. IPC 2152 results were pulled from Brooks [11] for 2 oz. layer thickness on an external layer. With this trace width set as a baseline, most all the ”effect” traces used the same trace width so it could be directly compared to this straight trace and what IPC 2152 predicts for a temperature rise. While this trace is useful for comparing to the other effects mentioned, its primary purpose was for tuning the simulation, tested at the four current loads mentioned in table 4.1.

100 mil trace

A straight trace with a width of 100 mil was used as another series of data points to tune the simulation. Testing was performed with the same four current loads of the 50 mil trace. Having a second width and thus a different curve to compare in IPC 2152 results allows for greater confident in the tuning of the simulation.

Thermal coupling

Many design aides including IPC 2152 assume that a trace will have adequate thermal isolation from other features on the board as to avoid thermal coupling - where one heated trace impacts another trace. This isn’t realistic in many design cases as the drive for smaller packaging and proximity of features continually condenses the PCB design features. To demonstrate this effect, the same trace was looped back near itself such that the section where the trace runs close to itself would experience this coupling.

Over plane trace

Unlike the previously mentioned trace, thermal coupling will not always be detrimental to a design. The purpose of this trace is to demonstrate cases that thermal coupling will be advantageous to the thermal performance of the PCB. This trace runs along the top layer but has a sizable polygon pour or plane directly below it on the bottom layer as seen in figure 4.1a. Although there is a layer of FR-4 insulating the top and bottom layers, their close proximity (63 mil thickness of the board) demonstrated this effect.

Inter-layer trace

A common feature in PCB design is the use of vias to connect circuits between conductor layers. The use of vias is particularly useful to thermally diffuse heat from a heat source such as a surface mounted component. The purpose of this trace is to demonstrate the effect that having vias intermittently along the path of a trace and alternating the trace between top to bottom layers will have on the thermal performance of the PCB.

Component on an isolated pad

Surface mounted components such as switches, converters, power resistors, and power diodes are often a large source of heat that a PCB experiences. To demonstrate this effect, a 1Ω power resistor was used at two different DC current loads. The heat generated by this component is found using equation 3.1. It should be noted that this particular component package has the body of the power resistor soldered to a large pad on the top layer. This first trace has an isolated pad meaning it is not connected to any other conductor layers. A heat sink was not used on any portion of the component as to better thermal couple the component to the PCB.

Component on a multi-layer pad

Surface mounted components that are known to produce any substantial amount of heat will have design features in the PCB to help diffuse this heat away from the component. Often a heat sink is used along the top surface of the component to assist with this dissipation. This was avoided in this testing as to better thermal couple the component to the PCB. Instead, another common design feature was implemented which involves connecting the mounting pad to the bottom layer with a pattern of vias and extending the size of the plane along the bottom layer as seen in figure 4.1a. A comparison between the the two component placement designs demonstrates the effectiveness of this feature.

Additional traces

It is difficult to predict the thermal performance of a trace with varying width. An additional two traces were added to the design to evaluate the simulations effectiveness in capturing this effect. One trace is straight in its trajectory while the other has varying width and trajectory. Not much detail will be given to these test traces since they were not fully examined due to limitations of time.

4.1.2 Test Procedure

The test plan outlined in table 4.1 was carried out in an environment of negligible air flow and the ambient air temperature was captured for each test performed. For each trace, the lowest current load was applied and held until the maximum trace temperature was stable. The trace's thermal image was then recorded along with the maximum trace temperature specified in the image. Subsequent tests at increasing current loads were performed until the highest current load test for a trace was completed. The board was then disconnected from all power sources and allowed to cool before proceeding to the next set of tests on another trace.

4.2 Simulation Tuning

Since the simulation is based on a model and no model perfectly captures the effects seen in real life, some corrections to the calculation and output of the simulation can be applied to better match physical testing covered in section 4.1.1.

4.2.1 Means of Adjustment

The following adjustments to the simulation were factors added to either decrease or increase the effect of specific physical parameters in the calculation. This allowed an intuitive manner to tune the simulation to match physical results.

Coefficient for dielectric material thermal conductivity 'k' - in plane

If a simulation result shows that the heat is not dissipating sufficiently from a heat source, whether from a current carrying trace or component, this parameter can be increased. If the heat seems to be dissipating from the heat source too greatly, this parameter should likely be decreased. This parameter was primarily tuned from the results of tests 1.1-1.4 and 2.1-2.4.

Coefficient for dielectric material thermal conductivity 'k' - through plane

If it is observed that dissipating heat from one layer to another through the dielectric material was insufficient, this parameter could be increased and vice versa if observed to dissipate more than anticipated. One difficulty in finding an appropriate value for this parameter is the fact that almost all of the tests listed in table 4.1 have the heat source and visible layer to be the same. Due to this limitation, tests 4.1-4.4 proved to be most useful in evaluating this parameter since the heat from the trace would invariably be thermal coupled to the plane on the layer beneath it and would then heat up the dielectric above it on the top layer.

Coefficients for convection and radiation heat transfer coefficient

If the overall board temperature is simulated to be too high, then it stands to reason that not enough heat is being dissipated away from the PCB. The only manner in which heat escapes the PCB is through convection and radiation. In order to understand which parameter to tune, equations 2.7 and 2.10 need to be reexamined. Seeing how the heat transfer coefficient of radiation is a function of temperature raised to the third power, a hotter surface will have a greater contribution from radiation to the overall dissipation of heat from the PCB. This is one of the primary reasons it was critical to test the PCB at various current loads and thus temperature raises.

If it is observed that the overall board temperature is too high at a greater extent at higher temperatures than lower temperatures, it stands to reason that the heat transfer coefficient from radiation should be increased. If the overall board temperature is more consistently high for all temperature ranges, it stands to reason that the convection heat transfer coefficient should be increased. The same logic applies if the overall board temperature is too low which will result in a lesser heat transfer coefficient being needed.

Power for radiation heat transfer coefficient

Even with the distinction between convection and radiation coefficients, given the fact that radiation is a function of temperature raised to a power means that a simple multiplication factor may not be enough to adequately capture it behavior. An additional factor of ΔT raised to some value can be applied to give greater flexibility in this regard.

Coefficient for component heat transfer coefficient

For this research, a component heat source is assumed to be a flat rectangle whose internal heat is either dissipated by convection and radiation into ambient air or is dissipated into the PCB through its contact area with its neighboring layer, whether the top layer for top mounted components or the bottom layer for bottom mounted components. The heat transfer coefficient for convection and radiation will directly impact the amount of heat being dissipated into the PCB so a coefficient altering this amount is useful. Tests 6.1, 6.2, 7.1 and 7.2 were most useful in tuning this parameter as they pertain to the surface mounted power resistor.

4.2.2 Difficulties in Tuning

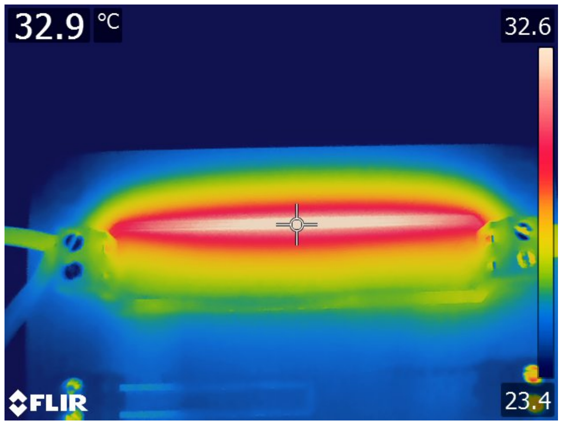
Since the effects of one tuning parameter are not truly decoupled from another, it was found that changing one parameter often lead to having to revisit the tuning of another parameter. This was a fairly time intensive task. It should be noted that all tuning adjustments were performed at a constant simulation resolution. Altering the resolution changed the simulation behavior and required new tuning factor values. For this research, the resolution was kept constant for all tests and subsequent result.

4.3 Simulation Results

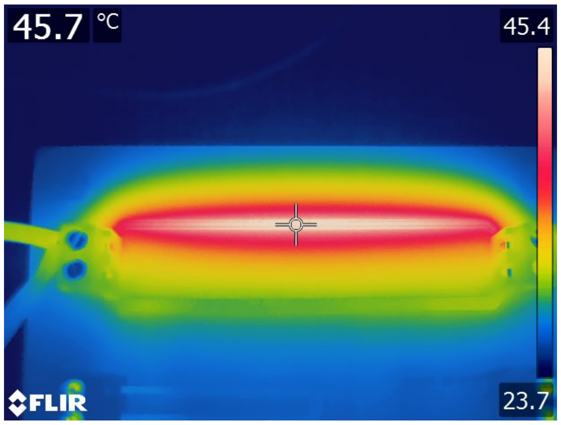
Using the tuning factors mentioned above, the following simulation results were found. It should be noted that all of the below test results have the same tuning factors applied and are all set at the same simulation resolution.

4.3.1 Tuning Traces

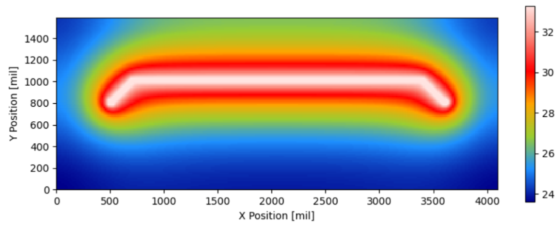
Figures 4.2 and 4.3 show the physical test results as captured with a thermal camera, as well as the output results from the simulation for each particular test with each sub-figure showing progressively larger current loads. Lower temperature rises had better correlation, both in absolute error as well as percent error. For the 50 and 100 mil wide traces, the simulation was able to tune within 17% and 10% respectively as seen in figure 4.13. The simulated results are closer to the actual results than what IPC 2152 predicts, which is conservative in all cases. Overall, the simulation tracks well with the tested results.



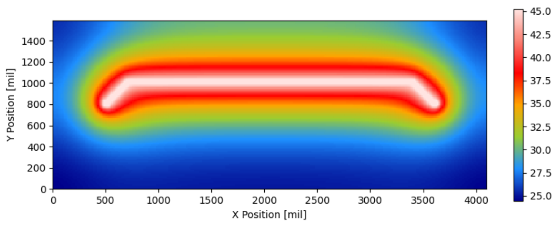
(a) Test 1.1: 4 Amp load.



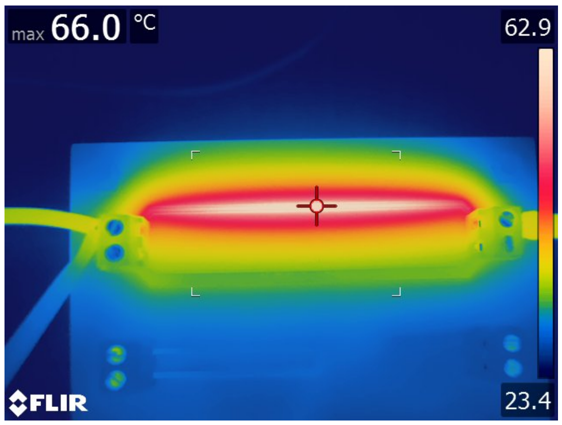
(b) Test 1.2: 6 Amp load.



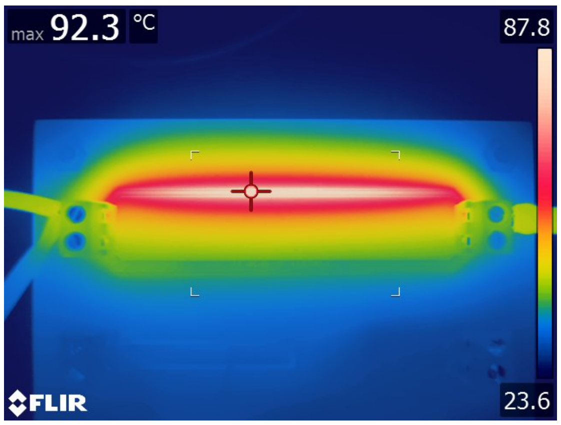
(c) Simulation 1.1: 4 Amp load.



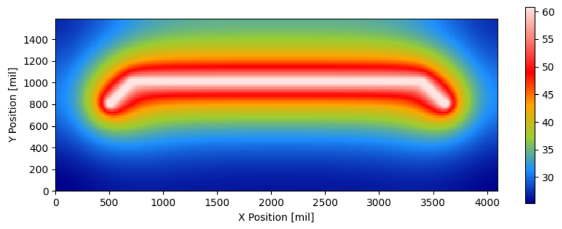
(d) Simulation 1.2: 6 Amp load.



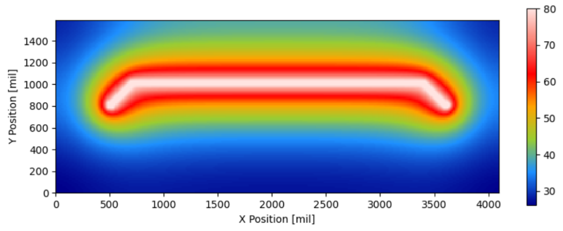
(e) Test 1.3: 8 Amp load.



(f) Test 1.4: 10 Amp load.

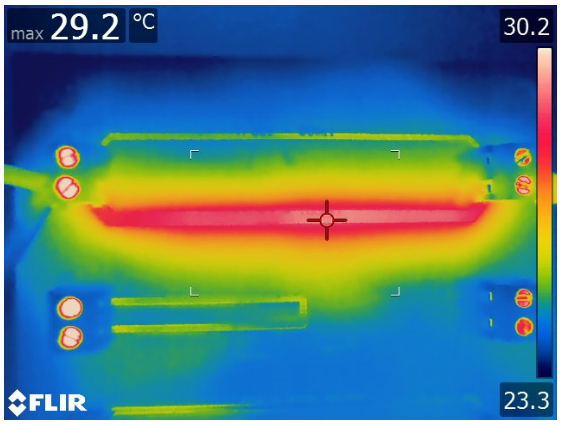


(g) Simulation 1.3: 8 Amp load.

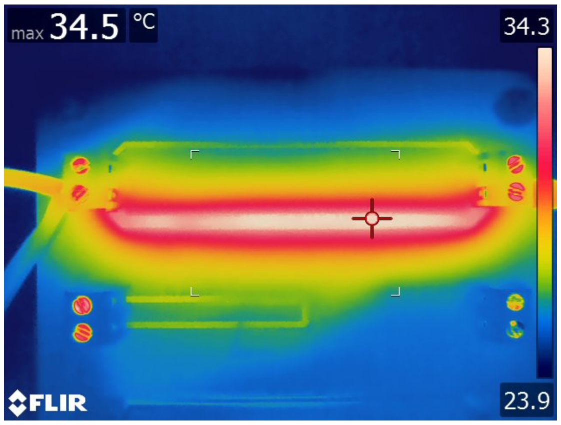


(h) Simulation 1.4: 10 Amp load.

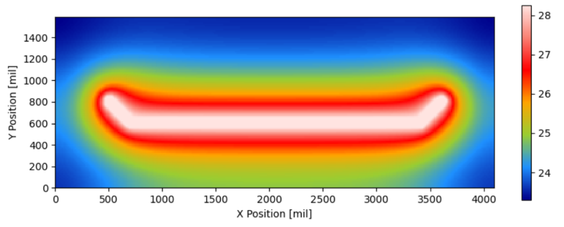
Fig. 4.2: 50 mil trace test results.



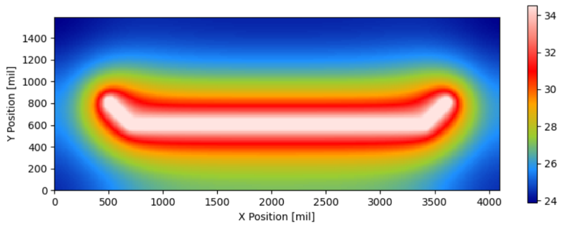
(a) Test 2.1: 4 Amp load.



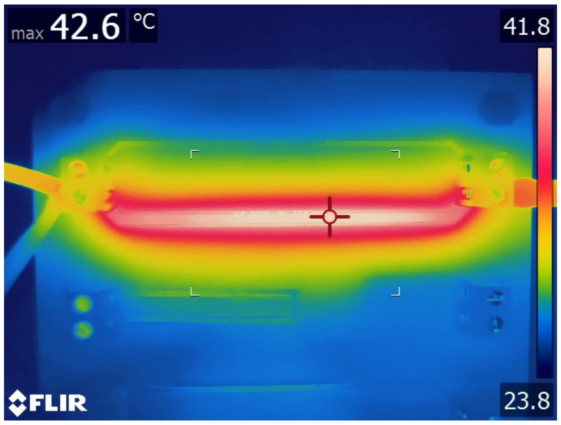
(b) Test 2.2: 6 Amp load.



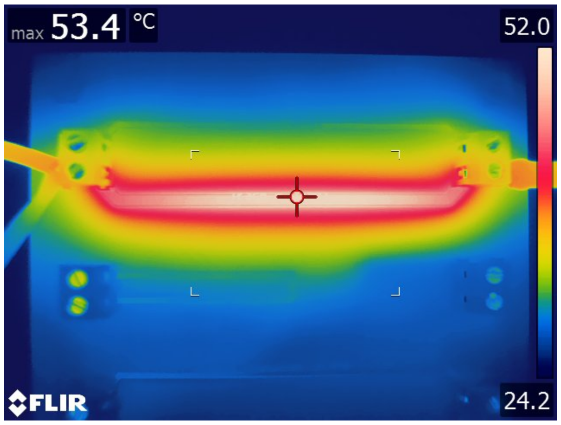
(c) Simulation 2.1: 4 Amp load.



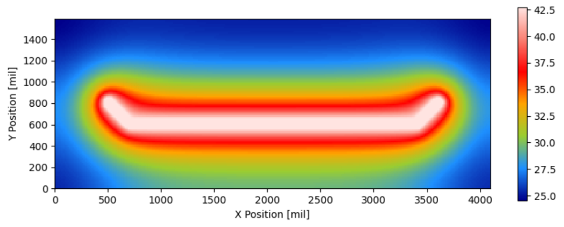
(d) Simulation 2.2: 6 Amp load.



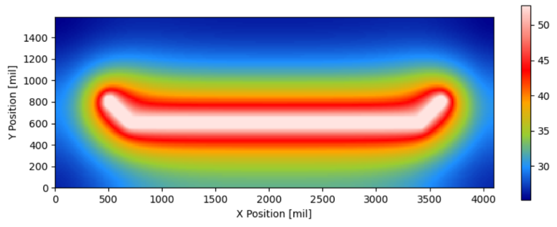
(e) Test 2.3: 8 Amp load.



(f) Test 2.4: 10 Amp load.

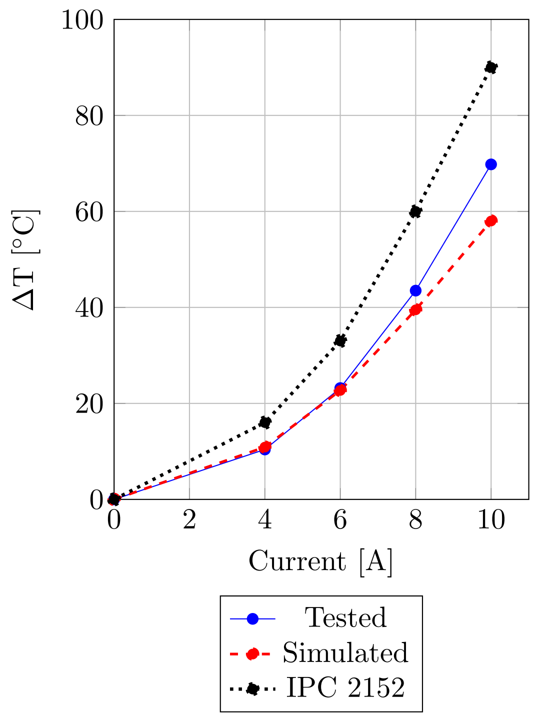


(g) Simulation 2.3: 8 Amp load.

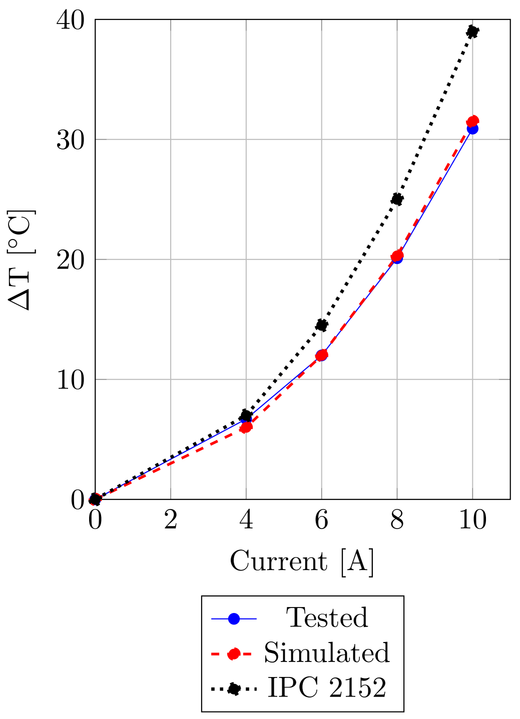


(h) Simulation 2.4: 10 Amp load.

Fig. 4.3: 100 mil trace test results.



(a) 50 mil trace



(b) 100 mil trace

Fig. 4.4: Correlation for tuning traces.

4.3.2 Effect Traces

The remaining tests and results were performed to demonstrate the simulation's ability to account for common design effects not captured in IPC 2152 and other design aides. These include thermal coupling, inter-layer current flow by use of vias and surface mounted components.

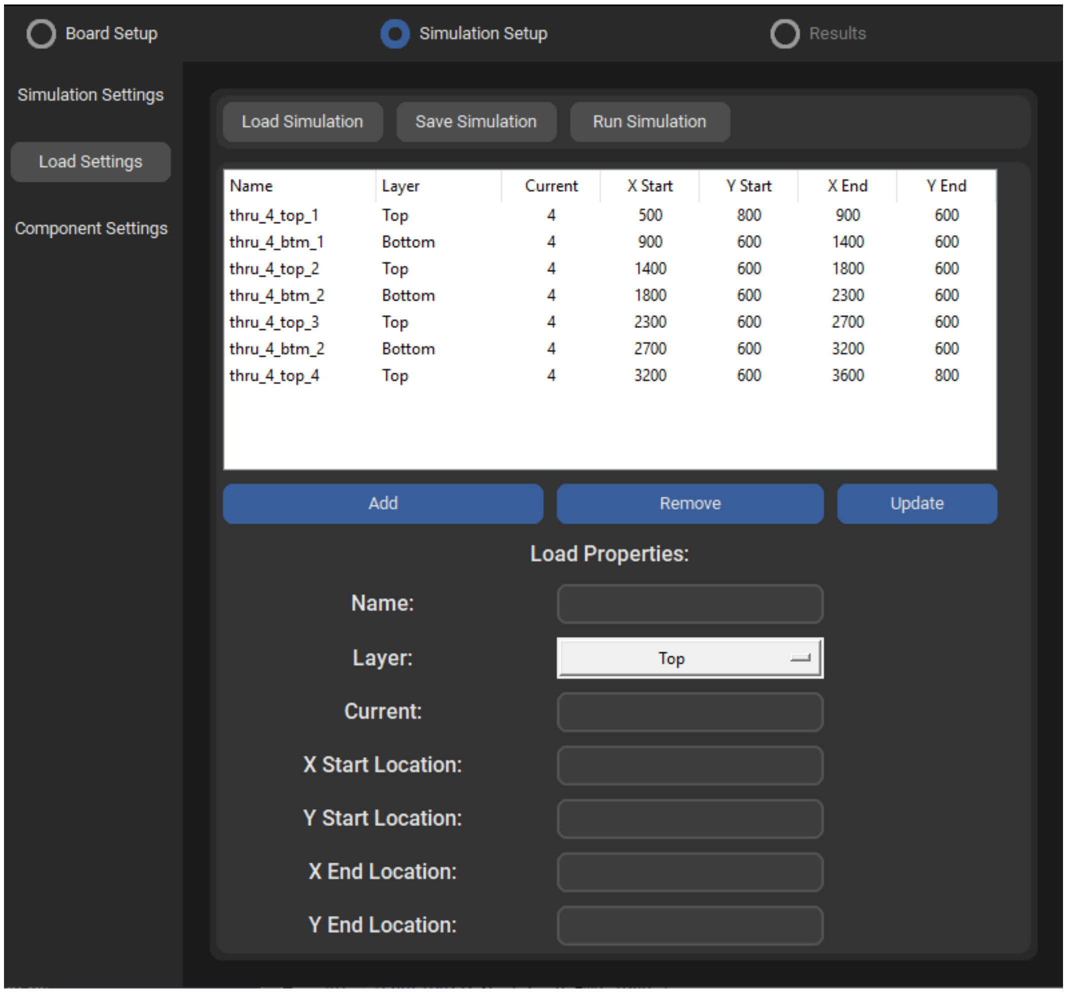
Thermal Coupling Correlation

Tests 3 and 4 were included to demonstrate how well the simulation represents the thermal coupling in traces. Figure 4.7 shows that the simulation is able to follow the general form of the heat map seen in the physical tests for test 3. However, the numeric values didn't track as well when compared to the straight trace in test 1 and can be seen in figure 4.12a. Moreover, figure 4.13 illustrates that the simulated values had an absolute percent error in temperature rise of as high as 24%. While the simulation was able to closely represent an isolated trace, this test shows that the simulation was not able to represent the negative affects of thermal coupling as all results are unconservative. Since IPC 2152 is quite conservative for this sized trace, its predicted values are still slightly conservative for this particular trace design.

For how poorly test 3 correlated, test 4 correlated much better as seen in figure 4.8. The numerical values matched very well as seen in figure 4.12b. It should be noted that IPC 2152 over predicts the temperature rise by a large margin since it does not take into account the thermal coupling of the heated trace to the large plane below it. This demonstrates one benefit of simulating this PCB feature over using the IPC 2152 prediction.

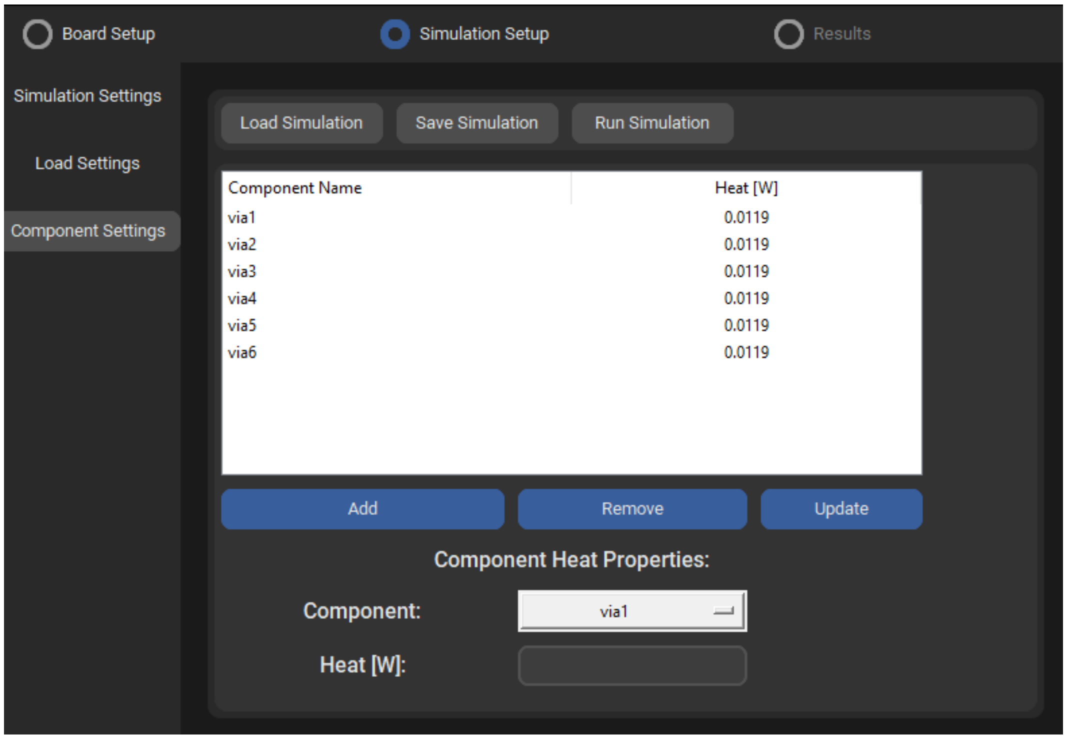
Inter-layer Correlation

For test 5, the simulation is not natively setup to handle current flowing through the vias. This means that the single current trace needed to be entered as seven different current loads, alternating from the top and bottom layers. This setup can be seen in figure 4.5. Figures 4.9 and 4.10 illustrate how poorly this simulation matches the physical testing. This is likely due to the fact that the simulation does not natively handle current being passed through vias between layers. To better correlate, component heat sources with heat values calculated using equation 3.2 were manually added to the board setup. The resistance values were calculated using the cross sectional area of the plated hole as the area A and the dielectric layer thickness as the length l. This setup can be seen in figure 4.6. Figures 4.9 and 4.10 illustrate that this added setup correlates very well with the physical testing as seen in figure 4.12c.



Name	Layer	Current	X Start	Y Start	X End	Y End
thru_4_top_1	Top	4	500	800	900	600
thru_4_btm_1	Bottom	4	900	600	1400	600
thru_4_top_2	Top	4	1400	600	1800	600
thru_4_btm_2	Bottom	4	1800	600	2300	600
thru_4_top_3	Top	4	2300	600	2700	600
thru_4_btm_2	Bottom	4	2700	600	3200	600
thru_4_top_4	Top	4	3200	600	3600	800

Fig. 4.5: GUI screen with seven loads to represent the current through the single trace.



Component Name	Heat [W]
via1	0.0119
via2	0.0119
via3	0.0119
via4	0.0119
via5	0.0119
via6	0.0119

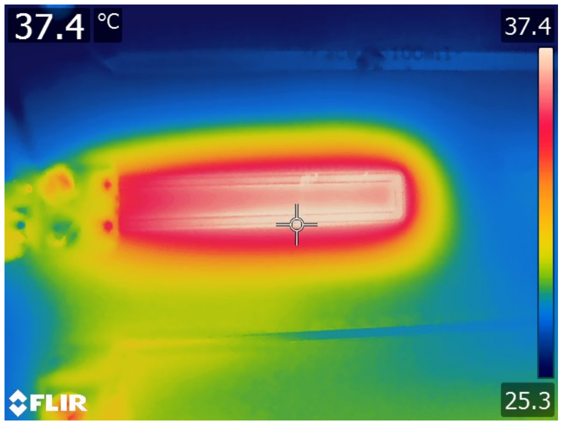
Component: via1

Heat [W]:

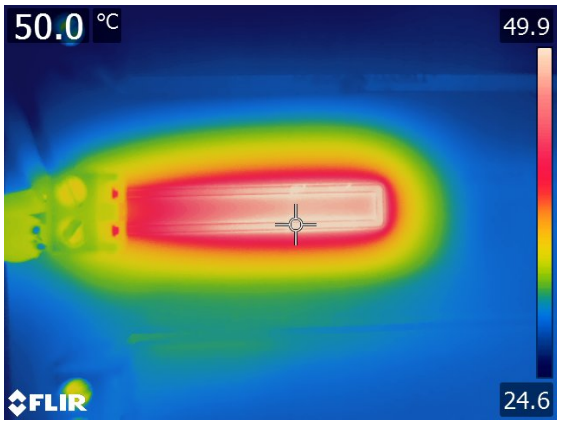
Fig. 4.6: GUI screen with six components to represent the conduction losses through each via.

Surface Mounted Component Correlation

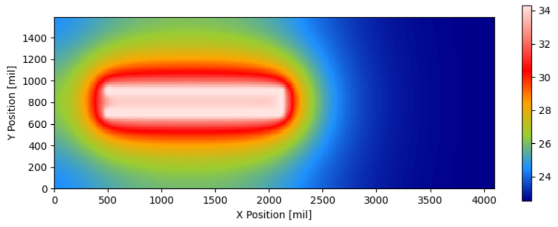
Tests 6 and 7 were created to demonstrate the simulation's ability to account for the use of surface mounted components, particularly those whose heat generation is significant enough that it needs to be considered for the thermal performance of the PCB. The low current used in these tests meant that the majority of the heat generated is from the component and not conduction losses in the traces. This is better seen in the physical testing than in the simulation results seen in figure 4.11. It should be noted that test 6 was used to tune the simulation's component correction factor and the same correction factor was used to verify the results in test 7 which correlate excellently as illustrated in figure 4.12d.



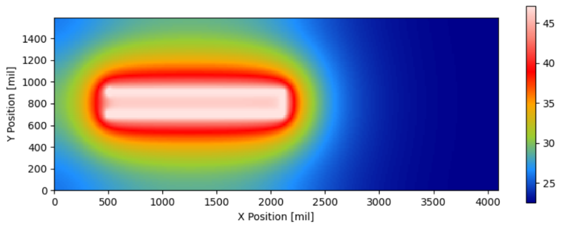
(a) Test 3.1: 4 Amp load.



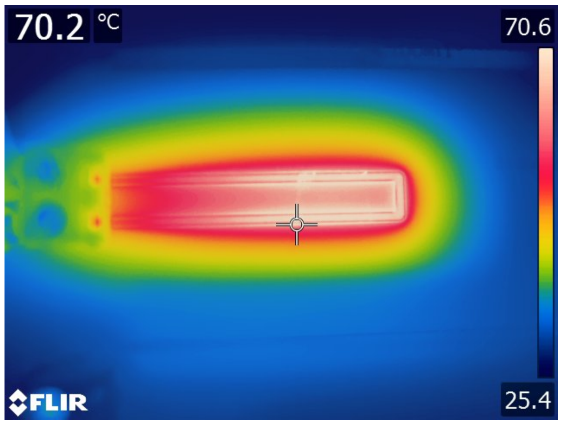
(b) Test 3.2: 6 Amp load.



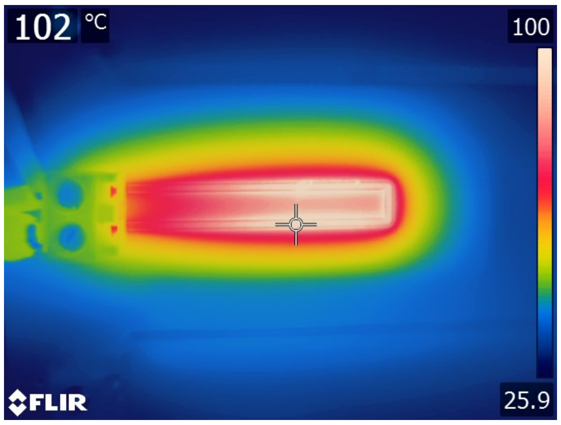
(c) Simulation 3.1: 4 Amp load.



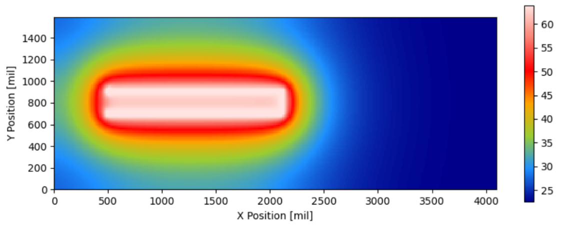
(d) Simulation 3.2: 6 Amp load.



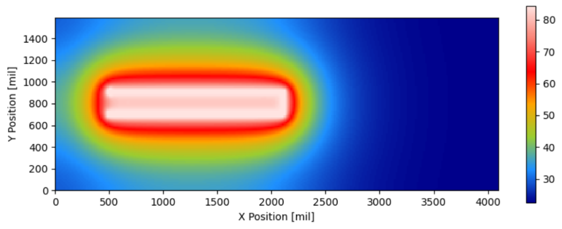
(e) Test 3.3: 8 Amp load.



(f) Test 3.4: 10 Amp load.

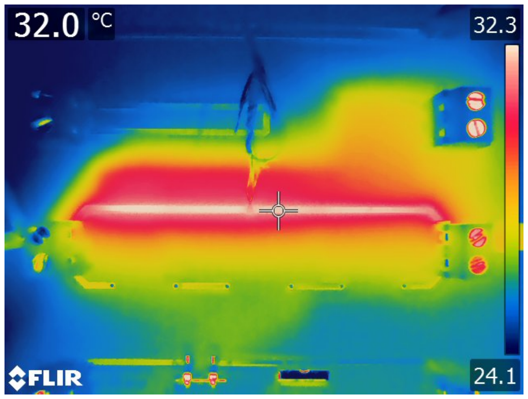


(g) Simulation 3.3: 8 Amp load.

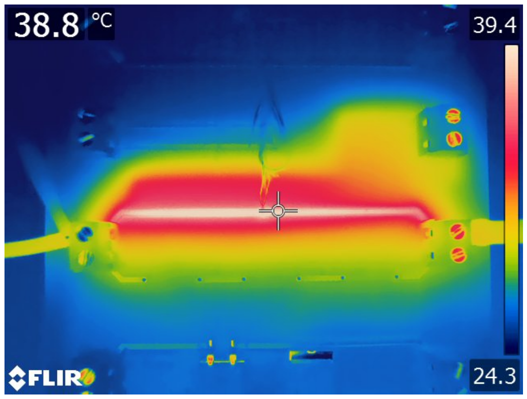


(h) Simulation 3.4: 10 Amp load.

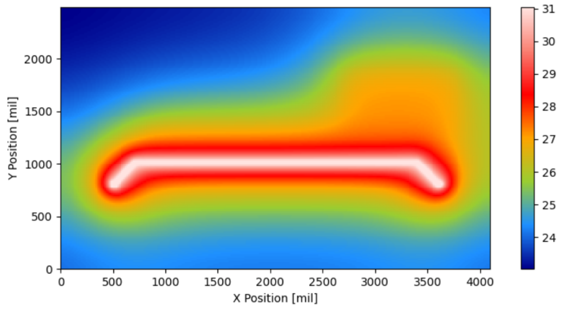
Fig. 4.7: Thermal coupling trace test results.



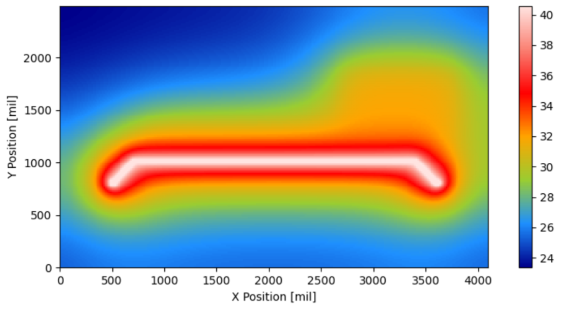
(a) Test 4.1: 4 Amp load.



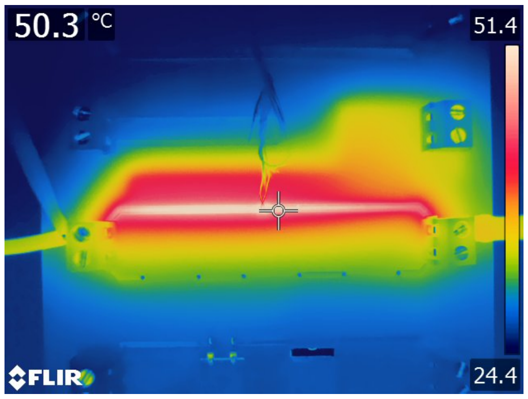
(b) Test 4.2: 6 Amp load.



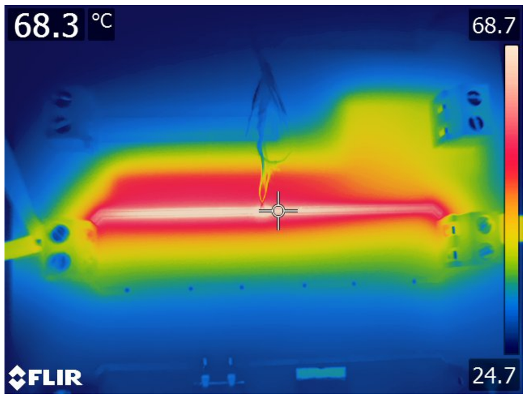
(c) Simulation 4.1: 4 Amp load.



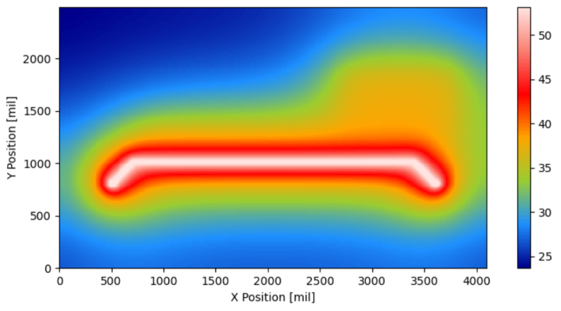
(d) Simulation 4.2: 6 Amp load.



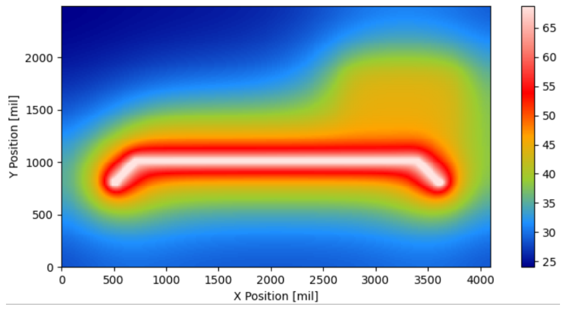
(e) Test 4.3: 8 Amp load.



(f) Test 4.4: 10 Amp load.

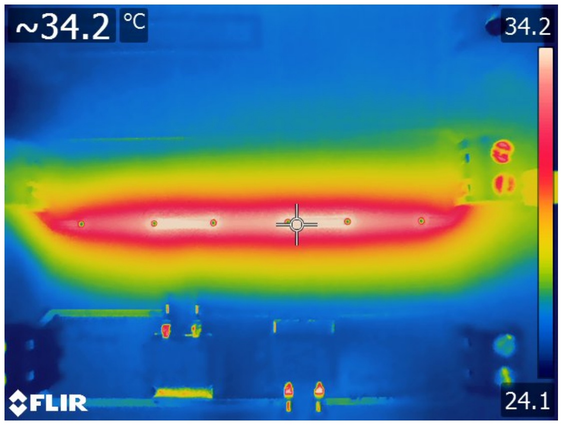


(g) Simulation 4.3: 8 Amp load.

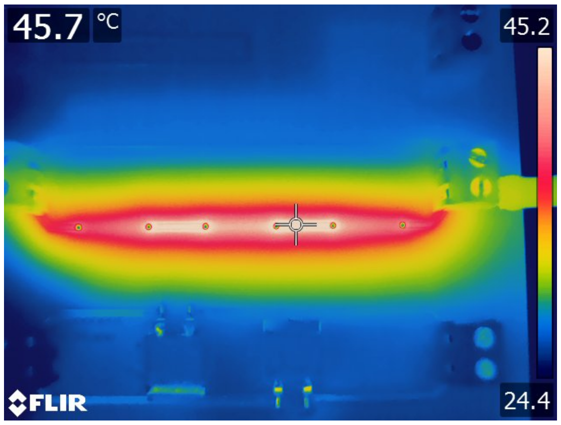


(h) Simulation 4.4: 10 Amp load.

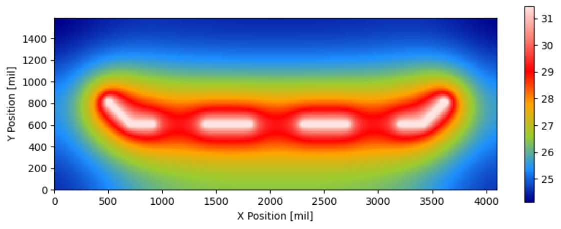
Fig. 4.8: Over plane trace test results.



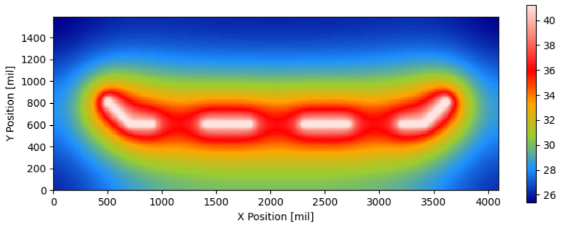
(a) Test 5.1: 4 Amp load.



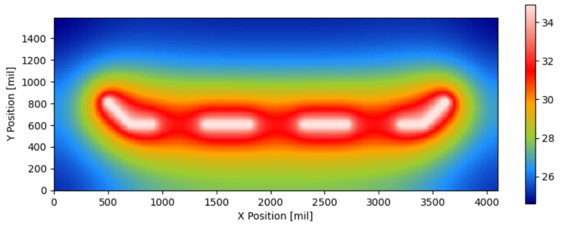
(b) Test 5.2: 6 Amp load.



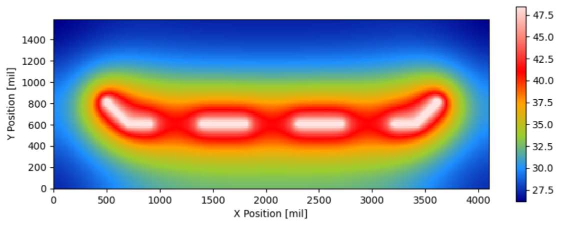
(c) Simulation 5.1: 4 Amp load without vias.



(d) Simulation 5.2: 6 Amp load without vias.

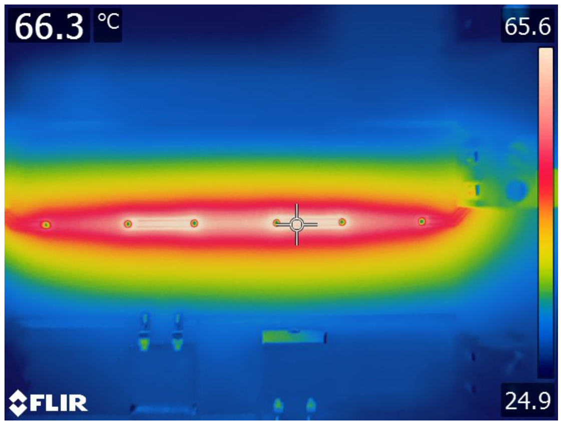


(e) Simulation 5.1: 4 Amp load with vias as component.

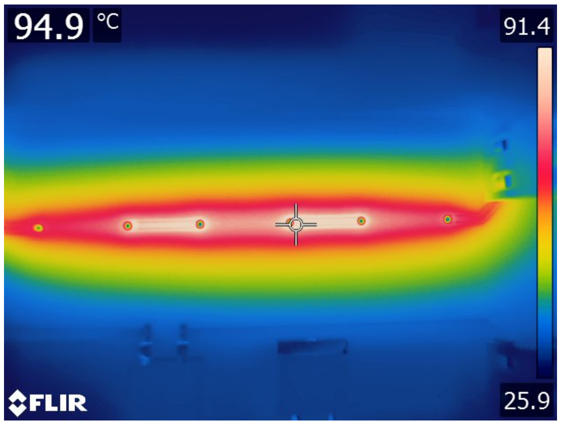


(f) Simulation 5.2: 6 Amp load with vias as component.

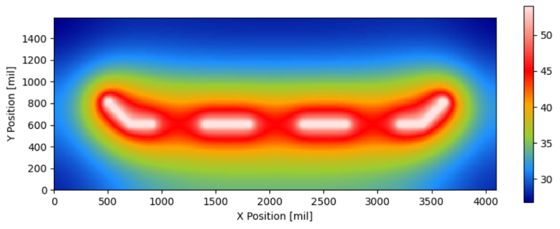
Fig. 4.9: Inter-layer trace test results.



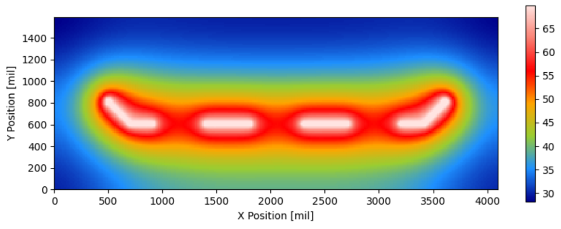
(a) Test 5.3: 8 Amp load.



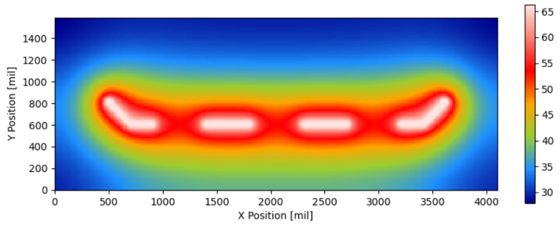
(b) Test 5.4: 10 Amp load.



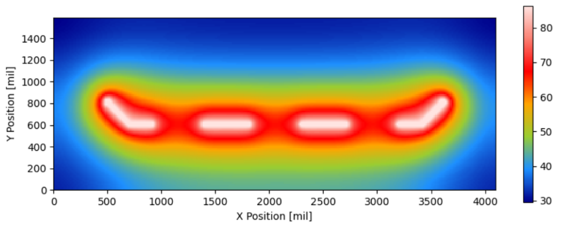
(c) Simulation 5.3: 8 Amp load without vias.



(d) Simulation 5.4: 10 Amp load without vias.

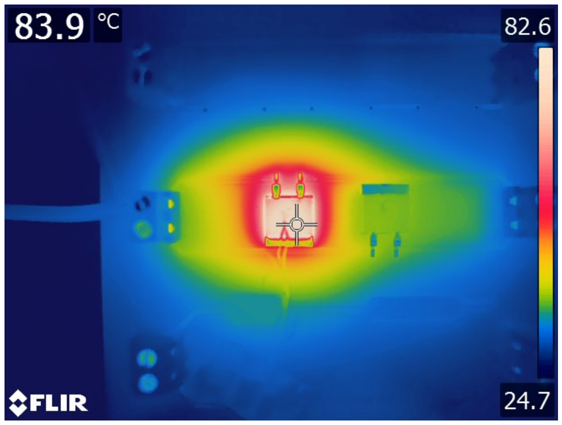


(e) Simulation 5.3: 8 Amp load with vias as component.

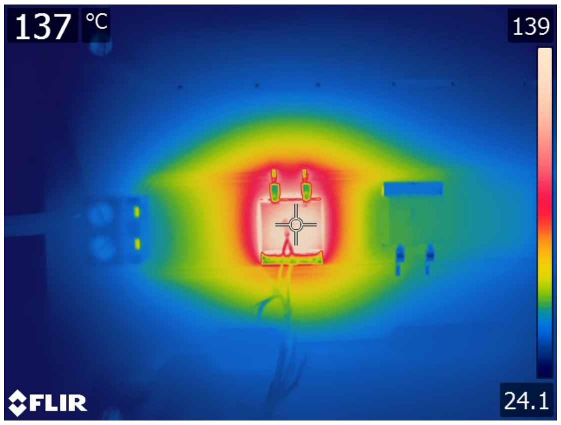


(f) Simulation 5.4: 10 Amp load with vias as component.

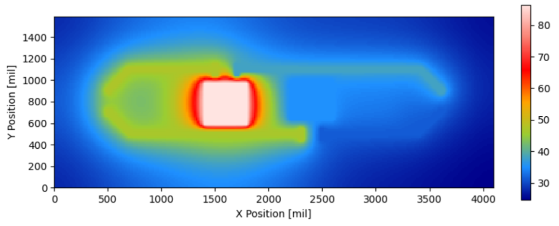
Fig. 4.10: Inter-layer trace test results continued.



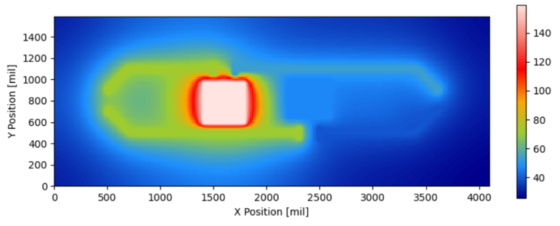
(a) Test 6.1: Isolated pad - 1 Amp load.



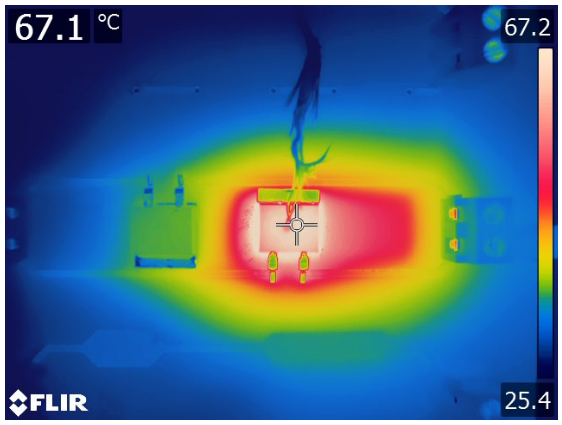
(b) Test 6.2: Isolated pad - 1.5 Amp load.



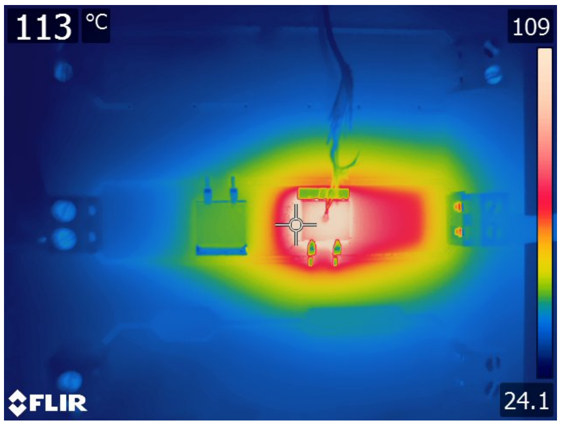
(c) Simulation 6.1: Isolated pad - 1 Amp load.



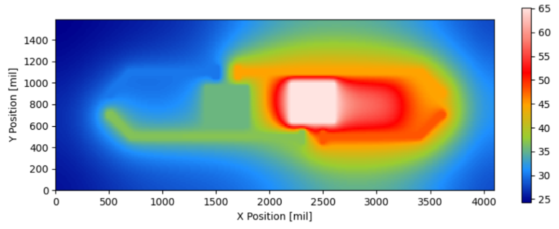
(d) Simulation 6.2: Isolated pad - 1.5 Amp load.



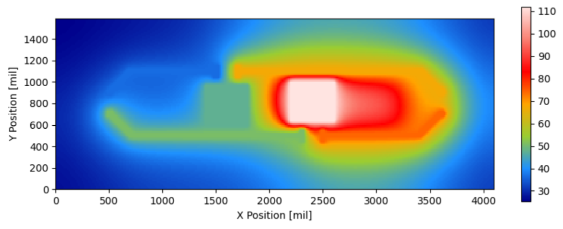
(e) Test 7.1: Multi-layer pad - 1 Amp load.



(f) Test 7.2: Multi-layer pad - 1.5 Amp load.

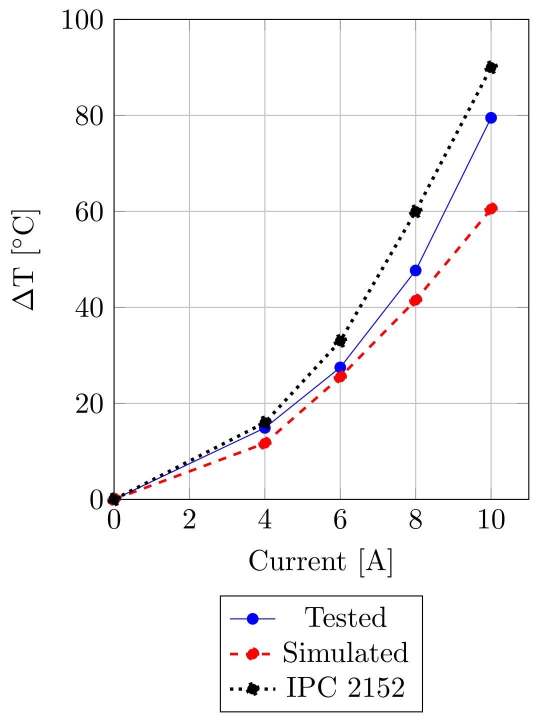


(g) Simulation 7.1: Multi-layer pad - 1 Amp load.

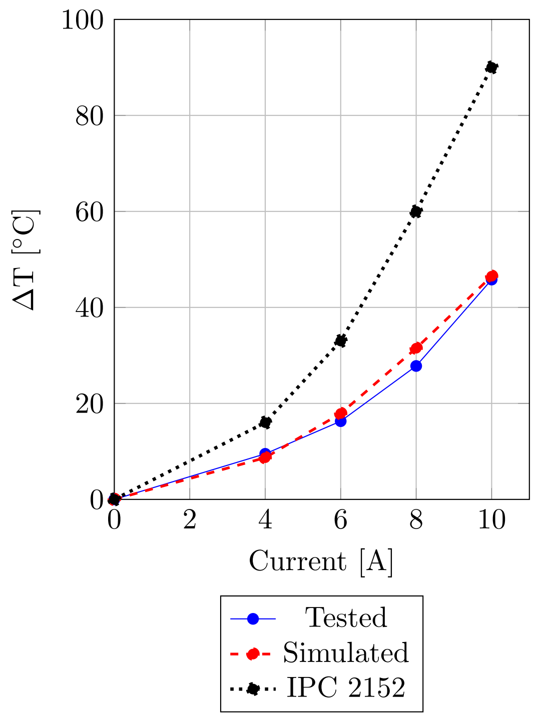


(h) Simulation 7.2: Multi-layer pad - 1.5 Amp load.

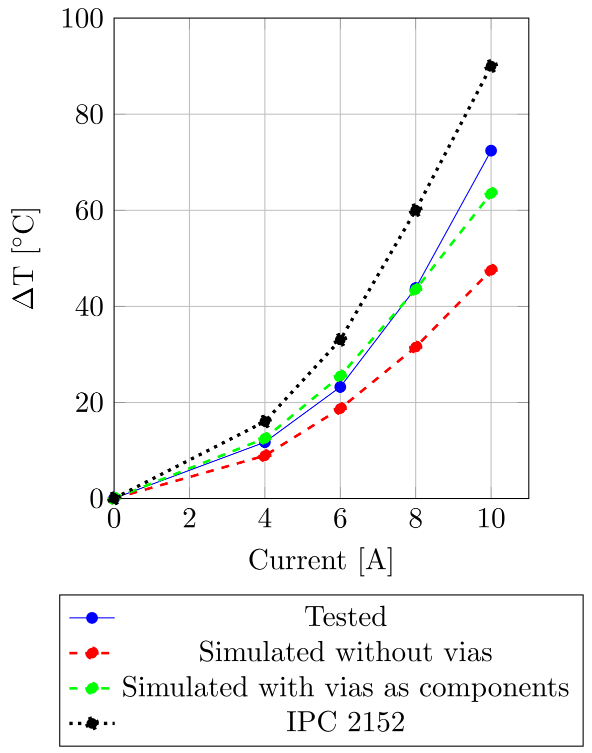
Fig. 4.11: Component trace test results.



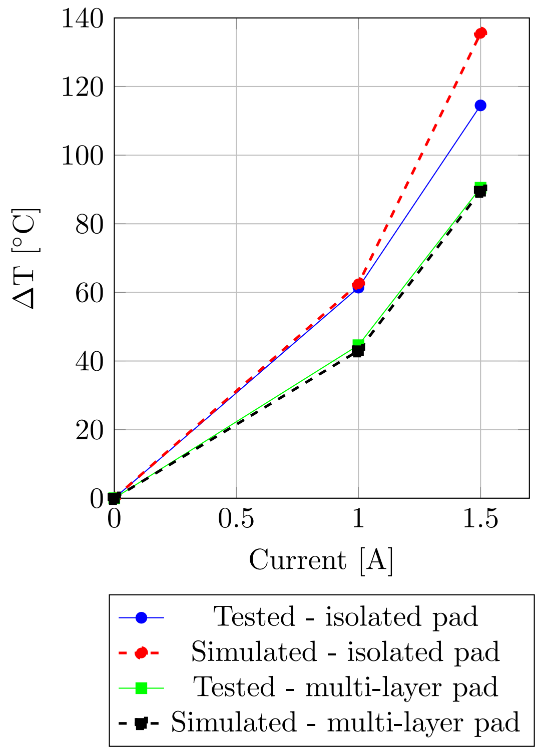
(a) Thermal coupling trace



(b) Over plane trace



(c) Inter-plane trace



(d) Component traces

Fig. 4.12: Correlation for various effects.

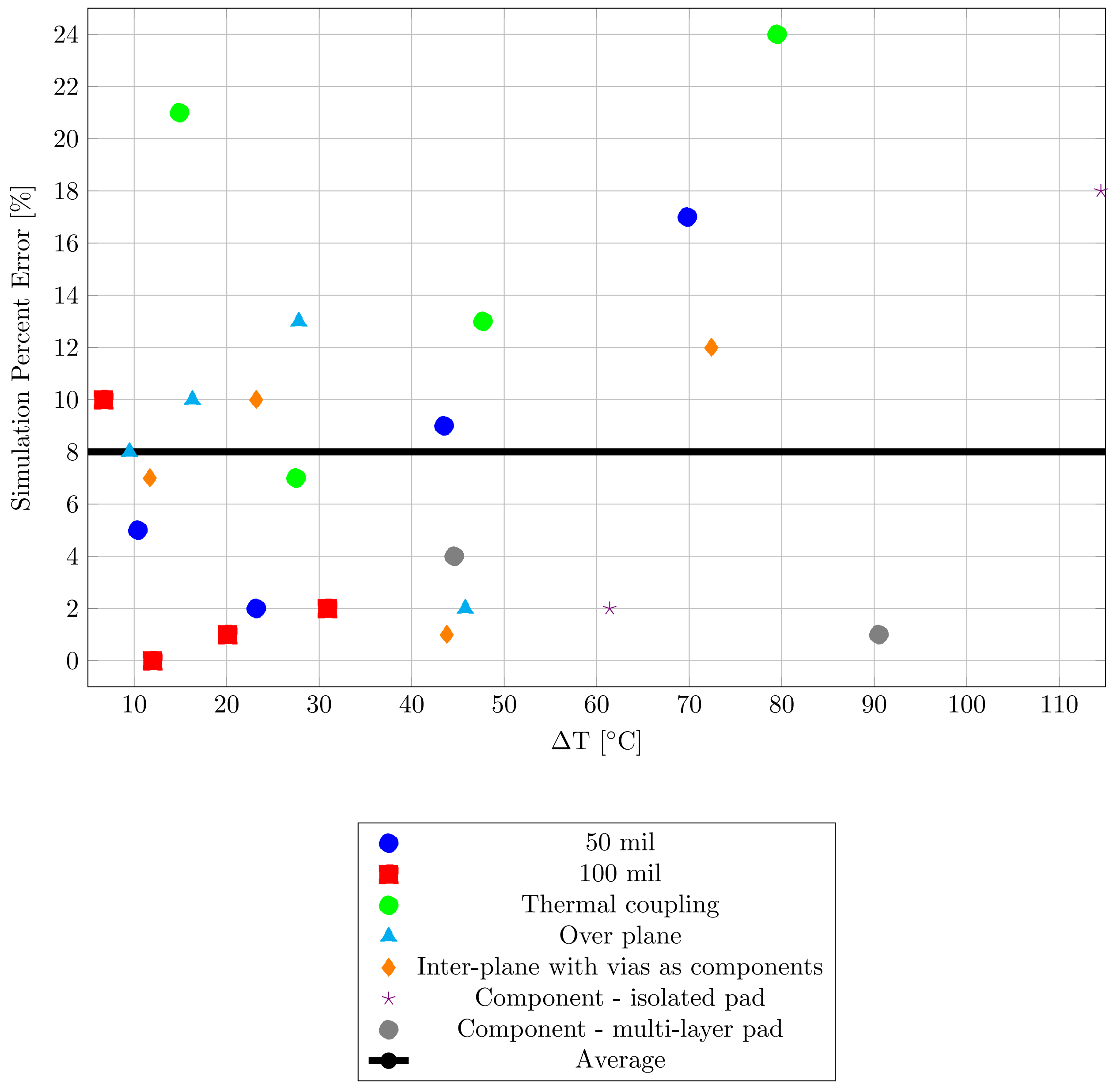


Fig. 4.13: Simulation results as percent error for each trace

CHAPTER 5
FUTURE WORK AND CONCLUSION

The algorithm and program described in chapter 3 can be a valuable resource for performing thermal analysis of a PCB design. However, there are additional features and functionality that could be developed given time and funding. These improvements are summarized in three area: GUI improvements, simulation tuning, and complexity of the heat transfer solution.

5.1 GUI Improvements

The GUI presented in this paper is adequate to demonstrate the methods developed in this research and facilitate quick analysis of thermal performance of a PCB design. There are still several aspects which could be improved and would be a great help to a broader number of users with varying experience in simulation and thermal analysis.

5.1.1 Program Feedback

For the steps mentioned in section 3.5, if the user has entered information that does not match between board setup and simulation setup, the simulation will fail to run without notification to the user. In order to prevent this, all of the conditions that one setting is dependant on another would need to be defined. Next, data validations for information entered into the GUI would need to be added. This would make sure that data entered in a field matches its needed data type such as integer, float, string, etc. Additionally, some method of messaging or communicating to the user the specific conflict would need to be added. Finally, anytime the program is busy in the background calculating a command from the user, some kind of indication would help the user understand that they need to wait for the process to finish before they can continue making modifications.

5.1.2 User Testing

The process defined in section 3.5 is what the author of this paper observed as being most straight forward. Performing user testing and seeing where untrained users make mistakes in entry or run into confusion on next steps would grant better insights into other improvements for the user interface.

5.2 Simulation Tuning

The tuning of the simulation to physical results proved to be time consuming and somewhat limited in their focus to the specific testing results mentioned in section 4.1. Several improvements could be made to improve both the accuracy of the simulation as well as user experience.

5.2.1 Additional Correction Factors

All of the simulation tuning factors described in section 4.2 were what was needed to tune the series of tests described in section 4.1. Other conditions or features not yet encountered could present additional needs of adjusting the simulation. In an effort to predict one such need, an additional tuning parameters for the thermal conductivity of the conductor material k were added, similar to the tuning factors for the thermal conductivity of the dielectric material. These tuning factors for the conductor material were not needed for tuning the test results mentioned in section 4.3.

5.2.2 Means of Tuning

Given that the means of tuning did follow a logical methodology, additional algorithms could be developed to perform most of this work automatically. To account for non-linear behavior in adjustment and tuning, machine learning could be implemented as well. This however would require a large amount of test data for the neural network to learn from. It is possible that the neural network could be trained using the curves presented in IPC 2152 rather than physical tests. Regardless of the method used to train the neural network, this could provide additional insights into the thermal performance of PCBs not covered by traditional analytical means.

5.3 Heat Transfer

The focus of this research was on providing a solution to steady state thermal performance of a PCB design given free convection conditions, DC loading, and no exterior means of heat dissipation. This presents several areas for possible improvements, many of which have their own cost of extra computation time.

5.3.1 Transient Solution and Loading

While the steady state thermal performance solution for electronic equipment, including PCBs, is adequate for many use cases, it may be desirable to understand the transient behavior of the PCB's thermal performance. Additional parameters and modifications to the algorithms mentions in chapters 2 and 3 would need to be made. Besides knowing the transient behavior of a PCB's thermal performance, many PCB's experience transient loading. This loading cycle or profile may be to a PCB's benefit (short periods of high loading followed by lower loads), or even potentially to its disadvantage. Having the ability to perform this analysis could prove beneficial to a PCB designer.

5.3.2 AC Current Loads

High frequency current loads present additional dynamics and constraints to how current flows in a conductor which in turn presents new types of losses. In order to perform this type of analysis, a more physics based approached to losses and potential gradients would need to be considered. This new approach could help provide a more complete analysis than what is presented in sections 3.3 and 3.3.2 - the downside likely being computation cost and time.

5.3.3 Forced Air Convection and Other Air Conditions

While it's useful to know the performance of a PCB in free convection conditions, most PCBs which will experience some kind of thermal performance strain will have some means of active cooling such as forced air. This would change the heat transfer coefficient mentioned in section 2.2.3 while the rest of the algorithm would remain the same. With that said, there are many factors into how well a forced air design contributes to the thermal performance of a PCB design such as if the air interfacing with the PCB is laminar or turbulent and if any impediments to the air flow exist on some portion of the PCB. Many of these concerns are best analyzed using computational fluid dynamics (CFD) software.

5.3.4 Heat Sinks

A design practice even more common than using forced air to assist the thermal performance of a PCB is the use of heat sinks on heat generating components. If this functionality were to be added to this body of research, it would likely be a factor of a component since that is the most common location for heat sink placement.

5.4 Summary of Contributions

The objectives of this research as outlined in chapter 1 are to first, develop a means of converting common PCB design files (gerber and drill files) into a mesh for finite element analysis (FEA), second, to find conduction losses in traces based on user input electric loads, third, complete a steady state heat transfer analysis of the PCB design, and finally, to present the input and output of this analysis in a graphic user interface (GUI). The material outlined in chapters 3 demonstrate the means to accomplish all four of these objectives. Chapter 4 demonstrates their effectiveness in producing a simulation that represents reality.

The use of this material, specifically the program and algorithm as provided in the appendix, can assist in better understanding the thermal performance of a PCB design. This method is especially useful in analyzing features in close proximity to each other as is common on most any PCB. Given the ever increasing power requirements on many PCB designs, this type of analysis can give valuable insights into a PCB's design.

REFERENCES

[1] G. N. Ellison, *Thermal Computations for Electronics Conductive, Radiative, and Convective Air Cooling.* 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742: CRC Press, 2020.

[2] ThoughtCo. (2017) Table of electrical resistivity and conductivity. [Online]. Available: https://www.thoughtco.com/table-of-electrical-resistivity-conductivity-608499

[3] PCBWay. (2017) 3 steps to determine / calculate number of pcb layers. [Online]. Available: https://www.pcbway.com/blog/Engineering_Technical/3_STEPS_How_to_determine_calculate_number_of_PCB_layers.html

[4] M. Marz, "Thermal management in high-density power converters," in *IEEE International Conference on Industrial Technology, 2003*, vol. 2, 2003, pp. 1196–1201 Vol.2.

[5] R. Bunea, N.-D. Codreanu, C. Ionescu, P. Svasta, and A. Vasile, "Pcb tracks thermal simulation, analysis and comparison to ipc-2152 for electrical current carrying capacity," in *3rd Electronics System Integration Technology Conference ESTC*, 2010, pp. 1–4.

[6] N. V. Vakrilov, D. V. Stoyanova, and N. M. Kafadarova, "Numerical thermal analysis of the pcb construction impact," in *2020 XI National Conference with International Participation (ELECTRONICA)*, 2020, pp. 1–5.

[7] A. F. Akawung and Y. Fujimoto, "Thermal analysis of air cooling system for electric machines using lumped parameter and flow resistance network," in *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*, 2021, pp. 1–6.

[8] P. von Böckh and T. Wetzel, *Heat Transfer Basics and Practice.* Heidelberg, Germany: Springer, 2012.

[9] Y. Shen, H. Wang, F. Blaabjerg, H. Zhao, and T. Long, "Thermal modeling and design optimization of pcb vias and pads," *IEEE Transactions on Power Electronics*, vol. 35, no. 1, pp. 882–900, 2020.

[10] G. Ellison, "Thermal analysis of circuit boards and microelectronic components using an analytical solution to the heat conduction equation," in *Twelfth Annual IEEE Semiconductor Thermal Measurement and Management Symposium. Proceedings*, 1996, pp. 144–150.

[11] D. G. Brooks and J. Adam, *PCB Trace and Via Currents and Temperatures: The Complete Analysis*, LaVergne, TN, 2017.

[12] Y. Shen, H. Zhao, T. Long, H. Wang, and F. Blaabjerg, "Two-dimensional thermal modeling and parametric optimization of printed circuit board vias," in *2019 IEEE Energy Conversion Congress and Exposition (ECCE)*, 2019, pp. 4931–4936.

[13] M. Honda, T. Hatakeyama, S. Nakagawa, R. Kibushi, and M. Ishizuka, “A thermal network model of a printed circuit board considering the thermal contraction flow caused by thermal vias,” in *2022 International Conference on Electronics Packaging (ICEP)*, 2022, pp. 109–110.

[14] J. Wei, T. Wan, X. Xue, and Y. Wang, “Approach towards accurate modeling of thermal resistance in thermal management of pcb,” in *2021 22nd International Conference on Electronic Packaging Technology (ICEPT)*, 2021, pp. 1–6.

[15] Red Blob Games. (2017) Circle fill on a grid. [Online]. Available: https://www.redblobgames.com/grids/circle-drawing/

APPENDICES

APPENDIX A
Heat Transfer Parameters

A.1 Material Properties

Table A.1: Thermal Conductivity for Common PCB Materials

Material	Conductivity $\left[\frac{W}{in^{\circ}C}\right]$
Copper	9.9
Gold	7.5
Aluminum	5.5
Nickel	2.3
Solder	1.46
FR-4 (in-plane)	0.02
FR-4 (through-plane)	0.007
Polyamide	0.005
Epoxy resin (unfilled)	0.004
Thermal compound	0.02

Table A.2: Electrical Resistivity for Common PCB Conductors [2]

Material	Resistivity $\left[\Omega - mil \times 10^{-4}\right]$
Copper	6.61
Gold	9.61
Aluminum	8.98
Nickel	27.5
Silver	6.26

APPENDIX B

Source Code

B.1 Repository and Project Notes

The following sections of source code can be found on GitHub at the following address: https://github.com/ziggy-usu/PCB-heat

The source files should be part of a single Python project and all additional packages should be installed prior to excecuting the project. The GUI can be accessed through excecuting *heat_gui.py*.

B.2 heat_gui.py

```
from tkinter import *
import customtkinter
import tkinter.filedialog as fd
from tkinter import ttk
import numpy as np
import json
import ast
import matplotlib

matplotlib.use('TkAgg')
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

import board_setup
```

```
customtkinter.set_appearance_mode("System") # Modes: "System" (standard), "Dark",
    "Light"
customtkinter.set_default_color_theme("dark-blue") # Themes: "blue" (standard), "
    green", "dark-blue"


class App(customtkinter.CTk):

    def __init__(self):
        super().__init__()

        win_wid = self.winfo_screenwidth()
        win_ht = self.winfo_screenheight()

        self.title("PCB Heat Analysis")
        self.geometry(f"{win_wid}x{win_ht}+0+0")
        self.process = -1

        self.protocol("WM_DELETE_WINDOW", self.on_closing) # call .on_closing()
            when app gets closed

        # ============ create two frames ============

        # configure grid layout (2x2)
        self.grid_columnconfigure(1, weight=1)
        self.grid_rowconfigure(1, weight=1)

        self.view_process()

        self.btn_board_setup.select()

        self.create_board_setup()
        self.create_sim_setup()
        self.create_results_frame()

        self.view_board_setup()
        self.view_board_settings()

    def view_process(self):
        self.frame_process = customtkinter.CTkFrame(master=self,
                                                     height=10)
        self.frame_process.grid(row=0, column=0, columnspan=2, sticky="nswe")

        # ========== frame_process ===========
        # configure grid layout (3x1)
        self.frame_process.rowconfigure(0, weight=1)
        self.frame_process.columnconfigure((0, 1, 2), weight=1)

        self.radio_var = IntVar(value=0)

        self.btn_board_setup = customtkinter.CTkRadioButton(master=self.
            frame_process,
                                                            variable=self.radio_var,
                                                            text='Board Setup',
                                                            value=0,
                                                            command=self.
                                                                view_board_setup)
        self.btn_board_setup.grid(row=2, column=0, pady=10, padx=20, sticky="nswe")

        self.btn_sim_setup = customtkinter.CTkRadioButton(master=self.frame_process
            ,
                                                          variable=self.radio_var,
                                                          text='Simulation Setup',
                                                          value=1,
                                                          command=self.view_sim_setup)
        self.btn_sim_setup.grid(row=2, column=1, pady=10, padx=20, sticky="nswe")
        self.btn_sim_setup.configure(state=DISABLED)

        self.btn_results = customtkinter.CTkRadioButton(master=self.frame_process,
                                                   variable=self.radio_var,
                                                   text='Results',
                                                   value=2,
                                                   command=self.view_results)
        self.btn_results.grid(row=2, column=2, pady=10, padx=20, sticky="nswe")
        self.btn_results.configure(state=DISABLED)

    def button_event(self):
        print("Button pressed")

    # board settings
    def create_board_setup(self):
        self.board_menu = customtkinter.CTkFrame(master=self,
                                             width=180,
                                             corner_radius=0)
        self.board_menu.grid_rowconfigure(0, minsize=10)
        self.board_menu.grid_rowconfigure((1, 2, 3), weight=1)

        self.btn_board_settings = customtkinter.CTkButton(master=self.board_menu,
                                                          text="Board Settings",
                                                          fg_color=("gray75", "gray30"),
                                                             # <- custom tuple-color
                                                          command=self.
                                                            view_board_settings)
        self.btn_board_settings.pack(pady=10)

        self.btn_layer_settings = customtkinter.CTkButton(master=self.board_menu,
                                                            text="Layer Settings",
                                                            fg_color=None, # <- custom
                                                               tuple-color
                                                            command=self.
                                                               view_layer_settings)
        self.btn_layer_settings.pack(pady=10)

        self.btn_board_component_settings = customtkinter.CTkButton(master=self.
            board_menu,
                                                           text="Board
                                                              Components",
                                                           fg_color=None, # <-
                                                              custom tuple-
                                                              color
                                                           command=self.
                                                              view_board_component_settings
                                                              )
        self.btn_board_component_settings.pack(pady=10)

        self.board_edit = customtkinter.CTkFrame(master=self)
        self.board_edit.grid_rowconfigure(1, weight=1)
        self.create_board_settings()
        self.create_edit_layer_settings()
        self.create_board_components_settings()
        self.create_board_save_load_frame()

    def create_board_settings(self):
        self.board_settings_frame = customtkinter.CTkFrame(master=self.board_edit)

        self.board_settings_frame.grid_columnconfigure(0, minsize=220)
        self.board_settings_frame.grid_columnconfigure(1, minsize=180)
        self.board_settings_frame.grid_columnconfigure(2, minsize=80)
        self.lbl_board_cond_material = customtkinter.CTkLabel(master=self.
            board_settings_frame,
                                                               text="Conductor Material
                                                                   :",
                                                               text_font=(
                                                                   "Roboto Medium", -16))
                                                                       # font name and
                                                                       size in px
        self.lbl_board_cond_material.grid(row=0, column=0, padx=5, pady=5, sticky="
            we")
        cond_material_options = [
            "Copper",
            "Aluminum",
            "Gold",
            "Silver"
        ]
        self.cond_material_clicked = StringVar()
        self.cond_material_clicked.set(cond_material_options[0])
        self.opt_board_cond_material = OptionMenu(self.board_settings_frame, self.
            cond_material_clicked,
                                                *cond_material_options, command=self.
                                                    update_board_settings)
        self.opt_board_cond_material.grid(row=0, column=1, pady=5, padx=5, sticky="
            we")

        self.lbl_board_diel_material = customtkinter.CTkLabel(master=self.
            board_settings_frame,
                                                               text="Dielectric Material
                                                                   :",
                                                               text_font=(
                                                        "Roboto Medium", -16))
                                                          # font name and
                                                          size in px
        self.lbl_board_diel_material.grid(row=1, column=0, padx=5, pady=5, sticky="
            we")
        diel_material_options = [
            "Fr-4"
        ]
        self.diel_material_clicked = StringVar()
        self.diel_material_clicked.set(diel_material_options[0])
        self.opt_board_diel_material = OptionMenu(self.board_settings_frame, self.
            diel_material_clicked,
                                                *diel_material_options, command=self.
                                                    update_board_settings)
        self.opt_board_diel_material.grid(row=1, column=1, pady=5, padx=5, sticky="
            we")

        self.lbl_board_plating_thickness = customtkinter.CTkLabel(master=self.
            board_settings_frame,
                                                        text="Plating Thickness
                                                             [oz]:",
                                                        text_font=(
                                                            "Roboto Medium",
                                                                -16)) # font
                                                                name and size
                                                                in px
        self.lbl_board_plating_thickness.grid(row=2, column=0, padx=5, pady=5,
            sticky="we")
        thickness_options = [
            '0.5',
            '1.0',
            '2.0'
```

```
]
self.plating_thickness_clicked = StringVar()
self.plating_thickness_clicked.set(thickness_options[0])
self.opt_board_plating_thickness = OptionMenu(self.board_settings_frame,
    self.plating_thickness_clicked,
                                          *thickness_options, command=self.
                                              update_board_settings)
self.opt_board_plating_thickness.grid(row=2, column=1, pady=5, padx=5,
    sticky="we")

self.lbl_board_drill_file = customtkinter.CTkLabel(master=self.
    board_settings_frame,
                                               text="Drill File:",
                                               text_font=(
                                                   "Roboto Medium", -16)) #
                                                       font name and size in
                                                       px
self.lbl_board_drill_file.grid(row=3, column=0, padx=5, pady=5, sticky="we
    ")
self.ent_drill_file_location = customtkinter.CTkEntry(master=self.
    board_settings_frame, width=120)
self.ent_drill_file_location.grid(row=3, column=1, padx=5, pady=5, sticky="
    we")
self.btn_browse_drill_file = customtkinter.CTkButton(master=self.
    board_settings_frame,
                                                 text="File...",
                                                 command=self.onDrillOpen)
self.btn_browse_drill_file.grid(row=3, column=2, pady=5, padx=5, sticky="we
    ")

self.lbl_board_keepout_file = customtkinter.CTkLabel(master=self.
    board_settings_frame,
                                                    text="Keepout File:",
                                                    text_font=(
                                                       "Roboto Medium", -16)) #
                                                           font name and size
                                                          in px
    self.lbl_board_keepout_file.grid(row=4, column=0, padx=5, pady=5, sticky="
        we")
    self.ent_keepout_file_location = customtkinter.CTkEntry(master=self.
        board_settings_frame, width=120)
    self.ent_keepout_file_location.grid(row=4, column=1, padx=5, pady=5, sticky
        ="we")
    self.btn_browse_keepout_file = customtkinter.CTkButton(master=self.
        board_settings_frame,
                                                      text="File...",
                                                      command=self.
                                                          onKeepOutOpen)
    self.btn_browse_keepout_file.grid(row=4, column=2, pady=5, padx=5, sticky="
        we")

def onDrillOpen(self):
    name = fd.askopenfilename(title="Drill File", filetypes=(("Drill files",
        "*.txt"), ("all files", "*.*")))
    self.ent_drill_file_location.insert(0, name)

def onKeepOutOpen(self):
    name = fd.askopenfilename(title="Keepout File", filetypes=(("Keepout files
        ", "*.GKO"), ("all files", "*.*")))
    self.ent_keepout_file_location.insert(0, name)

def create_edit_layer_settings(self):
    self.layer_settings_frame = customtkinter.CTkFrame(master=self.board_edit)
    self.layer_settings_frame.grid_columnconfigure(0, minsize=220)
    self.layer_settings_frame.grid_columnconfigure(1, minsize=180)
    self.layer_settings_frame.grid_columnconfigure(2, minsize=80)

    self.layer_tree = ttk.Treeview(master=self.layer_settings_frame)
    self.layer_tree['columns'] = ("Name", "Type", "Thickness", "Gerber File")
    self.layer_tree.column("#0", width=0, stretch=NO)
    self.layer_tree.column("Name", anchor=W, width=80)
    self.layer_tree.column("Type", anchor=W, width=50)
    self.layer_tree.column("Thickness", anchor=CENTER, width=45)
    self.layer_tree.column("Gerber File", anchor=E, width=260)
    self.layer_tree.heading("#0", text="", anchor=W)
    self.layer_tree.heading("Name", text="Name", anchor=W)
    self.layer_tree.heading("Type", text="Type", anchor=W)
    self.layer_tree.heading("Thickness", text="Thickness", anchor=W)
    self.layer_tree.heading("Gerber File", text="Gerber File", anchor=CENTER)
    self.layer_tree.bind('<ButtonRelease-1>', self.layer_tree_select_item)
    self.layer_tree.grid(row=0, column=0, columnspan=3, sticky="nswe", pady=5,
        padx=5)

    self.btn_layer_add = customtkinter.CTkButton(master=self.
        layer_settings_frame, text="Add",
                                                command=self.onLayerAdd)
    self.btn_layer_add.grid(row=1, column=0, pady=5, padx=5, sticky="we")

    self.btn_layer_remove = customtkinter.CTkButton(master=self.
        layer_settings_frame, text="Remove",
                                                   command=self.onLayerRemove)
    self.btn_layer_remove.grid(row=1, column=1, pady=5, padx=5, sticky="we")

    self.btn_layer_save = customtkinter.CTkButton(master=self.
        layer_settings_frame, text="Update",
                                              command=self.onLayerUpdate)
    self.btn_layer_save.grid(row=1, column=2, pady=5, padx=5, sticky="we")

    self.lbl_layer_edit = customtkinter.CTkLabel(master=self.
        layer_settings_frame,
                                             text="Layer Properties:",
                                             text_font=(
                                                "Roboto Medium", -16)) # font
                                                    name and size in px
    self.lbl_layer_edit.grid(row=3, column=0, columnspan=3, padx=5, pady=5,
        sticky="we")

    self.lbl_layer_name = customtkinter.CTkLabel(master=self.
        layer_settings_frame,
                                             text="Name:",
                                             text_font=(
                                                "Roboto Medium", -16)) # font
                                                    name and size in px
    self.lbl_layer_name.grid(row=4, column=0, padx=5, pady=5, sticky="we")
    self.ent_layer_name = customtkinter.CTkEntry(master=self.
        layer_settings_frame, width=120)
    self.ent_layer_name.grid(row=4, column=1, padx=5, pady=5, sticky="we")

    self.lbl_layer_type = customtkinter.CTkLabel(master=self.
        layer_settings_frame,
                                             text="Type:",
                                             text_font=(
                                                "Roboto Medium", -16)) # font
                                                    name and size in px
    self.lbl_layer_type.grid(row=5, column=0, padx=5, pady=5, sticky="we")
    layer_type_options = [
        "Conductor",
        "Dielectric"
    ]
    self.layer_type_clicked = StringVar()
    self.layer_type_clicked.set(layer_type_options[0])
    self.opt_layer_type = OptionMenu(self.layer_settings_frame, self.
        layer_type_clicked,
                                  *layer_type_options)
    self.opt_layer_type.grid(row=5, column=1, pady=5, padx=5, sticky="we")

    self.lbl_layer_thickness = customtkinter.CTkLabel(master=self.
        layer_settings_frame,
                                                 text="Thickness [mil]:",
                                                 text_font=(
                                                     "Roboto Medium", -16)) #
                                                         font name and size in
                                                         px
    self.lbl_layer_thickness.grid(row=6, column=0, padx=5, pady=5, sticky="we")
    self.ent_layer_thickness = customtkinter.CTkEntry(master=self.
        layer_settings_frame, width=120)
    self.ent_layer_thickness.grid(row=6, column=1, padx=5, pady=5, sticky="we")

    self.lbl_layer_gerber_file = customtkinter.CTkLabel(master=self.
        layer_settings_frame,
                                                   text="Gerber File:",
                                                   text_font=(
                                                       "Roboto Medium", -16)) #
                                                           font name and size
                                                          in px
    self.lbl_layer_gerber_file.grid(row=7, column=0, padx=5, pady=5, sticky="we
        ")
    self.ent_gerber_file_location = customtkinter.CTkEntry(master=self.
        layer_settings_frame, width=120)
    self.ent_gerber_file_location.grid(row=7, column=1, padx=5, pady=5, sticky
        ="we")
    self.btn_layer_gerber_file = customtkinter.CTkButton(master=self.
        layer_settings_frame, text="File...",
                                                  command=self.onGerberOpen)
    self.btn_layer_gerber_file.grid(row=7, column=2, pady=5, padx=5, sticky="we
        ")

def onGerberOpen(self):
    name = fd.askopenfilename(title="Gerber File", filetypes=(
        ("Gerber files", ".GTL G1 G2 G3 G4 G5 G6 G7 G8 G9 G10 GBL"), ("all
            files", "*.*")))
    self.ent_gerber_file_location.insert(0, name)

def onLayerAdd(self):
    self.layer_tree.insert(parent='', index='end', text='', values=(
        self.ent_layer_name.get(), self.layer_type_clicked.get(), self.
            ent_layer_thickness.get(),
        self.ent_gerber_file_location.get()))

    self.ent_layer_name.delete(0, END)
    self.ent_layer_thickness.delete(0, END)
    self.ent_gerber_file_location.delete(0, END)

def onLayerRemove(self):
    records = self.layer_tree.selection()
    if len(records) != 0:
        for record in records:
            self.layer_tree.delete(record)

def onLayerUpdate(self):
    selected_no = self.layer_tree.focus()
    if selected_no != '':
        self.layer_tree.item(selected_no, text='', values=(
            self.ent_layer_name.get(), self.layer_type_clicked.get(), self.
                ent_layer_thickness.get(),
            self.ent_gerber_file_location.get()))

def layer_tree_select_item(self, event):
    selected_no = self.layer_tree.focus()
    if selected_no != '':
        selected_vals = self.layer_tree.item(selected_no, 'values')

        self.ent_layer_name.delete(0, END)
        self.ent_layer_thickness.delete(0, END)
        self.ent_gerber_file_location.delete(0, END)

        self.ent_layer_name.insert(0, selected_vals[0])
        self.layer_type_clicked.set(selected_vals[1])
        self.ent_layer_thickness.insert(0, selected_vals[2])
        self.ent_gerber_file_location.insert(0, selected_vals[3])

def create_board_components_settings(self):
    self.board_components_frame = customtkinter.CTkFrame(master=self.board_edit
        )

    self.board_components_frame.grid_columnconfigure(0, minsize=220)
    self.board_components_frame.grid_columnconfigure(1, minsize=180)
    self.board_components_frame.grid_columnconfigure(2, minsize=80)

    self.board_components_tree = ttk.Treeview(master=self.
        board_components_frame)
    self.board_components_tree['columns'] = ("Name", "Side", "Width", "Length",
         "X Position", "Y Position")
```

```
self.board_components_tree.column("#0", width=0, stretch=NO)
self.board_components_tree.column("Name", anchor=W, width=80)
self.board_components_tree.column("Side", anchor=W, width=50)
self.board_components_tree.column("Width", anchor=CENTER, width=40)
self.board_components_tree.column("Length", anchor=CENTER, width=40)
self.board_components_tree.column("X Position", anchor=CENTER, width=60)
self.board_components_tree.column("Y Position", anchor=CENTER, width=60)
self.board_components_tree.heading("#0", text="", anchor=W)
self.board_components_tree.heading("Name", text="Name", anchor=W)
self.board_components_tree.heading("Side", text="Side", anchor=W)
self.board_components_tree.heading("Width", text="Width", anchor=W)
self.board_components_tree.heading("Length", text="Length", anchor=W)
self.board_components_tree.heading("X Position", text="X Position", anchor=
    CENTER)
self.board_components_tree.heading("Y Position", text="Y Position", anchor=
    CENTER)
self.board_components_tree.bind('<ButtonRelease-1>', self.
    board_components_select_item)
self.board_components_tree.grid(row=0, column=0, columnspan=3, sticky="nswe
    ", pady=5, padx=5)

self.btn_board_components_add = customtkinter.CTkButton(master=self.
    board_components_frame, text="Add",
                                                  command=self.
                                                      onBoardComponentAdd)
self.btn_board_components_add.grid(row=1, column=0, pady=5, padx=5, sticky
    ="we")

self.btn_board_components_remove = customtkinter.CTkButton(master=self.
    board_components_frame, text="Remove",
                                                       command=self.
                                                           onBoardComponentRemove
                                                           )
self.btn_board_components_remove.grid(row=1, column=1, pady=5, padx=5,
    sticky="we")

self.btn_board_components_save = customtkinter.CTkButton(master=self.
    board_components_frame, text="Update",
                                                     command=self.
                                                         onBoardComponentUpdate
                                                         )
self.btn_board_components_save.grid(row=1, column=2, pady=5, padx=5, sticky
    ="we")

self.lbl_board_components_edit = customtkinter.CTkLabel(master=self.
    board_components_frame,
                                                    text="Component
                                                        Properties:",
                                                    text_font=(
                                                        "Roboto Medium", -16)
                                                            ) # font name and
                                                             size in px
self.lbl_board_components_edit.grid(row=2, column=0, columnspan=3, padx=5,
    pady=5, sticky="we")

self.lbl_board_components_name = customtkinter.CTkLabel(master=self.
    board_components_frame,
                                                    text="Name:",
                                                    text_font=("Roboto
                                                        Medium", -16))
self.lbl_board_components_name.grid(row=3, column=0, padx=5, pady=5, sticky
    ="we")
```

```
self.ent_board_components_name = customtkinter.CTkEntry(master=self.
    board_components_frame, width=120)
self.ent_board_components_name.grid(row=3, column=1, padx=5, pady=5, sticky
    ="we")

self.lbl_board_component_side = customtkinter.CTkLabel(master=self.
    board_components_frame,
                                                     text="Board Side:",
                                                     text_font=(
                                                         "Roboto Medium", -16))
                                                              # font name and
                                                             size in px
self.lbl_board_component_side.grid(row=4, column=0, padx=5, pady=5, sticky
    ="we")
board_component_side_options = [
    "Top",
    "Bottom"
]
self.board_component_side_clicked = StringVar()
self.board_component_side_clicked.set(board_component_side_options[0])
self.opt_board_component_side = OptionMenu(self.board_components_frame,
    self.board_component_side_clicked,
                                        *board_component_side_options,
                                        command=self.
                                            update_board_component_settings)
self.opt_board_component_side.grid(row=4, column=1, pady=5, padx=5, sticky
    ="we")

self.lbl_board_components_width = customtkinter.CTkLabel(master=self.
    board_components_frame,
                                                       text="Width:",
                                                text_font=("Roboto
                                                    Medium", -16))
self.lbl_board_components_width.grid(row=5, column=0, padx=5, pady=5,
    sticky="we")

self.ent_board_components_width = customtkinter.CTkEntry(master=self.
    board_components_frame, width=120)
self.ent_board_components_width.grid(row=5, column=1, padx=5, pady=5,
    sticky="we")

self.lbl_board_components_length = customtkinter.CTkLabel(master=self.
    board_components_frame,
                                                 text="Length:",
                                                 text_font=("Roboto
                                                     Medium", -16))
self.lbl_board_components_length.grid(row=6, column=0, padx=5, pady=5,
    sticky="we")

self.ent_board_components_length = customtkinter.CTkEntry(master=self.
    board_components_frame, width=120)
self.ent_board_components_length.grid(row=6, column=1, padx=5, pady=5,
    sticky="we")

self.lbl_board_components_x_pos = customtkinter.CTkLabel(master=self.
    board_components_frame,
                                                text="X Position:",
                                                text_font=("Roboto
                                                    Medium", -16))
self.lbl_board_components_x_pos.grid(row=7, column=0, padx=5, pady=5,
    sticky="we")
    self.ent_board_components_x_pos = customtkinter.CTkEntry(master=self.
        board_components_frame, width=120)
    self.ent_board_components_x_pos.grid(row=7, column=1, padx=5, pady=5,
        sticky="we")

    self.lbl_board_components_y_pos = customtkinter.CTkLabel(master=self.
        board_components_frame,
                                                             text="Y Position:",
                                                             text_font=("Roboto
                                                                 Medium", -16))
    self.lbl_board_components_y_pos.grid(row=8, column=0, padx=5, pady=5,
        sticky="we")

    self.ent_board_components_y_pos = customtkinter.CTkEntry(master=self.
        board_components_frame, width=120)
    self.ent_board_components_y_pos.grid(row=8, column=1, padx=5, pady=5,
        sticky="we")

def onBoardComponentAdd(self):
    self.board_components_tree.insert(parent='', index='end', text='', values=(
        self.ent_board_components_name.get(), self.board_component_side_clicked
            .get(),
        self.ent_board_components_width.get(), self.ent_board_components_length
            .get(),
        self.ent_board_components_x_pos.get(), self.ent_board_components_y_pos.
            get()))

    self.ent_board_components_name.delete(0, END)
    self.ent_board_components_width.delete(0, END)
    self.ent_board_components_length.delete(0, END)
    self.ent_board_components_x_pos.delete(0, END)
    self.ent_board_components_y_pos.delete(0, END)
```

```
def onBoardComponentRemove(self):
    records = self.board_components_tree.selection()
    if len(records) != 0:
        for record in records:
            self.board_components_tree.delete(record)

def onBoardComponentUpdate(self):
    selected_no = self.board_components_tree.focus()
    if selected_no != '':
        self.board_components_tree.item(selected_no, text='', values=(
            self.ent_board_components_name.get(), self.
                board_component_side_clicked.get(),
            self.ent_board_components_width.get(), self.
                ent_board_components_length.get(),
            self.ent_board_components_x_pos.get(), self.
                ent_board_components_y_pos.get()))

def board_components_select_item(self, event):
    selected_no = self.board_components_tree.focus()
    if selected_no != '':
        selected_vals = self.board_components_tree.item(selected_no, 'values')

        self.ent_board_components_name.delete(0, END)
        self.ent_board_components_width.delete(0, END)
        self.ent_board_components_length.delete(0, END)
        self.ent_board_components_x_pos.delete(0, END)
        self.ent_board_components_y_pos.delete(0, END)

        self.ent_board_components_name.insert(0, selected_vals[0])
        self.board_component_side_clicked.set(selected_vals[1])
        self.ent_board_components_width.insert(0, selected_vals[2])
```

```
            self.ent_board_components_length.insert(0, selected_vals[3])
            self.ent_board_components_x_pos.insert(0, selected_vals[4])
            self.ent_board_components_y_pos.insert(0, selected_vals[5])

    def update_board_component_settings(self, event):
        pass

    def create_board_save_load_frame(self):
        self.board_save_load_frame = customtkinter.CTkFrame(master=self.board_edit)
        self.board_save_load_frame.grid(row=0, column=0, sticky="nswe", padx=5,
            pady=5)

        self.btn_board_load = customtkinter.CTkButton(master=self.
            board_save_load_frame,
                                                text="Load Board",
                                                fg_color=("gray75", "gray30"), #
                                                    <- custom tuple-color
                                                command=self.OnBoardLoad)
        self.btn_board_load.grid(row=0, column=0, sticky="we", padx=5, pady=5)

        self.btn_board_save = customtkinter.CTkButton(master=self.
            board_save_load_frame,
                                                text="Save Board",
                                                fg_color=("gray75", "gray30"), #
                                                    <- custom tuple-color
                                                command=self.OnBoardSave)
        self.btn_board_save.grid(row=0, column=1, sticky="we", padx=5, pady=5)

        self.btn_start_sim_setup = customtkinter.CTkButton(master=self.
            board_save_load_frame,
                                                     text="Setup Simulation",
                                                     fg_color=("gray75", "gray30")
                                                         , # <- custom tuple-color
                                                     command=self.OnStartSimSetup)
        self.btn_start_sim_setup.grid(row=0, column=2, sticky="we", padx=5, pady=5)

    def OnBoardLoad(self):
        board_load_name = fd.askopenfilename(title="Board Setup File",
                                        filetypes=(("Board Setup files", "*.pcb"),
                                            ("all files", "*.*")))

        with open(board_load_name, "r") as infile:
            board_setting_data = infile.read()
            board_settings = ast.literal_eval(json.loads(board_setting_data))

            self.cond_material_clicked.set(board_settings.get('conductor_material')
                )
            self.diel_material_clicked.set(board_settings.get('dielectric_material
                '))
            self.plating_thickness_clicked.set(board_settings.get('
                plating_thickness'))
            self.ent_drill_file_location.delete(0, END)
            self.ent_keepout_file_location.delete(0, END)
            self.ent_drill_file_location.insert(0, board_settings.get('drill_file')
                )
            self.ent_keepout_file_location.insert(0, board_settings.get('
                keepout_file'))

            # need to delete all from tree
            for record in self.layer_tree.get_children():
                self.layer_tree.delete(record)
            layers_setting = board_settings.get('layers')
            for layer_setting in layers_setting:
               if layer_setting.get('gerber_file') is None:
                   self.layer_tree.insert(parent='', index='end', text='', values=(
                       layer_setting.get('name'),
                       layer_setting.get('layer_type'),
                       layer_setting.get('thickness')
                   ))
               else:
                   self.layer_tree.insert(parent='', index='end', text='', values=(
                       layer_setting.get('name'),
                       layer_setting.get('layer_type'),
                       layer_setting.get('thickness'),
                       layer_setting.get('gerber_file')
                   ))

            # need to delete all from tree
            for record in self.board_components_tree.get_children():
               self.board_components_tree.delete(record)
            board_components_settings = board_settings.get('components')
            for board_component_setting in board_components_settings:
               self.board_components_tree.insert(parent='', index='end', text='',
                   values=(
                   board_component_setting.get('name'),
                   board_component_setting.get('side'),
                   board_component_setting.get('width'),
                   board_component_setting.get('length'),
                   board_component_setting.get('x_position'),
                   board_component_setting.get('y_position')
               ))

     def compile_board_settings(self):
            layers_setup = list()
            board_components_setup = list()
```

```
# Get layer settings
for layer_setting in self.layer_tree.get_children():
    selected_vals = self.layer_tree.item(layer_setting, 'values')
    this_layer_setup = board_setup.LayerSetup(selected_vals[0],
                                              selected_vals[1],
                                              selected_vals[2],
                                              selected_vals[3])
    layers_setup.append(this_layer_setup)

# Get component settings
for component_settings in self.board_components_tree.get_children():
    selected_vals = self.board_components_tree.item(component_settings, '
        values')
    this_component_setup = board_setup.ComponentSetup(selected_vals[0],
                                                      selected_vals[1],
                                                      selected_vals[2],
                                                      selected_vals[3],
                                                      selected_vals[4],
                                                      selected_vals[5])
    board_components_setup.append(this_component_setup)

# Board setup
board_settings = board_setup.BoardSetup(self.cond_material_clicked.get(),
                                        self.diel_material_clicked.get(),
                                        self.plating_thickness_clicked.get(),
                                        self.ent_drill_file_location.get(),
                                        self.ent_keepout_file_location.get(),
                                        layers_setup,
                                        board_components_setup)

return board_settings
```

```
def OnBoardSave(self):
    board_save_name = fd.asksaveasfilename(defaultextension='.pcb', filetypes
        =[("PCB files", '*.pcb')],
                                        title="Board Setup File")
    # need to take all the gui widgets, build setup classes and export into
        json
    if board_save_name != '':
        board_settings = self.compile_board_settings()

        with open(board_save_name, "w") as outfile:
            outfile.write(json.dumps(board_setup.BoardEncoder().encode(
                board_settings)))



def OnStartSimSetup(self):
    self.board_layer_names = list()

    for layer in self.layer_tree.get_children():
        layer_values = self.layer_tree.item(layer, 'values')
        self.board_layer_names.append(layer_values[0])

    self.board_component_names = list()
    for board_component in self.board_components_tree.get_children():
        component_values = self.board_components_tree.item(board_component, '
            values')
        self.board_component_names.append(component_values[0])

    self.create_sim_settings_frame()
    self.create_sim_tuning_frame()
    self.create_loads_settings_frame()
    self.create_components_settings_frame()
```

```
        self.view_sim_settings()

        self.btn_sim_setup.configure(state=NORMAL)
        self.btn_sim_setup.select()
        self.view_sim_setup()

    def update_board_settings(self, event):
        pass

    def view_board_setup(self):
        self.sim_menu.grid_remove()
        self.sim_edit.grid_remove()
        self.results_menu.grid_remove()
        self.results_edit.grid_remove()
        self.board_menu.grid(row=1, column=0, sticky="nswe")
        self.board_edit.grid(row=1, column=1, sticky="nswe", padx=20, pady=20)

    def view_board_settings(self):
        self.layer_settings_frame.grid_remove()
        self.board_components_frame.grid_remove()
        self.board_settings_frame.grid(row=1, column=0, sticky="nswe", padx=5, pady
            =5)

        self.btn_board_settings.configure(fg_color=("gray75", "gray30"))
        self.btn_layer_settings.configure(fg_color=None)
        self.btn_board_component_settings.configure(fg_color=None)

    def view_layer_settings(self):
        self.board_settings_frame.grid_remove()
        self.board_components_frame.grid_remove()
        self.layer_settings_frame.grid(row=1, column=0, sticky="nswe", padx=5, pady
            =5)
        self.btn_layer_settings.configure(fg_color=("gray75", "gray30"))
        self.btn_board_settings.configure(fg_color=None)
        self.btn_board_component_settings.configure(fg_color=None)

def view_board_component_settings(self):
        self.board_settings_frame.grid_remove()
        self.layer_settings_frame.grid_remove()
        self.board_components_frame.grid(row=1, column=0, sticky="nswe", padx=5,
            pady=5)

        self.btn_board_component_settings.configure(fg_color=("gray75", "gray30"))
        self.btn_layer_settings.configure(fg_color=None)
        self.btn_board_settings.configure(fg_color=None)

# simulation settings
def create_sim_setup(self):
        self.sim_menu = customtkinter.CTkFrame(master=self,
                                           width=180,
                                           corner_radius=0)

        self.sim_menu.grid_rowconfigure(0, minsize=10)
        self.sim_menu.grid_rowconfigure((1, 2), weight=1)

        self.btn_sim_settings = customtkinter.CTkButton(master=self.sim_menu,
                                                   text="Simulation Settings",
                                                   fg_color=("gray75", "gray30"), #
                                                        <- custom tuple-color
                                                   command=self.view_sim_settings)
        self.btn_sim_settings.pack(pady=10)

        self.btn_sim_tuning = customtkinter.CTkButton(master=self.sim_menu,
                                                   text="Tuning Settings",
                                                   fg_color=("gray75", "gray30"), #
                                                       <- custom tuple-color
                                                   command=self.view_sim_tuning)
    self.btn_sim_tuning.pack(pady=10)

    self.btn_loads_settings = customtkinter.CTkButton(master=self.sim_menu,
                                                     text="Load Settings",
                                                     fg_color=("gray75", "gray30"),
                                                          # <- custom tuple-color
                                                     command=self.
                                                         view_loads_settings)
    self.btn_loads_settings.pack(pady=10)

    self.btn_component_settings = customtkinter.CTkButton(master=self.sim_menu,
                                                         text="Component Settings",
                                                         fg_color=("gray75", "
                                                             gray30"), # <- custom
                                                             tuple-color
                                                         command=self.
                                                             view_components_settings
                                                             )
    self.btn_component_settings.pack(pady=10)

    self.sim_edit = customtkinter.CTkFrame(master=self)
    self.sim_edit.grid_rowconfigure(1, weight=1)
    self.sim_edit.grid_columnconfigure(0, weight=1)
    self.create_sim_save_load_frame()

def create_sim_settings_frame(self):
    self.sim_settings_frame = customtkinter.CTkFrame(master=self.sim_edit)
    self.sim_settings_frame.grid_columnconfigure(0, minsize=250)
    self.sim_settings_frame.grid_columnconfigure(1, minsize=180)

    self.lbl_sim_resolution = customtkinter.CTkLabel(master=self.
        sim_settings_frame,
                                                 text="Resolution [mil]:",
                                                 text_font=(
                                                       "Roboto Medium", -16)) #
                                                           font name and size in
                                                           px
    self.lbl_sim_resolution.grid(row=0, column=0, padx=5, pady=5, sticky="we")

    self.sim_res_tkr = Spinbox(master=self.sim_settings_frame, from_=1, to
        =1000)
    self.sim_res_tkr.grid(row=0, column=1, padx=5, pady=5, sticky="we")

    self.lbl_sim_ambient = customtkinter.CTkLabel(master=self.
        sim_settings_frame,
                                              text="Ambient Temp [C]:",
                                              text_font=(
                                                     "Roboto Medium", -16)) # font
                                                         name and size in px
    self.lbl_sim_ambient.grid(row=1, column=0, padx=5, pady=5, sticky="we")

    self.ent_sim_ambient = customtkinter.CTkEntry(master=self.
        sim_settings_frame, width=120)
    self.ent_sim_ambient.grid(row=1, column=1, padx=5, pady=5, sticky="we")

    self.lbl_sim_board_orient = customtkinter.CTkLabel(master=self.
        sim_settings_frame,
                                                   text="Board Orientation:",
                                                   text_font=(
                                                      "Roboto Medium", -16)) #
                                                         font name and size in
                                                         px
    self.lbl_sim_board_orient.grid(row=2, column=0, padx=5, pady=5, sticky="we
       ")

    sim_board_orient_options = [
       "X-axis, Y-axis Zero",
       "X-axis -90",
       "X-axis +90",
       "Y-axis -90",
       "Y-axis +90"
    ]
    self.sim_board_orient_clicked = StringVar()
    self.sim_board_orient_clicked.set(sim_board_orient_options[0])
    self.opt_sim_board_orient = OptionMenu(self.sim_settings_frame, self.
       sim_board_orient_clicked,
                                   *sim_board_orient_options, command=self.
                                      update_sim_settings)
    self.opt_sim_board_orient.grid(row=2, column=1, pady=5, padx=5, sticky="we
       ")

def create_sim_tuning_frame(self):
    self.sim_tuning_frame = customtkinter.CTkFrame(master=self.sim_edit)

    self.lbl_sim_cond_k_in_plane = customtkinter.CTkLabel(master=self.
       sim_tuning_frame,
                                          text="Conductor 'k', in-plane:",
                                          text_font=(
                                             "Roboto Medium", -16)) # font
                                                name and size in px
```

```
self.lbl_sim_cond_k_in_plane.grid(row=0, column=0, padx=5, pady=5, sticky="
    we")

self.ent_sim_cond_k_in_plane = customtkinter.CTkEntry(master=self.
    sim_tuning_frame, width=120)
self.ent_sim_cond_k_in_plane.grid(row=0, column=1, padx=5, pady=5, sticky="
    we")

self.lbl_sim_cond_k_thru_plane = customtkinter.CTkLabel(master=self.
    sim_tuning_frame,
                                                  text="Conductor 'k',
                                                      through-plane:",
                                                  text_font=(
                                                      "Roboto Medium", -16))
                                                          # font name and
                                                          size in px
self.lbl_sim_cond_k_thru_plane.grid(row=1, column=0, padx=5, pady=5, sticky
    ="we")

self.ent_sim_cond_k_thru_plane = customtkinter.CTkEntry(master=self.
    sim_tuning_frame, width=120)
self.ent_sim_cond_k_thru_plane.grid(row=1, column=1, padx=5, pady=5, sticky
    ="we")

self.lbl_sim_diel_k_in_plane = customtkinter.CTkLabel(master=self.
    sim_tuning_frame,
                                                  text="Dielectric 'k', in-
                                                      plane:",
                                                  text_font=(
                                                      "Roboto Medium", -16))
                                                          # font name and
                                                          size in px
```

```
self.lbl_sim_diel_k_in_plane.grid(row=2, column=0, padx=5, pady=5, sticky="
    we")

self.ent_sim_diel_k_in_plane = customtkinter.CTkEntry(master=self.
    sim_tuning_frame, width=120)
self.ent_sim_diel_k_in_plane.grid(row=2, column=1, padx=5, pady=5, sticky="
    we")

self.lbl_sim_diel_k_thru_plane = customtkinter.CTkLabel(master=self.
    sim_tuning_frame,
                                                    text="Dielectric 'k',
                                                        through-plane:",
                                                    text_font=(
                                                        "Roboto Medium", -16)
                                                            ) # font name and
                                                             size in px
self.lbl_sim_diel_k_thru_plane.grid(row=3, column=0, padx=5, pady=5, sticky
    ="we")

self.ent_sim_diel_k_thru_plane = customtkinter.CTkEntry(master=self.
    sim_tuning_frame, width=120)
self.ent_sim_diel_k_thru_plane.grid(row=3, column=1, padx=5, pady=5, sticky
    ="we")

self.lbl_sim_conv_coef = customtkinter.CTkLabel(master=self.
    sim_tuning_frame,
                                                  text="PCB Convection
                                                      Coefficient:",
                                                  text_font=(
                                                      "Roboto Medium", -16))
                                                          # font name and
                                                          size in px
```

```
self.lbl_sim_conv_coef.grid(row=4, column=0, padx=5, pady=5, sticky="we")

self.ent_sim_conv_coef = customtkinter.CTkEntry(master=self.
    sim_tuning_frame, width=120)
self.ent_sim_conv_coef.grid(row=4, column=1, padx=5, pady=5, sticky="we")

self.lbl_sim_rad_coef = customtkinter.CTkLabel(master=self.sim_tuning_frame
    ,
                                                   text="PCB Radiation
                                                       Coefficient:",
                                                   text_font=(
                                                       "Roboto Medium", -16)
                                                           ) # font name and
                                                            size in px
self.lbl_sim_rad_coef.grid(row=5, column=0, padx=5, pady=5, sticky="we")

self.ent_sim_rad_coef = customtkinter.CTkEntry(master=self.sim_tuning_frame
    , width=120)
self.ent_sim_rad_coef.grid(row=5, column=1, padx=5, pady=5, sticky="we")

self.lbl_sim_rad_pow = customtkinter.CTkLabel(master=self.sim_tuning_frame,
                                          text="PCB Radiation Power:",
                                          text_font=(
                                              "Roboto Medium", -16)) # font
                                                   name and size in px
self.lbl_sim_rad_pow.grid(row=6, column=0, padx=5, pady=5, sticky="we")

self.ent_sim_rad_pow = customtkinter.CTkEntry(master=self.sim_tuning_frame,
     width=120)
self.ent_sim_rad_pow.grid(row=6, column=1, padx=5, pady=5, sticky="we")
```

```
        self.lbl_sim_comp_htc = customtkinter.CTkLabel(master=self.sim_tuning_frame
            ,
                                                       text="Component HTC Coefficient
                                                           :",
                                                       text_font=(
                                                           "Roboto Medium", -16)) # font
                                                               name and size in px
        self.lbl_sim_comp_htc.grid(row=7, column=0, padx=5, pady=5, sticky="we")

        self.ent_sim_comp_htc = customtkinter.CTkEntry(master=self.sim_tuning_frame
            , width=120)
        self.ent_sim_comp_htc.grid(row=7, column=1, padx=5, pady=5, sticky="we")

    def create_loads_settings_frame(self):
        self.sim_loads_frame = customtkinter.CTkFrame(master=self.sim_edit)

        self.sim_loads_frame.grid_columnconfigure(0, minsize=250)
        self.sim_loads_frame.grid_columnconfigure(1, minsize=210)
        self.sim_loads_frame.grid_columnconfigure(2, minsize=80)

        self.loads_tree = ttk.Treeview(master=self.sim_loads_frame)
        self.loads_tree['columns'] = ("Name", "Layer", "Current", "X Start", "Y
            Start", "X End", "Y End")
        self.loads_tree.column("#0", width=0, stretch=NO)
        self.loads_tree.column("Name", anchor=W, width=80)
        self.loads_tree.column("Layer", anchor=W, width=60)
        self.loads_tree.column("Current", anchor=CENTER, width=40)
        self.loads_tree.column("X Start", anchor=CENTER, width=40)
        self.loads_tree.column("Y Start", anchor=CENTER, width=40)
        self.loads_tree.column("X End", anchor=CENTER, width=40)
        self.loads_tree.column("Y End", anchor=CENTER, width=40)
        self.loads_tree.heading("#0", text="", anchor=W)
```

```
self.loads_tree.heading("Name", text="Name", anchor=W)
self.loads_tree.heading("Layer", text="Layer", anchor=W)
self.loads_tree.heading("Current", text="Current", anchor=CENTER)
self.loads_tree.heading("X Start", text="X Start", anchor=CENTER)
self.loads_tree.heading("Y Start", text="Y Start", anchor=CENTER)
self.loads_tree.heading("X End", text="X End", anchor=CENTER)
self.loads_tree.heading("Y End", text="Y End", anchor=CENTER)
self.loads_tree.bind('<ButtonRelease-1>', self.loads_tree_select_item)
self.loads_tree.grid(row=0, column=0, columnspan=3, sticky="nswe", pady=5,
    padx=5)

self.btn_loads_add = customtkinter.CTkButton(master=self.sim_loads_frame,
    text="Add",
                                        command=self.onLoadAdd)
self.btn_loads_add.grid(row=1, column=0, pady=5, padx=5, sticky="we")

self.btn_loads_remove = customtkinter.CTkButton(master=self.sim_loads_frame
    , text="Remove",
                                           command=self.onLoadRemove)
self.btn_loads_remove.grid(row=1, column=1, pady=5, padx=5, sticky="we")

self.btn_load_save = customtkinter.CTkButton(master=self.sim_loads_frame,
    text="Update",
                                        command=self.onLoadUpdate)
self.btn_load_save.grid(row=1, column=2, pady=5, padx=5, sticky="we")

self.lbl_load_edit = customtkinter.CTkLabel(master=self.sim_loads_frame,
                                       text="Load Properties:",
                                       text_font=(
                                           "Roboto Medium", -16)) # font
                                               name and size in px
```

```
self.lbl_load_edit.grid(row=3, column=0, columnspan=3, padx=5, pady=5,
    sticky="we")

self.lbl_load_name = customtkinter.CTkLabel(master=self.sim_loads_frame,
                                             text="Name:",
                                             text_font=(
                                                 "Roboto Medium", -16)) # font
                                                     name and size in px
self.lbl_load_name.grid(row=4, column=0, padx=5, pady=5, sticky="we")
self.ent_load_name = customtkinter.CTkEntry(master=self.sim_loads_frame,
    width=120)
self.ent_load_name.grid(row=4, column=1, padx=5, pady=5, sticky="we")

self.lbl_load_layer = customtkinter.CTkLabel(master=self.sim_loads_frame,
                                              text="Layer:",
                                              text_font=(
                                                  "Roboto Medium", -16)) # font
                                                      name and size in px
self.lbl_load_layer.grid(row=5, column=0, padx=5, pady=5, sticky="we")
load_layer_options = self.board_layer_names
self.load_layer_clicked = StringVar()
self.load_layer_clicked.set(load_layer_options[0])
self.opt_load_layer = OptionMenu(self.sim_loads_frame, self.
    load_layer_clicked,
                              *load_layer_options,
                              command=self.update_board_component_settings)
self.opt_load_layer.grid(row=5, column=1, pady=5, padx=5, sticky="we")

self.lbl_load_current = customtkinter.CTkLabel(master=self.sim_loads_frame,
                                                text="Current:",
                                                text_font=(
                                                "Roboto Medium", -16)) # font
                                                    name and size in px
self.lbl_load_current.grid(row=6, column=0, padx=5, pady=5, sticky="we")

self.ent_load_current = customtkinter.CTkEntry(master=self.sim_loads_frame,
     width=120)
self.ent_load_current.grid(row=6, column=1, padx=5, pady=5, sticky="we")

self.lbl_load_x_start = customtkinter.CTkLabel(master=self.sim_loads_frame,
                                            text="X Start Location:",
                                            text_font=(
                                                "Roboto Medium", -16)) # font
                                                    name and size in px
self.lbl_load_x_start.grid(row=7, column=0, padx=5, pady=5, sticky="we")

self.ent_load_x_start = customtkinter.CTkEntry(master=self.sim_loads_frame,
     width=120)
self.ent_load_x_start.grid(row=7, column=1, padx=5, pady=5, sticky="we")

self.lbl_load_y_start = customtkinter.CTkLabel(master=self.sim_loads_frame,
                                            text="Y Start Location:",
                                            text_font=(
                                                "Roboto Medium", -16)) # font
                                                    name and size in px
self.lbl_load_y_start.grid(row=8, column=0, padx=5, pady=5, sticky="we")

self.ent_load_y_start = customtkinter.CTkEntry(master=self.sim_loads_frame,
     width=120)
self.ent_load_y_start.grid(row=8, column=1, padx=5, pady=5, sticky="we")

self.lbl_load_x_end = customtkinter.CTkLabel(master=self.sim_loads_frame,
                                          text="X End Location:",
                                            text_font=(
                                               "Roboto Medium", -16)) # font
                                                  name and size in px
    self.lbl_load_x_end.grid(row=9, column=0, padx=5, pady=5, sticky="we")

    self.ent_load_x_end = customtkinter.CTkEntry(master=self.sim_loads_frame,
        width=120)
    self.ent_load_x_end.grid(row=9, column=1, padx=5, pady=5, sticky="we")

    self.lbl_load_y_end = customtkinter.CTkLabel(master=self.sim_loads_frame,
                                            text="Y End Location:",
                                            text_font=(
                                               "Roboto Medium", -16)) # font
                                                  name and size in px
    self.lbl_load_y_end.grid(row=10, column=0, padx=5, pady=5, sticky="we")

    self.ent_load_y_end = customtkinter.CTkEntry(master=self.sim_loads_frame,
        width=120)
    self.ent_load_y_end.grid(row=10, column=1, padx=5, pady=5, sticky="we")

def onLoadAdd(self):
    self.loads_tree.insert(parent='', index='end', text='', values=(
        self.ent_load_name.get(), self.load_layer_clicked.get(), self.
            ent_load_current.get(),
        self.ent_load_x_start.get(), self.ent_load_y_start.get(),
        self.ent_load_x_end.get(), self.ent_load_y_end.get()))

    self.ent_load_name.delete(0, END)
    self.ent_load_current.delete(0, END)
    self.ent_load_x_start.delete(0, END)
    self.ent_load_y_start.delete(0, END)
    self.ent_load_x_end.delete(0, END)
    self.ent_load_y_end.delete(0, END)

def onLoadRemove(self):
    records = self.loads_tree.selection()
    if len(records) != 0:
        for record in records:
            self.loads_tree.delete(record)

def onLoadUpdate(self):
    selected_no = self.loads_tree.focus()
    if selected_no != '':
        self.loads_tree.item(selected_no, text='', values=(
            self.ent_load_name.get(), self.load_layer_clicked.get(), self.
                ent_load_current.get(),
            self.ent_load_x_start.get(), self.ent_load_y_start.get(),
            self.ent_load_x_end.get(), self.ent_load_y_end.get()))

def loads_tree_select_item(self, event):
    selected_no = self.loads_tree.focus()
    if selected_no != '':
        selected_vals = self.loads_tree.item(selected_no, 'values')

        self.ent_load_name.delete(0, END)
        self.ent_load_current.delete(0, END)
        self.ent_load_x_start.delete(0, END)
        self.ent_load_y_start.delete(0, END)
        self.ent_load_x_end.delete(0, END)
        self.ent_load_y_end.delete(0, END)

        self.ent_load_name.insert(0, selected_vals[0])
        self.load_layer_clicked.set(selected_vals[1])
        self.ent_load_current.insert(0, selected_vals[2])
```

```
            self.ent_load_x_start.insert(0, selected_vals[3])
            self.ent_load_y_start.insert(0, selected_vals[4])
            self.ent_load_x_end.insert(0, selected_vals[5])
            self.ent_load_y_end.insert(0, selected_vals[6])

    def create_components_settings_frame(self):
        self.sim_components_frame = customtkinter.CTkFrame(master=self.sim_edit)

        self.sim_components_frame.grid_columnconfigure(0, minsize=220)
        self.sim_components_frame.grid_columnconfigure(1, minsize=180)
        self.sim_components_frame.grid_columnconfigure(2, minsize=80)

        self.component_heat_tree = ttk.Treeview(master=self.sim_components_frame)
        self.component_heat_tree['columns'] = ("Component", "Heat")
        self.component_heat_tree.column("#0", width=0, stretch=NO)
        self.component_heat_tree.column("Component", anchor=W, width=80)
        self.component_heat_tree.column("Heat", anchor=CENTER, width=80)
        self.component_heat_tree.heading("Component", text="Component Name", anchor
            =W)
        self.component_heat_tree.heading("Heat", text="Heat [W]", anchor=CENTER)
        self.component_heat_tree.bind('<ButtonRelease-1>', self.
            component_heat_tree_select_item)
        self.component_heat_tree.grid(row=0, column=0, columnspan=3, sticky="nswe",
             pady=5, padx=5)

        self.btn_component_heat_add = customtkinter.CTkButton(master=self.
            sim_components_frame, text="Add",
                                                       command=self.
                                                           onComponentHeatAdd)
        self.btn_component_heat_add.grid(row=1, column=0, pady=5, padx=5, sticky="
            we")
```

```
self.btn_component_heat_remove = customtkinter.CTkButton(master=self.
    sim_components_frame, text="Remove",
                                                        command=self.
                                                            onComponentHeatRemove
                                                            )
self.btn_component_heat_remove.grid(row=1, column=1, pady=5, padx=5, sticky
    ="we")

self.btn_component_heat_save = customtkinter.CTkButton(master=self.
    sim_components_frame, text="Update",
                                                      command=self.
                                                          onComponentHeatUpdate)
self.btn_component_heat_save.grid(row=1, column=2, pady=5, padx=5, sticky="
    we")

self.lbl_component_heat_edit = customtkinter.CTkLabel(master=self.
    sim_components_frame,
                                                     text="Component Heat
                                                         Properties:",
                                                     text_font=(
                                                         "Roboto Medium", -16))
                                                             # font name and
                                                             size in px
self.lbl_component_heat_edit.grid(row=2, column=0, columnspan=3, padx=5,
    pady=5, sticky="we")

self.lbl_component_name = customtkinter.CTkLabel(master=self.
    sim_components_frame,
                                                text="Component:",
                                                text_font=(
                                                        "Roboto Medium", -16)) #
                                                            font name and size in
                                                            px
    self.lbl_component_name.grid(row=3, column=0, padx=5, pady=5, sticky="we")
    if len(self.board_component_names) > 0:
        component_name_options = self.board_component_names
    else:
        component_name_options = ["None"]
        self.btn_component_heat_add.configure(state=DISABLED)
        self.btn_component_heat_remove.configure(state=DISABLED)
        self.btn_component_heat_save.configure(state=DISABLED)
    self.component_name_clicked = StringVar()
    self.component_name_clicked.set(component_name_options[0])
    self.opt_component_name = OptionMenu(self.sim_components_frame, self.
        component_name_clicked,
                                        *component_name_options)
    self.opt_component_name.grid(row=3, column=1, pady=5, padx=5, sticky="we")

    self.lbl_component_heat = customtkinter.CTkLabel(master=self.
        sim_components_frame,
                                                    text="Heat [W]:",
                                                    text_font=(
                                                        "Roboto Medium", -16)) #
                                                            font name and size in
                                                            px
    self.lbl_component_heat.grid(row=4, column=0, padx=5, pady=5, sticky="we")
    self.ent_component_heat = customtkinter.CTkEntry(master=self.
        sim_components_frame, width=120)
    self.ent_component_heat.grid(row=4, column=1, padx=5, pady=5, sticky="we")

def onComponentHeatAdd(self):
    self.component_heat_tree.insert(parent='', index='end', text='', values=(
        self.component_name_clicked.get(), self.ent_component_heat.get()))

    self.ent_component_heat.delete(0, END)

def onComponentHeatRemove(self):
    records = self.component_heat_tree.selection()
    if len(records) != 0:
        for record in records:
            self.component_heat_tree.delete(record)

def onComponentHeatUpdate(self):
    selected_no = self.component_heat_tree.focus()
    if selected_no != '':
        self.component_heat_tree.item(selected_no, text='', values=(
            self.component_name_clicked.get(), self.ent_component_heat.get()))

def component_heat_tree_select_item(self, event):
    selected_no = self.component_heat_tree.focus()
    if selected_no != '':
        selected_vals = self.component_heat_tree.item(selected_no, 'values')

        self.ent_component_heat.delete(0, END)

        self.component_name_clicked.set(selected_vals[0])
        self.ent_component_heat.insert(0, selected_vals[1])

def create_sim_save_load_frame(self):
    self.sim_save_load_frame = customtkinter.CTkFrame(master=self.sim_edit)
    self.sim_save_load_frame.grid(row=0, column=0, sticky="nswe", padx=5, pady
        =5)
    self.btn_sim_load = customtkinter.CTkButton(master=self.sim_save_load_frame
        ,
                                             text="Load Simulation",
                                             fg_color=("gray75", "gray30"), # <-
                                                  custom tuple-color
                                             command=self.OnSimLoad)
    self.btn_sim_load.grid(row=0, column=0, sticky="nwse", padx=5, pady=5)

    self.btn_sim_save = customtkinter.CTkButton(master=self.sim_save_load_frame
        ,
                                             text="Save Simulation",
                                             fg_color=("gray75", "gray30"), # <-
                                                  custom tuple-color
                                             command=self.OnSimSave)
    self.btn_sim_save.grid(row=0, column=1, sticky="nwse", padx=5, pady=5)

    self.btn_sim_run = customtkinter.CTkButton(master=self.sim_save_load_frame,
                                            text="Run Simulation",
                                            fg_color=("gray75", "gray30"), # <-
                                                custom tuple-color
                                            command=self.OnSimRun)
    self.btn_sim_run.grid(row=0, column=2, sticky="nwse", padx=5, pady=5)

def OnSimLoad(self):
    sim_load_name = fd.askopenfilename(title="Simulation Setup File",
                                    filetypes=(("Simulation Setup files", "*.
                                        sim"), ("all files", "*.*")))

    with open(sim_load_name, "r") as infile:
        sim_setting_data = infile.read()
        sim_settings = ast.literal_eval(json.loads(sim_setting_data))
        tuning_settings = sim_settings.get('tuning')
```

```
self.sim_res_tkr.delete(0, END)
self.sim_res_tkr.insert(0, sim_settings.get('resolution'))
self.ent_sim_ambient.delete(0, END)
self.ent_sim_ambient.insert(0, sim_settings.get('ambient'))
self.sim_board_orient_clicked.set(sim_settings.get('orientation'))

self.ent_sim_cond_k_in_plane.delete(0, END)
self.ent_sim_cond_k_in_plane.insert(0, tuning_settings.get('
    cond_k_inplane'))
self.ent_sim_cond_k_thru_plane.delete(0, END)
self.ent_sim_cond_k_thru_plane.insert(0, tuning_settings.get('
    cond_k_thruplane'))
self.ent_sim_diel_k_in_plane.delete(0, END)
self.ent_sim_diel_k_in_plane.insert(0, tuning_settings.get('
    diel_k_inplane'))
self.ent_sim_diel_k_thru_plane.delete(0, END)
self.ent_sim_diel_k_thru_plane.insert(0, tuning_settings.get('
    diel_k_thruplane'))
self.ent_sim_conv_coef.delete(0, END)
self.ent_sim_conv_coef.insert(0, tuning_settings.get('conv_coef'))
self.ent_sim_rad_coef.delete(0, END)
self.ent_sim_rad_coef.insert(0, tuning_settings.get('rad_coef'))
self.ent_sim_rad_pow.delete(0, END)
self.ent_sim_rad_pow.insert(0, tuning_settings.get('rad_pow'))
self.ent_sim_comp_htc.delete(0, END)
self.ent_sim_comp_htc.insert(0, tuning_settings.get('component_htc'))

for record in self.loads_tree.get_children():
    self.loads_tree.delete(record)
loads_setting = sim_settings.get('loads')
for load_setting in loads_setting:
    self.loads_tree.insert(parent='', index='end', text='', values=(
                load_setting.get('name'),
                load_setting.get('layer'),
                load_setting.get('current'),
                load_setting.get('x_start'),
                load_setting.get('y_start'),
                load_setting.get('x_end'),
                load_setting.get('y_end')
            ))

        for record in self.component_heat_tree.get_children():
            self.component_heat_tree.delete(record)
        component_heats_settings = sim_settings.get('component_heats')
        for component_heat_setting in component_heats_settings:
            self.component_heat_tree.insert(parent='', index='end', text='',
                values=(
                component_heat_setting.get('component_name'),
                component_heat_setting.get('heat')
            ))

def compile_sim_settings(self):
    loads_setup = list()
    component_heats_setup = list()

    # Get load settings
    for load_setting in self.loads_tree.get_children():
        selected_vals = self.loads_tree.item(load_setting, 'values')
        this_load_setup = board_setup.LoadSetup(selected_vals[0],
                                                selected_vals[1],
                                                selected_vals[2],
                                                selected_vals[3],
                                                selected_vals[4],
                                                selected_vals[5],
                                                selected_vals[6])
        loads_setup.append(this_load_setup)

    # Get component heat settings
    for component_heat_settings in self.component_heat_tree.get_children():
        selected_vals = self.component_heat_tree.item(component_heat_settings,
            'values')
        this_component_heat_setup = board_setup.ComponentHeatSetup(
            selected_vals[0],
                                                                selected_vals[1])
        component_heats_setup.append(this_component_heat_setup)

    tuning_setup = board_setup.TuningSetup(self.ent_sim_cond_k_in_plane.get(),
                                           self.ent_sim_cond_k_thru_plane.get(),
                                           self.ent_sim_diel_k_in_plane.get(),
                                           self.ent_sim_diel_k_thru_plane.get(),
                                           self.ent_sim_conv_coef.get(),
                                           self.ent_sim_rad_coef.get(),
                                           self.ent_sim_rad_pow.get(),
                                           self.ent_sim_comp_htc.get())

    # Board setup
    sim_settings = board_setup.SimulationSetup(self.sim_res_tkr.get(),
                                               self.ent_sim_ambient.get(),
                                               self.sim_board_orient_clicked.get(),
                                               tuning_setup,
                                               loads_setup,
                                               component_heats_setup)

    return sim_settings

def OnSimSave(self):
    sim_save_name = fd.asksaveasfilename(defaultextension='.sim', filetypes=[("
        sim settings files", '*.sim')],
                                         title="Simulation Settings File")
    if sim_save_name != '':
        sim_settings = self.compile_sim_settings()

        with open(sim_save_name, "w") as outfile:
            outfile.write(json.dumps(board_setup.BoardEncoder().encode(
                sim_settings)))

def OnSimRun(self):
    board_settings = self.compile_board_settings()
    sim_settings = self.compile_sim_settings()

    self.heat_analysis = board_setup.run_simulation(board_settings,
        sim_settings)
    self.layer_cnt = len(self.heat_analysis.board.layers)

    self.create_result_figure_frame()

    self.btn_results.configure(state=NORMAL)
    self.btn_results.select()
    self.view_results()

def update_sim_settings(self, event=0):
    pass

def view_sim_setup(self):
    if self.btn_sim_setup.__getattribute__('state') == NORMAL:
        self.board_menu.grid_remove()
        self.board_edit.grid_remove()
        self.results_menu.grid_remove()
        self.results_edit.grid_remove()
        self.sim_menu.grid(row=1, column=0, sticky="nswe")
        self.sim_edit.grid(row=1, column=1, sticky="nswe", padx=20, pady=20)

def view_sim_settings(self):
    self.sim_settings_frame.grid(row=1, column=0, sticky="nswe", padx=5, pady
        =5)
    self.sim_tuning_frame.grid_remove()
    self.sim_loads_frame.grid_remove()
    self.sim_components_frame.grid_remove()

    self.btn_sim_settings.configure(fg_color=("gray75", "gray30"))
    self.btn_sim_tuning.configure(fg_color=None)
    self.btn_loads_settings.configure(fg_color=None)
    self.btn_component_settings.configure(fg_color=None)

def view_sim_tuning(self):
    self.sim_settings_frame.grid_remove()
    self.sim_tuning_frame.grid(row=1, column=0, sticky="nswe", padx=5, pady=5)
    self.sim_loads_frame.grid_remove()
    self.sim_components_frame.grid_remove()

    self.btn_sim_settings.configure(fg_color=None)
    self.btn_sim_tuning.configure(fg_color=("gray75", "gray30"))
    self.btn_loads_settings.configure(fg_color=None)
    self.btn_component_settings.configure(fg_color=None)

def view_loads_settings(self):
    self.sim_settings_frame.grid_remove()
    self.sim_tuning_frame.grid_remove()
    self.sim_loads_frame.grid(row=1, column=0, sticky="nswe", padx=5, pady=5)
    self.sim_components_frame.grid_remove()
    self.btn_sim_settings.configure(fg_color=None)
    self.btn_sim_tuning.configure(fg_color=None)
    self.btn_loads_settings.configure(fg_color=("gray75", "gray30"))
    self.btn_component_settings.configure(fg_color=None)

def view_components_settings(self):
    self.sim_settings_frame.grid_remove()
    self.sim_tuning_frame.grid_remove()
    self.sim_loads_frame.grid_remove()
    self.sim_components_frame.grid(row=1, column=0, sticky="nswe", padx=5, pady
        =5)

    self.btn_sim_settings.configure(fg_color=None)
    self.btn_sim_tuning.configure(fg_color=None)
    self.btn_loads_settings.configure(fg_color=None)
    self.btn_component_settings.configure(fg_color=("gray75", "gray30"))

# results settings
def create_results_frame(self):
    self.results_menu = customtkinter.CTkFrame(master=self,
                                               width=180,
                                               corner_radius=0)

    self.results_menu.grid_rowconfigure(0, minsize=10)
    self.results_menu.grid_rowconfigure((1, 2, 3, 4), weight=1)

    self.btn_results_conductor = customtkinter.CTkButton(master=self.
        results_menu,
                                                         text="Conductor Traces",
                                                fg_color=("gray75", "gray30
                                                    "), # <- custom tuple-
                                                    color
                                                command=self.
                                                    OnResultConductor)
    self.btn_results_conductor.pack(pady=10)

    self.btn_results_losses = customtkinter.CTkButton(master=self.results_menu,
                                                 text="Conduction Losses",
                                                 fg_color=("gray75", "gray30"),
                                                     # <- custom tuple-color
                                                 command=self.OnResultLosses)
    self.btn_results_losses.pack(pady=10)

    self.btn_results_temperature = customtkinter.CTkButton(master=self.
        results_menu,
                                                      text="Temperature",
                                                      fg_color=("gray75", "
                                                          gray30"), # <- custom
                                                          tuple-color
                                                      command=self.
                                                          OnResultsTemperature)
    self.btn_results_temperature.pack(pady=10)
    self.results_edit = customtkinter.CTkFrame(master=self)
    self.results_edit.grid_rowconfigure(0, weight=1)
    self.results_edit.grid_columnconfigure(0, weight=1)

def create_result_figure_frame(self):
    self.results_figure_frame = customtkinter.CTkFrame(master=self.results_edit
        )
    self.results_figure_frame.grid(row=0, column=0, padx=5, pady=5, sticky="
        nwse")
    self.results_figure_frame.rowconfigure(0, weight=1)
    self.results_figure_frame.columnconfigure(0, weight=1)

    self.result_layer_text = "Layer: "
    self.lbl_results_layer = customtkinter.CTkLabel(master=self.results_edit,
                                                    text=self.result_layer_text,
                                                    text_font=(
                                                        "Roboto Medium", -16)) #
                                                            font name and size in px
    self.lbl_results_layer.grid(row=1, column=0, padx=5, pady=5, sticky="we")

    self.slider_layer = customtkinter.CTkSlider(master=self.results_edit,
                                                from_=0,
                                                to=self.layer_cnt - 1,
                                                number_of_steps=self.layer_cnt - 1,
                                                command=self.results_plot_update)
    self.results_state = 2
    self.slider_layer.set(0)
    self.slider_layer.grid(row=2, column=0, pady=10, padx=20, sticky="we")

    self.OnResultsTemperature()
    self.results_plot_update()

def results_plot_update(self, event=None):
    which_layer = int(self.slider_layer.get())
    layer_name = self.heat_analysis.board.layers[which_layer].name

    self.lbl_results_layer.configure(text='Layer: ' + layer_name)

    self.result_figure = Figure()

    ax = self.result_figure.add_subplot()
```

```
axes = [
    np.asarray(range(0,
                    self.heat_analysis.board.simulation.resolution * self.
                        heat_analysis.board.mat_ht + 1,
                    self.heat_analysis.board.simulation.resolution)),
    np.asarray(range(0,
                    self.heat_analysis.board.simulation.resolution * self.
                        heat_analysis.board.mat_wid + 1,
                    self.heat_analysis.board.simulation.resolution))]
if self.results_state == 0:
    plot = ax.pcolormesh(axes[1], axes[0],
                        np.transpose(np.where(self.heat_analysis.board.
                            layers[which_layer].cond_mat > 0, 1, 0)),
                        cmap='summer')
elif self.results_state == 1:
    plot = ax.pcolormesh(axes[1], axes[0], np.transpose(self.heat_analysis.
        board.layers[which_layer].Q_mat), cmap='Wistia')
elif self.results_state == 2:
    c_map_colors = ["darkblue", "dodgerblue", "yellowgreen", "orange", "red
        ", "salmon", "mistyrose"]
    heat_cmap = matplotlib.colors.LinearSegmentedColormap.from_list("
        heatcmap", c_map_colors)

    plot = ax.pcolormesh(axes[1], axes[0], np.rot90(self.heat_analysis.
        temp_mat[which_layer]), cmap=heat_cmap)
    self.result_figure.colorbar(plot)
ax.set_aspect('equal')
ax.set_xlabel('X Position [mil]')
ax.set_ylabel('Y Position [mil]')

figure_canvas = FigureCanvasTkAgg(self.result_figure, master=self.
    results_figure_frame)
```

```
        self.results_figure_frame.bind("<Configure>", self.resize_window)

        figure_canvas.get_tk_widget().grid(row=0, column=0, padx=5, pady=5, sticky
            ="nsew")

    def resize_window(self, event=None):
        win_wid = self.results_figure_frame.winfo_width()
        win_ht = self.results_edit.winfo_height()
        self.result_figure.set_size_inches(win_wid, win_ht)

    def OnResultConductor(self):
        self.btn_results_conductor.configure(fg_color=("gray75", "gray30"))
        self.btn_results_losses.configure(fg_color=None)
        self.btn_results_temperature.configure(fg_color=None)
        self.results_state = 0
        self.results_plot_update()

    def OnResultLosses(self):
        self.btn_results_conductor.configure(fg_color=None)
        self.btn_results_losses.configure(fg_color=("gray75", "gray30"))
        self.btn_results_temperature.configure(fg_color=None)
        self.results_state = 1
        self.results_plot_update()

    def OnResultsTemperature(self):
        self.btn_results_conductor.configure(fg_color=None)
        self.btn_results_losses.configure(fg_color=None)
        self.btn_results_temperature.configure(fg_color=("gray75", "gray30"))
        self.results_state = 2
        self.results_plot_update()

    def view_results(self):
        self.board_menu.grid_remove()
        self.board_edit.grid_remove()
        self.sim_menu.grid_remove()
        self.sim_edit.grid_remove()
        self.results_menu.grid(row=1, column=0, sticky="nswe")
        self.results_edit.grid(row=1, column=1, sticky="nswe", padx=20, pady=20)

    def on_closing(self, event=0):
        self.destroy()

    def start(self):
        self.mainloop()



if __name__ == "__main__":
    app = App()
    app.start()
```

B.3 board_setup.py

```
from json import JSONEncoder

import Helpers
import current_tracing
import drill
import heat_transfer
import layer
import pcb_board
import tracer
```

```
class BoardEncoder(JSONEncoder):
    def default(self, o):
        return o.__dict__



class BoardSetup:
    def __init__(self, conductor_material, dielectric_material, plating_thickness,
         drill_file, keepout_file, layers,
                components):
        self.conductor_material = conductor_material
        self.dielectric_material = dielectric_material
        self.plating_thickness = plating_thickness
        self.drill_file = drill_file
        self.keepout_file = keepout_file
        self.layers = layers
        self.components = components



class LayerSetup:
    def __init__(self, name, layer_type, thickness, gerber_file):
        self.name = name
        self.layer_type = layer_type
        self.thickness = thickness
        self.gerber_file = gerber_file



class ComponentSetup:
    def __init__(self, name, side, width, length, x_position, y_position):
        self.name = name
        self.side = side
        self.width = width
        self.length = length
        self.x_position = x_position
        self.y_position = y_position


class SimulationSetup:
    def __init__(self, resolution, ambient, orientation, tuning, loads,
        component_heats):
        self.resolution = resolution
        self.ambient = ambient
        self.orientation = orientation
        self.tuning = tuning
        self.loads = loads
        self.component_heats = component_heats


class TuningSetup:
    def __init__(self, cond_k_inplane, cond_k_thruplane, diel_k_inplane,
        diel_k_thruplane, conv_coef, rad_coef, rad_pow,
                component_htc):
        self.cond_k_inplane = cond_k_inplane
        self.cond_k_thruplane = cond_k_thruplane
        self.diel_k_inplane = diel_k_inplane
        self.diel_k_thruplane = diel_k_thruplane
        self.conv_coef = conv_coef
        self.rad_coef = rad_coef
        self.rad_pow = rad_pow
        self.component_htc = component_htc

class LoadSetup:
    def __init__(self, name, layer, current, x_start, y_start, x_end, y_end):
        self.name = name
        self.layer = layer
        self.current = current
        self.x_start = x_start
        self.y_start = y_start
        self.x_end = x_end
        self.y_end = y_end


class ComponentHeatSetup:
    def __init__(self, component_name, heat):
        self.component_name = component_name
        self.heat = heat


def run_simulation(board_settings, sim_settings):
    this_sim_orientation = [0, 0]
    if sim_settings.orientation == "X-axis -90":
        this_sim_orientation = [-1, 0]
    elif sim_settings.orientation == "X-axis +90":
        this_sim_orientation = [1, 0]
    elif sim_settings.orientation == "Y-axis -90":
        this_sim_orientation = [0, -1]
    elif sim_settings.orientation == "Y-axis +90":
        this_sim_orientation = [0, 1]

    simulation = tracer.Simulation(int(sim_settings.resolution), float(
        sim_settings.ambient), this_sim_orientation,
                                   True,
                                   float(sim_settings.tuning.cond_k_inplane),
                                   float(sim_settings.tuning.cond_k_thruplane),
                                   float(sim_settings.tuning.diel_k_inplane),
                                   float(sim_settings.tuning.diel_k_thruplane),
                                   float(sim_settings.tuning.conv_coef),
                                   float(sim_settings.tuning.rad_coef),
                                   float(sim_settings.tuning.rad_pow),
                                   float(sim_settings.tuning.component_htc))
    keepOutGerber = board_settings.keepout_file
    keepOutLines = Helpers.load_Gerber(keepOutGerber)
    board_dims = layer.get_board_dims(keepOutLines)

    drillFile = board_settings.drill_file
    platingThickness = float(board_settings.plating_thickness)
    platingThicknessUnit = 'oz'
    if drillFile != '':
        drillLines = Helpers.load_Gerber(drillFile)
        drillLayer = drill.Drill(drillLines, board_dims, simulation.resolution,
            platingThickness,
                              platingThicknessUnit)
        drillLayer.trace_drill()

    conductorMaterial = board_settings.conductor_material
    dielectricMaterial = board_settings.dielectric_material
    platingMaterial = conductorMaterial

    board_components = list()
    for component_heat in sim_settings.component_heats:
        for board_component in board_settings.components:
            if board_component.name == component_heat.component_name:
                this_board_component = tracer.Component(board_component.name,
                                            [float(board_component.width), float(
                                                board_component.length)],
                                            [float(board_component.x_position),
                                                float(board_component.y_position)],
                                            float(component_heat.heat),
                                                board_component.side)
                board_components.append(this_board_component)
                break

    layer_list = []
    for layer_setting in board_settings.layers:
        thisLayerName = layer_setting.name
        thisLayerType = layer_setting.layer_type
        thisLayerGerber = layer_setting.gerber_file
        if thisLayerGerber != '':
            thisLayerLines = Helpers.load_Gerber(thisLayerGerber)
        thisLayerThickness = float(layer_setting.thickness)
        thisLayerThicknessUnit = 'mil'

        electric_loads_for_layer = list()
        for load_setting in sim_settings.loads:
            if load_setting.layer == layer_setting.name:
                thisLoadThisLayerCurrent = float(load_setting.current)
                thisLoadThisLayerStart = [float(load_setting.x_start), float(
                    load_setting.y_start)]
                thisLoadThisLayerEnd = [float(load_setting.x_end), float(
                    load_setting.y_end)]
                thisLoadThisLayer = current_tracing.ElectricLoad(load_setting.name,
                    thisLoadThisLayerCurrent,
                                                            thisLoadThisLayerStart,
                                                                thisLoadThisLayerEnd
                                                                )
                electric_loads_for_layer.append(thisLoadThisLayer)

        thisLayer = layer.Layer(thisLayerName, thisLayerType, thisLayerLines,
            conductorMaterial,
                            thisLayerThickness, thisLayerThicknessUnit,
                                board_dims, simulation,
                            electric_loads_for_layer)

        if not isinstance(thisLayerGerber, type(None)) and thisLayer.layer_type ==
            "Conductor":
            if simulation.show_process:
                print("Tracing layer: " + layer_setting.name)
            thisLayer.trace_layer()

            if simulation.show_process:
                print("Finding networks: " + layer_setting.name)
            thisLayer.find_networks()

            if simulation.show_process:
                print("Calculating conduction losses: " + layer_setting.name)
            thisLayer.find_cond_loss()

        # this happens to both conductor and insulating layers
        if not isinstance(drillFile, type(None)):
            if simulation.show_process:
                print("Drilling holes: " + layer_setting.name)
            thisLayer.drill_holes(drillLayer)

        layer_list.append(thisLayer)

    board = pcb_board.Board(layer_list, board_components, simulation,
        conductorMaterial, dielectricMaterial)

    heat_transfer_analysis = heat_transfer.Simultaneous(board)
    heat_transfer_analysis.solve()

    return heat_transfer_analysis
```

B.4 Helpers.py

```
from json import JSONEncoder


import Helpers
import current_tracing
import drill
import heat_transfer
import layer
import pcb_board
import tracer



class BoardEncoder(JSONEncoder):
    def default(self, o):
        return o.__dict__



class BoardSetup:
    def __init__(self, conductor_material, dielectric_material, plating_thickness,
         drill_file, keepout_file, layers,
                components):
        self.conductor_material = conductor_material
        self.dielectric_material = dielectric_material
        self.plating_thickness = plating_thickness
        self.drill_file = drill_file
        self.keepout_file = keepout_file
        self.layers = layers
        self.components = components



class LayerSetup:
    def __init__(self, name, layer_type, thickness, gerber_file):
        self.name = name
        self.layer_type = layer_type
        self.thickness = thickness
        self.gerber_file = gerber_file


class ComponentSetup:
    def __init__(self, name, side, width, length, x_position, y_position):
        self.name = name
        self.side = side
        self.width = width
        self.length = length
        self.x_position = x_position
        self.y_position = y_position


class SimulationSetup:
    def __init__(self, resolution, ambient, orientation, tuning, loads,
        component_heats):
        self.resolution = resolution
        self.ambient = ambient
        self.orientation = orientation
        self.tuning = tuning
        self.loads = loads
        self.component_heats = component_heats


class TuningSetup:
    def __init__(self, cond_k_inplane, cond_k_thruplane, diel_k_inplane,
        diel_k_thruplane, conv_coef, rad_coef, rad_pow,
                component_htc):
```

```
        self.cond_k_inplane = cond_k_inplane
        self.cond_k_thruplane = cond_k_thruplane
        self.diel_k_inplane = diel_k_inplane
        self.diel_k_thruplane = diel_k_thruplane
        self.conv_coef = conv_coef
        self.rad_coef = rad_coef
        self.rad_pow = rad_pow
        self.component_htc = component_htc

class LoadSetup:
    def __init__(self, name, layer, current, x_start, y_start, x_end, y_end):
        self.name = name
        self.layer = layer
        self.current = current
        self.x_start = x_start
        self.y_start = y_start
        self.x_end = x_end
        self.y_end = y_end


class ComponentHeatSetup:
    def __init__(self, component_name, heat):
        self.component_name = component_name
        self.heat = heat



def run_simulation(board_settings, sim_settings):
    this_sim_orientation = [0, 0]
    if sim_settings.orientation == "X-axis -90":
        this_sim_orientation = [-1, 0]
    elif sim_settings.orientation == "X-axis +90":
        this_sim_orientation = [1, 0]
```

```
elif sim_settings.orientation == "Y-axis -90":
    this_sim_orientation = [0, -1]
elif sim_settings.orientation == "Y-axis +90":
    this_sim_orientation = [0, 1]

simulation = tracer.Simulation(int(sim_settings.resolution), float(
    sim_settings.ambient), this_sim_orientation,
                              True,
                              float(sim_settings.tuning.cond_k_inplane),
                              float(sim_settings.tuning.cond_k_thruplane),
                              float(sim_settings.tuning.diel_k_inplane),
                              float(sim_settings.tuning.diel_k_thruplane),
                              float(sim_settings.tuning.conv_coef),
                              float(sim_settings.tuning.rad_coef),
                              float(sim_settings.tuning.rad_pow),
                              float(sim_settings.tuning.component_htc))
keepOutGerber = board_settings.keepout_file
keepOutLines = Helpers.load_Gerber(keepOutGerber)
board_dims = layer.get_board_dims(keepOutLines)

drillFile = board_settings.drill_file
platingThickness = float(board_settings.plating_thickness)
platingThicknessUnit = 'oz'
if drillFile != '':
    drillLines = Helpers.load_Gerber(drillFile)
    drillLayer = drill.Drill(drillLines, board_dims, simulation.resolution,
        platingThickness,
                            platingThicknessUnit)
    drillLayer.trace_drill()

conductorMaterial = board_settings.conductor_material
dielectricMaterial = board_settings.dielectric_material
```

```
platingMaterial = conductorMaterial

board_components = list()
for component_heat in sim_settings.component_heats:
    for board_component in board_settings.components:
        if board_component.name == component_heat.component_name:
            this_board_component = tracer.Component(board_component.name,
                                            [float(board_component.width), float(
                                                board_component.length)],
                                            [float(board_component.x_position),
                                                float(board_component.y_position)],
                                            float(component_heat.heat),
                                                board_component.side)
            board_components.append(this_board_component)
            break

layer_list = []
for layer_setting in board_settings.layers:
    thisLayerName = layer_setting.name
    thisLayerType = layer_setting.layer_type
    thisLayerGerber = layer_setting.gerber_file
    if thisLayerGerber != '':
        thisLayerLines = Helpers.load_Gerber(thisLayerGerber)
    thisLayerThickness = float(layer_setting.thickness)
    thisLayerThicknessUnit = 'mil'

    electric_loads_for_layer = list()
    for load_setting in sim_settings.loads:
        if load_setting.layer == layer_setting.name:
            thisLoadThisLayerCurrent = float(load_setting.current)
            thisLoadThisLayerStart = [float(load_setting.x_start), float(
                load_setting.y_start)]
            thisLoadThisLayerEnd = [float(load_setting.x_end), float(
                load_setting.y_end)]
            thisLoadThisLayer = current_tracing.ElectricLoad(load_setting.name,
                thisLoadThisLayerCurrent,
                                                      thisLoadThisLayerStart,
                                                          thisLoadThisLayerEnd
                                                          )
            electric_loads_for_layer.append(thisLoadThisLayer)

    thisLayer = layer.Layer(thisLayerName, thisLayerType, thisLayerLines,
        conductorMaterial,
                          thisLayerThickness, thisLayerThicknessUnit,
                              board_dims, simulation,
                          electric_loads_for_layer)

    if not isinstance(thisLayerGerber, type(None)) and thisLayer.layer_type ==
        "Conductor":
        if simulation.show_process:
            print("Tracing layer: " + layer_setting.name)
        thisLayer.trace_layer()

        if simulation.show_process:
            print("Finding networks: " + layer_setting.name)
        thisLayer.find_networks()

        if simulation.show_process:
            print("Calculating conduction losses: " + layer_setting.name)
        thisLayer.find_cond_loss()

    # this happens to both conductor and insulating layers
    if not isinstance(drillFile, type(None)):
        if simulation.show_process:
```

```
                print("Drilling holes: " + layer_setting.name)
            thisLayer.drill_holes(drillLayer)

        layer_list.append(thisLayer)

    board = pcb_board.Board(layer_list, board_components, simulation,
        conductorMaterial, dielectricMaterial)

    heat_transfer_analysis = heat_transfer.Simultaneous(board)
    heat_transfer_analysis.solve()

    return heat_transfer_analysis
```

B.5 heat_transfer.py

```
import time
import numpy as np

from scipy.interpolate import interp1d
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import spsolve
from math import ceil

import tracer



def lookup_g_v2(temperature_c):
    temp_vec = np.asarray([0, 20, 40, 60, 80, 100, 200])
    g_v2_vec = np.asarray([8.28e5, 6.54e5, 5.2e5, 4.27e5, 3.47e5, 2.89e5, 1.28e5])

    g_v2_interp = interp1d(temp_vec, g_v2_vec)
    return g_v2_interp(temperature_c)


def lookup_k_air(temperature_c):
    temp_vec = np.asarray([0, 20, 40, 60, 80, 100, 200])
    k_vec = np.asarray([6.02e-4, 6.375e-4, 6.731e-4, 7.087e-4, 7.442e-4, 7.798e-4,
         9.398e-4])

    k_interp = interp1d(temp_vec, k_vec)

    return k_interp(temperature_c)


class Simultaneous:
    def __init__(self, board):

        self.board = board
        self.simulation = self.board.simulation
        # build 'G' tensor matrix, and 'S' boundary conditions vector
        self.g_row = list()
        self.g_col = list()
        self.g_data = list()
        self.neighbor_c_list = list()
        self.s_list = list()

        # build G matrix and S vector
        self.grid_dims = np.append(np.size(self.board.layers), np.shape(self.board.
            layers[0].Q_mat))
        self.g_dim = np.prod(self.grid_dims)
        self.s_vec = list() # np.zeros([self.g_dim, 1])
        self.skipped_cells = list()
        # calculate reusable values
        self.cell_wid = self.simulation.resolution
        self.cell_ht = self.cell_wid
        self.cell_dep = self.board.layers[0].thickness

        # self.temp_mat = np.ones(self.grid_dims) * self.find_ave_board_temp()
        if self.simulation.show_process:
            print("Creating simultaneous simulation")
        self.htc = list()
        self.temp_mat = self.calc_initial_board_temps()

        self.q_components = np.zeros(np.append(2, np.shape(self.board.layers[0].
            Q_mat)))

    def calc_initial_board_temps(self):
        t_ave = self.calc_ave_board_temp()

        if self.simulation.show_process:
            print("Initial mean temp:" + str(t_ave))

        temp_cell_mat = list()
        for layer in range(0, self.board.mat_dep):
            layer_temp_mat = t_ave * np.ones_like(self.board.layers[layer].cond_mat
                )
            temp_cell_mat.append(layer_temp_mat)

        return np.asarray(temp_cell_mat)

    def calc_ave_board_temp(self):
        # find total heat generated in board from cond loss and component (70%)
        q_tot = self.find_total_board_q()
```

```
iter_limit = 10000
iter = 0

[L, W, H] = self.get_general_board_dims()
c_2_k = 273.16
Ta = self.simulation.ambient
beta_a = 1/(Ta+c_2_k)

# guess dt = 20 deg C
dt = 20
temp_thresh = 0.1
temp_delta = temp_thresh * 2

while abs(temp_delta) > temp_thresh:
    iter += 1

    Ts = Ta + dt
    Tf = (Ts + Ta) / 2

    g_v2 = lookup_g_v2(Tf)
    k_air = lookup_k_air(Tf)
    Pr = 0.71

    Ra_less_P = g_v2*beta_a*dt*Pr

    P_top_btm = W * L / (2 * (W + L))
    P_side = H

    Ra_top_btm = Ra_less_P * P_top_btm ** 3
    Ra_side = Ra_less_P * P_side ** 3

    if Ra_top_btm < 8e6:
```

```
    C_top = 0.54
    n_top = 1/4
else:
    C_top = 0.15
    n_top = 1 / 3

if Ra_side < 1e9:
    C_side = 0.59
    n_side = 1 / 4
else:
    C_side = 0.13
    n_side = 1 / 3

C_btm = 0.27
n_btm = 1/4

epsilon = 0.8
sigma = 5.67 * 1e-8 / 1550

# convection
h_top = (C_top * k_air * Ra_top_btm ** n_top / P_top_btm) / self.
    simulation.conv_coef
h_side = (C_side * k_air * Ra_side ** n_side / P_side) / self.
    simulation.conv_coef
h_btm = (C_btm * k_air * Ra_top_btm ** n_btm / P_top_btm) / self.
    simulation.conv_coef
q_conv = h_top * (L * W) * dt + \
        h_side * (2 * W * H) * dt + \
        h_side * (2 * L * H) * dt + \
        h_btm * (L * W) * dt

# radiation
```

```
        Tsk = Ts + c_2_k
        Tak = Ta + c_2_k
        rad_correction = self.simulation.rad_coef * (Tsk - Tak) ** self.
            simulation.rad_pow
        Cr = 2 * epsilon * sigma * (W * L + H * L + H * W) / rad_correction
        q_rad = Cr * (Tsk ** 4 - Tak ** 4)

        q_out = q_conv + q_rad
        q_error = q_tot - q_out

        if q_error > 0:
            temp_delta = q_error / q_tot
        else:
            temp_delta = -0.8 * dt
        dt += temp_delta

        if iter > iter_limit:
            if self.simulation.show_process:
                print("Iteration limit on finding board temp")
            break

    self.htc = [h_top, h_side, h_btm]
    Ts = Ta + dt
    return Ts

def get_general_board_dims(self):
    board_L = self.board.mat_ht * self.simulation.resolution
    board_W = self.board.mat_wid * self.simulation.resolution
    board_H = 0
    for layer in range(0, self.board.mat_dep):
        board_H += self.board.layers[layer].thickness
    if self.simulation.board_orientation == [0, 0]:
        L = board_L / 1000
        W = board_W / 1000
        H = board_H / 1000

    elif self.simulation.board_orientation == [1, 0] or self.simulation.
        board_orientation == [-1, 0]:
        L = board_H / 1000
        W = board_W / 1000
        H = board_L / 1000

    elif self.simulation.board_orientation == [0, 1] or self.simulation.
        board_orientation == [0, -1]:
        L = board_L / 1000
        W = board_H / 1000
        H = board_W / 1000

    return [L, W, H]

def find_total_board_q(self):
    q_cond = 0
    for layer in range(0, self.board.mat_dep): # k direction
        q_cond += np.sum(self.board.layers[layer].Q_mat)
    q_comp = 0
    for component in self.board.components:
        q_comp += component.heat

    return q_cond + q_comp

def series_conductance(self, c1, c2):
    return 1 / (1 / c1 + 1 / c2)
```

```
def prepare_sparse_matrices(self):
    start_time = time.perf_counter()
    self.s_vec = list()
    for layer in range(0, self.grid_dims[0]): # k direction
        self.cell_dep = self.board.layers[layer].thickness
        for col in range(0, self.grid_dims[2]): # j direction
            for row in range(0, self.grid_dims[1]): # i direction
                this_board_coord = [layer, row, col]
                # empty list for neighboring cells - used to build matrix cell
                    for self (sum of neighbor 'C's)
                self.neighbor_c_list = list()
                self.s_list = list()
                this_q = self._find_this_q(this_board_coord)

                self._handle_neighbors(this_board_coord)

                # handle self location in G matrix
                this_c = sum(self.neighbor_c_list)
                g_row_ind = row + col * self.board.mat_wid + layer * (self.board
                    .mat_wid * self.board.mat_ht)
                self.g_row.append(g_row_ind)
                self.g_col.append(g_row_ind)
                self.g_data.append(this_c)
                this_s = sum(self.s_list) * self.simulation.ambient + this_q
                self.s_vec.append(this_s)

    if self.simulation.show_process:
        print("Matrix prep time: " + str(time.perf_counter() - start_time))

def calc_component_heat(self):
    # total_components_heat = np.zeros_like(self.layers[0].Q_mat)
```

```
self.q_components = np.zeros(np.append(2, np.shape(self.board.layers[0].
    Q_mat)))
for component in self.board.components:
    component_heat = np.zeros_like(self.board.layers[0].Q_mat)
    # determine if top or bottom layer component
    if component.side == 'Top':
        related_layer = 0
        conv_dir = self.get_conv_dir([-1, 0, 0])
    else:
        related_layer = len(self.board.layers) - 1
        conv_dir = self.get_conv_dir([1, 0, 0])
    # Find related layer cells which are conductor material (set)
    related_cells = list()
    cells_temps = list()
    for row in range(ceil((component.x - component.width / 2) / self.
        simulation.resolution),
                    ceil((component.x + component.width / 2) / self.
                        simulation.resolution)):
        for col in range(ceil((component.y - component.length / 2) / self.
            simulation.resolution),
                        ceil((component.y + component.length / 2) / self.
                            simulation.resolution)):

            # assumes nearly all conduction is transferred through exposed
                conductor
            if self.board.layers[related_layer].cond_mat[row, col] > tracer.
                Cell.INSULATOR.value:
                related_cells.append([row, col])
                cells_temps.append(self.temp_mat[related_layer, row, col])

    # Find mean temperature of cells -> dT = T_mean - T_ambient
    temp_mean = np.average(np.asarray(cells_temps))
```

```
        delta_temp = temp_mean - self.simulation.ambient

        # calculate heat lose through top of component through convection and
            radiation
        htc = self.get_htc(conv_dir, self.simulation.ambient, temp_mean) * self
            .simulation.comp_htc_coef
        heat_conv_rad = htc * component.width * component.length * delta_temp
        # the remaining heat is conductive heat into board, its heat is
            distributed to the set of layer cells
        heat_cond = (component.heat - heat_conv_rad)
        heat_per_cell = heat_cond / len(related_cells)

        for related_cell in related_cells:
            component_heat[related_cell[0], related_cell[1]] = heat_per_cell
        self.q_components[np.sign(related_layer)] += component_heat

def get_htc(self, orientation, temp_amb, temp_surf):
    htc_conv = 0
    C2K = 273.16

    # free convection
    if orientation == 'vertical':
        htc_conv = self.htc[1]
    elif orientation == 'horizontal top':
        htc_conv = self.htc[0]
    elif orientation == 'horizontal bottom':
        htc_conv = self.htc[2]

    # radiation
    Tsk = temp_surf + C2K
    Tak = temp_amb + C2K
```

```
        rad_correction = self.simulation.rad_coef * (Tsk - Tak) ** self.simulation.
            rad_pow

        epsilon = 0.8 # assumed for solder mask (close to epoxy paint)
        # Stefan Boltzmann Constant converted to W / (in^2 K^4)
        sigma = 5.67 * 1e-8 / 1550
        htc_rad = epsilon * sigma * ((Tsk) * (Tsk) + (Tak) * (Tak)) * (
                (Tsk) + (Tak)) / rad_correction
        # unit of W / (mil^2 C), from W / (in^2 K)
        htc = (htc_conv + htc_rad) * 1e-6

        return htc

    def solve(self):
        if self.simulation.show_process:
            print("Solving simultaneous solution")

        self.calc_component_heat()

        self.prepare_sparse_matrices()
        g_mat = csr_matrix((np.asarray(self.g_data), (np.asarray(self.g_row), np.
            asarray(self.g_col))),
                          shape=(len(self.s_vec), len(self.s_vec)))

        solve_start = time.perf_counter()
        t_vec = np.transpose(spsolve(g_mat, np.asarray(self.s_vec)))
        if self.simulation.show_process:
            print("Matrix solve time: " + str(time.perf_counter() - solve_start))

        self.temp_mat = np.rot90(t_vec.reshape(self.grid_dims[0], self.grid_dims
            [2], self.grid_dims[1]),
                                k=3, axes=(1, 2))
        if self.simulation.show_process:
            print("Final mean temp:" + str(np.mean(self.temp_mat)))

    def _handle_neighbors(self, board_coord):
        # look in i-1 dir
        neighbor_dir = [0, -1, 0]
        self._neighbor_ij(board_coord, neighbor_dir)

        # look in i+1 dir
        neighbor_dir = [0, 1, 0]
        self._neighbor_ij(board_coord, neighbor_dir)

        # look in j-1 dir
        neighbor_dir = [0, 0, -1]
        self._neighbor_ij(board_coord, neighbor_dir)

        # look in j+1 dir
        neighbor_dir = [0, 0, 1]
        self._neighbor_ij(board_coord, neighbor_dir)

        # look in k-1 dir
        neighbor_dir = [-1, 0, 0]
        self._neighbor_k(board_coord, neighbor_dir)

        # look in k+1 dir
        neighbor_dir = [1, 0, 0]
        self._neighbor_k(board_coord, neighbor_dir)

    def _find_this_q(self, coord):

        # find q from conduction losses
        q_cond = self.board.layers[coord[0]].Q_mat[coord[1], coord[2]]
```

```
        # find q from component source (q_cond = q_tot - q_conv - q_rad)
        if coord[0] == 0:
            q_component = self.q_components[0, coord[1], coord[2]]
        elif coord[0] == self.board.mat_dep - 1:
            q_component = self.q_components[1, coord[1], coord[2]]
        else:
            q_component = 0

        return q_cond + q_component

    def get_area(self, neighbor_dir):
        if neighbor_dir[0] != 0:
            area = self.cell_wid * self.cell_ht
        else:
            area = self.cell_wid * self.cell_dep
        return area

    def get_depth(self, neighbor_dir):
        if neighbor_dir[0] == 0:
            depth = self.cell_dep
        else:
            depth = self.cell_wid
        return depth

    def get_conv_dir(self, neighbor_dir):
        conv_dir = 'vertical'
        if self.simulation.board_orientation == [0, 0]:
            if neighbor_dir == [-1, 0, 0]:
                conv_dir = 'horizontal top'
            elif neighbor_dir == [1, 0, 0]:
                conv_dir = 'horizontal bottom'
        elif self.simulation.board_orientation == [1, 0]:
            if neighbor_dir == [0, 0, 1]:
                conv_dir = 'horizontal top'
            elif neighbor_dir == [0, 0, -1]:
                conv_dir = 'horizontal bottom'

        elif self.simulation.board_orientation == [0, 1]:
            if neighbor_dir == [0, -1, 0]:
                conv_dir = 'horizontal top'
            elif neighbor_dir == [0, 1, 0]:
                conv_dir = 'horizontal bottom'

        elif self.simulation.board_orientation == [-1, 0]:
            if neighbor_dir == [0, 0, -1]:
                conv_dir = 'horizontal top'
            elif neighbor_dir == [0, 0, 1]:
                conv_dir = 'horizontal bottom'

        elif self.simulation.board_orientation == [0, -1]:
            if neighbor_dir == [0, 1, 0]:
                conv_dir = 'horizontal top'
            elif neighbor_dir == [0, -1, 0]:
                conv_dir = 'horizontal bottom'

        return conv_dir

    def neighbor_c_air(self, coord, neighbor_dir):
        area = self.get_area(neighbor_dir)
        conv_dir = self.get_conv_dir(neighbor_dir)
        htc = self.get_htc(conv_dir, self.simulation.ambient, self.temp_mat[coord
            [0], coord[1], coord[2]])
        return htc * area

    def neighbor_c_cond(self, coord, neighbor_dir):
        k_coord = [coord[0] + neighbor_dir[0], coord[1] + neighbor_dir[1], coord[2]
             + neighbor_dir[2]]

        neighbor_c = self.find_c_cond(k_coord, neighbor_dir)

        return neighbor_c

    def self_c_cond(self, coord, neighbor_dir):
        k_coord = coord

        self_c = self.find_c_cond(k_coord, neighbor_dir)

        return self_c

    def find_c_cond(self, coord, neighbor_dir):
        this_k_cell_material = self.board.layers[coord[0]].cond_mat[coord[1], coord
            [2]]
        if this_k_cell_material > tracer.Cell.INSULATOR.value:
            this_k = np.sum(
                np.fabs(np.asarray(neighbor_dir)) * np.asarray(self.board.cond_k) /
                    self.simulation.cond_k_coef)
        elif this_k_cell_material == tracer.Cell.AIR.value:
            this_k = np.sum(np.fabs(np.asarray(neighbor_dir)) * np.asarray(self.
                board.sold_k))
        else:
            this_k = np.sum(
                np.fabs(np.asarray(neighbor_dir)) * np.asarray(self.board.diel_k) /
                    self.simulation.diel_k_coef)
        area = self.get_area(neighbor_dir)
        # depth is half due to grid format
        depth = self.get_depth(neighbor_dir) / 2

        return this_k * area / depth

    def _neighbor_ij(self, coord, neighbor_dir):
        if neighbor_dir[1] != 0:
            which_coord = 1
            self_dir = [0, 1, 0]

        else:
            which_coord = 2
            self_dir = [0, 0, 1]

        # find self C
        self_c = self.self_c_cond(coord, self_dir)

        # if neighbor is air on border, add to S vector and G matrix
        # edge of board -> conv and rad thru air
        if (coord[which_coord] == 0 and neighbor_dir[which_coord] == -1) or (
                coord[1] == (self.board.mat_wid - 1) and neighbor_dir[1] == 1) or (
                coord[2] == (self.board.mat_ht - 1) and neighbor_dir[2] == 1):
            neighbor_c = self.neighbor_c_air(coord, neighbor_dir)
            dir_c = self.series_conductance(self_c, neighbor_c)
            self.s_list.append(dir_c)
            self.neighbor_c_list.append(dir_c)
        else:
            neighbor_c = self.neighbor_c_cond(coord, neighbor_dir)
            dir_c = self.series_conductance(self_c, neighbor_c)
            self.neighbor_c_list.append(dir_c)
            g_row_ind = coord[1] + coord[2] * self.board.mat_wid + coord[0] * (self
                .board.mat_wid * self.board.mat_ht)
            g_col_ind = g_row_ind + neighbor_dir[1] + neighbor_dir[2] * self.board.
                mat_wid
            self.g_row.append(g_row_ind)
            self.g_col.append(g_col_ind)
            self.g_data.append(-dir_c)

    def _neighbor_k(self, coord, neighbor_dir):
        which_coord = 0
        self_c = self.self_c_cond(coord, [1, 0, 0])

        # if neighbor is air, add to S vector and G matrix
        # edge of board -> conv and rad thru air
        if (coord[which_coord] == 0 and neighbor_dir[which_coord] == -1) or (
                coord[which_coord] == (self.board.mat_dep - 1) and neighbor_dir[
                    which_coord] == 1):
            neighbor_c = self.neighbor_c_air(coord, neighbor_dir)
            dir_c = self.series_conductance(self_c, neighbor_c)
            self.s_list.append(dir_c)
            self.neighbor_c_list.append(dir_c)
        # conduction thru board
        # find neighbor C, then add to G matrix and nei_Cs
        # assumes that holes are made is k direction so if this isn't an air cell,
            it's neighbor in k won't be either
        else:
            neighbor_c = self.neighbor_c_cond(coord, neighbor_dir)
            dir_c = self.series_conductance(self_c, neighbor_c)
            self.neighbor_c_list.append(dir_c)

            g_row_ind = coord[1] + coord[2] * self.board.mat_wid + coord[0] * (self
                .board.mat_wid * self.board.mat_ht)
            g_col_ind = g_row_ind + neighbor_dir[0] * (self.board.mat_wid * self.
                board.mat_ht)
            self.g_row.append(g_row_ind)
            self.g_col.append(g_col_ind)
            self.g_data.append(-dir_c)
```

B.6 pcb_board.py

```
def get_k_values(material):
    # values in W/inC
    # [thru-plane(k), in-plane (i, j)]
    switch = {
        'Copper': [9.9, 9.9, 9.9],
        'Aluminum': [5.5, 5.5, 5.5],
        'Gold': [7.5, 7.5, 7.5],
        'Silver': [10.6, 10.6, 10.6],
        'Nickel': [2.3, 2.3, 2.3],
        'Solder': [1.46, 1.46, 1.46],
        'Epoxy': [0.09, 0.09, 0.09],
        'Fr-4': [0.00737, 0.020574, 0.020574],
        'Polyamide': [0.005, 0.005, 0.005],
        'ThermalCompound': [0.02, 0.02, 0.02]
    }

    return switch.get(material)



class Board:
    def __init__(self, layers, components, simulation, cond_material,
        diel_material):
        self.layers = layers
        self.components = components
        self.simulation = simulation
        self.cond_material = cond_material
        self.diel_material = diel_material
        [self.mat_wid, self.mat_ht] = self.layers[0].cond_mat.shape
        self.mat_dep = len(self.layers)
        self.cond_k = get_k_values(self.cond_material)
        self.diel_k = get_k_values(self.diel_material)
        self.sold_k = get_k_values('Solder')
```

B.7 layer.py

```
import numpy as np

import current_tracing
import tracer

mm_to_mil = 1000 / 25.4


def get_board_dims(keep_out_lines):
    sig_dig = find_sig_digit(keep_out_lines)

    # find units
    units = find_units(keep_out_lines)

    # find overall dims
    return find_overall_dims(keep_out_lines, units, sig_dig)


def find_overall_dims(keep_out_lines, units, sig_dig):
```

```
width = 0
height = 0
feature_id = 'x'
feature_active = 'none'
# first, find line that indicates new feature
for line in keep_out_lines:
    if line.startswith('%AD'):
        feature_arr = line.split(',')
        feature_id_str = feature_arr[0]
        feature_id = feature_id_str[3:-1] + '*\n'

    if line == feature_id:
        feature_active = feature_id

    if feature_active == feature_id:
        # find x dim -> width
        line_x = line.find('X')
        line_y = line.find('Y')
        line_d = line.find('D')
        if line_x != -1:
            if line_y != -1:
                width = float(line[line_x + 1: line_y]) * sig_dig
            else:
                width = float(line[line_x + 1: line_d]) * sig_dig

        # find y dim -> height
        if line_y != -1:
            height = float(line[line_y + 1: line_d]) * sig_dig

    if width != 0 and height != 0:
        break
    return [width, height]



def find_units(keep_out_lines):
    units = 'in'
    for line in keep_out_lines:
        if line.startswith('%MO'):
            if line.endswith('IN*%\n'):
                units = 'in'
            elif line.endswith('MM*%\n'):
                units = 'mm'
            break
    return units



def find_sig_digit(keep_out_lines):
    sig_dig = 1
    for line in keep_out_lines:
        if line.startswith('%FSLA'):
            if line.endswith('X23Y23*%\n'):
                sig_dig = 1
            elif line.endswith('X24Y24*%\n'):
                sig_dig = 0.1
            elif line.endswith('X25Y25*%\n'):
                sig_dig = 0.01
            elif line.endswith('X42Y42*%\n'):
                sig_dig = 0.01 * mm_to_mil
            elif line.endswith('X43Y43*%\n'):
                sig_dig = 0.001 * mm_to_mil
            elif line.endswith('X44Y44*%\n'):
                sig_dig = 0.0001 * mm_to_mil
            break
    return sig_dig


class Layer:
    def __init__(self, name, layer_type, lines, cond_material, thickness_val,
        thickness_type, dims, simulation, loads):
        self.name = name
        self.layer_type = layer_type
        self.lines = lines
        self.cond_material = cond_material
        self.thickness = self.convert_layer_thickness(thickness_val, thickness_type
            )
        self.width = int(dims[0])
        self.height = int(dims[1])
        self.simulation = simulation
        self.sim_res = self.simulation.resolution
        self.cond_mat = np.zeros([int(self.width / self.sim_res), int(self.height /
             self.sim_res)], dtype=int)
        self.res_mat = np.zeros(np.shape(self.cond_mat), dtype=float)
        self.Q_mat = np.zeros(np.shape(self.cond_mat), dtype=float)
        self.loads = loads

        if not isinstance(lines, type(None)):
            self.sig_dig = find_sig_digit(lines)
            self.units = find_units(lines)

    def trace_layer(self):
        # go line by line instructions
        feature_list = []
        this_feature_id = 'none'
        x_trace = -1.0
        y_trace = -1.0
        x_trace_prev = x_trace
```

```
y_trace_prev = y_trace
draw_trace = False
pour_count = 0

for line in self.lines:
    # if feature call out, add to feature array
    if line.startswith('%AD'):
        feature_arr = line.split(',')
        feature_id_str = feature_arr[0]
        feature_dim_str = feature_arr[1]

        feature_id = feature_id_str[3:-1] + '*\n'
        feature_class = feature_id_str[-1]
        if feature_class == 'C':
            if self.units == 'in':
                feature_rad = float(feature_dim_str[0:-3]) * 500
            else:
                feature_rad = float(feature_dim_str[0:-3]) * mm_to_mil / 2
            new_feature = tracer.Circle(feature_id, feature_rad)

        elif feature_class == 'R' or feature_class == 'O':
            feature_dims = feature_dim_str[0:-3].split('X')
            if self.units == 'in':
                feature_width = float(feature_dims[0]) * 1000
                feature_height = float(feature_dims[1]) * 1000
            else:
                feature_width = float(feature_dims[0]) * mm_to_mil
                feature_height = float(feature_dims[1]) * mm_to_mil

            if feature_class == 'R':
                new_feature = tracer.Rectangle(feature_id, feature_width,
                    feature_height)
            elif feature_class == 'O':
                new_feature = tracer.Oval(feature_id, feature_width,
                    feature_height)

        feature_list.append(new_feature)

    # if feature, start tracing
    elif line.startswith('D') and not line.startswith('D02') and not line.
        startswith('D03'):
        draw_trace = True
        which_feature = 0
        this_feature_id = line
        for feature in feature_list:

            if feature.f_id == this_feature_id:
                break
            which_feature = which_feature + 1

    elif line.startswith('G36'):
        # start of pour feature
        draw_trace = False
        this_feature_id = 'pour'
        pour_points = []
        pour_count = pour_count + 1

    elif line.startswith('G37'):
        # end of pour feature
        draw_trace = True
        this_feature_id = 'pour_done'
        new_feature = tracer.Pour('pour' + str(pour_count), pour_points)
        feature_list.append(new_feature)
        tracer.trace_pour(self.cond_mat, pour_points, self.sim_res, tracer.
            Cell.CONDUCTOR.value)

    elif this_feature_id == 'pour':
        # Begin gathering points for pour
        [x_trace, y_trace, x_trace_prev, y_trace_prev, line_x, line_y,
            line_d] = Layer.x_y_d_pos(self, line,
                                                                            x_

                                                                            y_

                                                                            x_

                                                                            y_

        if x_trace != x_trace_prev or y_trace != y_trace_prev:
            pour_points.append([x_trace, y_trace])

    elif draw_trace:
        [x_trace, y_trace, x_trace_prev, y_trace_prev, line_x, line_y,
            line_d] = Layer.x_y_d_pos(self, line,
                                                                            x_

                                                                            y_
```

```
x_

y_
```

```
if line_d != -1:
    feature_step = line[line_d:]
    if feature_step == 'D01*\n':
        # continue drawing from previous x & y
        this_feature = feature_list[which_feature]
        tracer.trace_line(self.cond_mat, x_trace_prev, y_trace_prev,
             x_trace, y_trace,
                          this_feature.radius, self.sim_res, tracer.
                              Cell.CONDUCTOR.value)

    elif feature_step == 'D03*\n':
        # end trace/shape
        this_feature = feature_list[which_feature]
        if this_feature.__class__ == tracer.Rectangle:
            tracer.trace_rectangle(self.cond_mat, x_trace, y_trace,
                this_feature.width,
                                   this_feature.height, self.sim_res,
                                       tracer.Cell.CONDUCTOR.value)
        elif this_feature.__class__ == tracer.Oval:
            tracer.trace_oval(self.cond_mat, x_trace, y_trace,
                this_feature.width, this_feature.height,
                              self.sim_res, tracer.Cell.CONDUCTOR.value
                                  )
        elif this_feature.__class__ == tracer.Circle:
```

```
                        tracer.trace_circle(self.cond_mat, x_trace, y_trace,
                            this_feature.radius, self.sim_res,
                                      tracer.Cell.CONDUCTOR.value)

def x_y_d_pos(self, this_line, x, y, x_prev, y_prev):
    line_x = this_line.find('X')
    line_y = this_line.find('Y')
    line_d = this_line.find('D')

    if line_x != -1:
        if line_y != -1:
            x_prev = x
            x = float(this_line[line_x + 1: line_y]) * self.sig_dig
        else:
            x_prev = x
            x = float(this_line[line_x + 1: line_d]) * self.sig_dig
    else:
        x_prev = x

    if line_y != -1:
        y_prev = y
        y = float(this_line[line_y + 1: line_d]) * self.sig_dig
    else:
        y_prev = y

    return [x, y, x_prev, y_prev, line_x, line_y, line_d]

def convert_layer_thickness(self, thickness_val, thickness_type):
    if thickness_type == 'oz':
        thickness_rtn = thickness_val * 1.37
    elif thickness_type == 'in':
        thickness_rtn = thickness_val / 1000
    elif thickness_type == 'mil':
        thickness_rtn = thickness_val
    elif thickness_type == 'mm':
        thickness_rtn = thickness_val * .0254
    return thickness_rtn

def find_networks(self):
    # starting network ID is 1
    new_network_id = 1
    [n_rows, n_cols] = self.cond_mat.shape

    net_add = 7

    for row in range(0, n_rows):
        for col in range(0, n_cols):
            if self.cond_mat[row, col] != tracer.Cell.INSULATOR.value and self.
                cond_mat[
                row, col] != tracer.Cell.AIR.value:
                for neighbor_row in range(row - 1, row + 2):
                    for neighbor_col in range(col - 1, col + 2):
                        if 0 <= neighbor_row <= n_rows - 1 and 0 <= neighbor_col
                            <= n_cols - 1 \
                                and not (neighbor_row == row and neighbor_col ==
                                    col):
                            neighbor_cond_val = self.cond_mat[neighbor_row,
                                neighbor_col]
                            # check if neighbor cell already has a network
                            if self.cond_mat[neighbor_row, neighbor_col] > 0:
                                # check whether this cell doesn't yet have a
                                    network
                                if self.cond_mat[row, col] == tracer.Cell.
                                    CONDUCTOR.value:
```

```
                        self.cond_mat[row, col] = self.cond_mat[
                            neighbor_row, neighbor_col]
                    # check whether neighbor cell's network is lower
                        number
                    elif self.cond_mat[neighbor_row, neighbor_col] <
                        self.cond_mat[row, col]:
                        self.cond_mat[row, col] = self.cond_mat[
                            neighbor_row, neighbor_col]
                    # case where this cell already has a network
                    # and it's network is lower number than it's
                        neighbor
                    elif self.cond_mat[neighbor_row, neighbor_col] >
                        self.cond_mat[row, col]:
                        # find all cells with neighboring network
                            number
                        # set all of those cells to this cell network
                            number
                        higher_network = np.argwhere(self.cond_mat ==
                            neighbor_cond_val)
                        for higher_cell in higher_network:
                            self.cond_mat[higher_cell[0], higher_cell
                                [1]] = self.cond_mat[row, col]
                # case where this cell has a network but the neighbor
                     cell doesn't
                elif self.cond_mat[neighbor_row, neighbor_col] < 0 \
                        and self.cond_mat[row, col] != tracer.Cell.
                            CONDUCTOR.value:
                    self.cond_mat[neighbor_row, neighbor_col] = self.
                        cond_mat[row, col]

        if self.cond_mat[row, col] == tracer.Cell.CONDUCTOR.value:
            self.cond_mat[row, col] = new_network_id
```

```
                    new_network_id = new_network_id + net_add
                    if net_add == 7:
                        net_add = 5
                    elif net_add == 5:
                        net_add = -3
                    else:
                        net_add = 7

def find_cond_loss(self):
    cond_q_maps = []

    # for each load, find associated network
    for electric_load in self.loads:
        if self.simulation.show_process:
            print("Losses for load: " + electric_load.name)
        # convert start and end locations to matrix coord
        path_start = [int(electric_load.path_start[0] / self.sim_res),
                      int(electric_load.path_start[1] / self.sim_res)]
        path_end = [int(electric_load.path_end[0] / self.sim_res),
                    int(electric_load.path_end[1] / self.sim_res)]

        # find which network is at start
        this_network = self.cond_mat[path_start[0], path_start[1]]
        # check that end is same network
        end_network_check = self.cond_mat[path_end[0], path_end[1]]
        if this_network == end_network_check:
            # filter cond_mat to a matrix which only contains this network
            network_map = np.where(self.cond_mat == this_network, 1, 0)
            # pass in filtered matrix (map), start, end into current tracing for
                 res_mat
            # for a network with a load, use current tracing to find resistance
                of each cell.
            # multiply the resistance by the current squared to get Q for cell
            this_network_q_mat = electric_load.current * electric_load.current \
                                 * current_tracing.set_res_values(path_start,
                                     path_end, network_map,
                                                            self.thickness,
                                                               self.
                                                               cond_material)
            cond_q_maps.append(this_network_q_mat)

    for this_q_map in cond_q_maps:
        self.Q_mat = np.add(self.Q_mat, this_q_map)

def drill_holes(self, drill_layer):
    hole_cells = np.argwhere(drill_layer.hole_mat == tracer.Cell.AIR.value)
    plated_cells = np.argwhere(drill_layer.hole_mat == tracer.Cell.CONDUCTOR.
        value)
    for hole_cell in hole_cells:
        self.cond_mat[hole_cell[0], hole_cell[1]] = tracer.Cell.AIR.value
    for plated_cell in plated_cells:
        if self.cond_mat[plated_cell[0], plated_cell[1]] <= tracer.Cell.
            INSULATOR.value:
            self.cond_mat[plated_cell[0], plated_cell[1]] = -1 * tracer.Cell.
                CONDUCTOR.value
```

B.8 drill.py

```
import numpy as np
import tracer

mm_to_mil = 1000 / 25.4
```

```
class Drill:

    def __init__(self, lines, dims, sim_resolution, thickness_val, thickness_type)
        :
        self.lines = lines
        self.width = int(dims[0])
        self.height = int(dims[1])
        self.sim_res = sim_resolution
        self.thickness = self.convert_layer_thickness(thickness_val, thickness_type
            )

        self.hole_mat = np.zeros([int(self.width / sim_resolution), int(self.height
             / sim_resolution)], dtype=int)

        if not isinstance(lines, type(None)):
            [self.sig_dig, self.units] = self.find_units_sig_dig()

    def trace_drill(self):
        # go line by line instructions
        hole_list = []
        this_hole_id = 'none'
        hole_ref = -1
        x_hole = -1.0
        y_hole = -1.0
        which_hole = -1

        hole_list_add = True

        for line in self.lines:
            # if hole call out, add to hole array
            if line.startswith('T'):
```

```
    # still adding the tools to the hole array
    if hole_list_add:
        hole_arr = line.split('F')
        hole_id_str = hole_arr[0]
        hole_id = int(hole_id_str[1:])
        hole_dim_str = hole_arr[1].split('C')
        if self.units == "in":
            hole_rad = float(hole_dim_str[1]) * 500
        else:
            hole_rad = float(hole_dim_str[1]) * mm_to_mil / 2
        new_hole = tracer.Circle(hole_id, hole_rad)
        hole_list.append(new_hole)

    # call out for switching tool
    else:
        which_hole = 0
        this_hole_id = int(line[1:])
        for hole in hole_list:
            if hole.f_id == this_hole_id:
                break
            which_hole = which_hole + 1

elif line.startswith('%'):
    # change from adding tools to using tools
    hole_list_add = False

elif line.startswith('X') or line.startswith('Y'):
    this_hole = hole_list[which_hole]
    [x_hole, y_hole] = self.x_y_pos(line, x_hole, y_hole)
    tracer.trace_circle(self.hole_mat, x_hole, y_hole, this_hole.radius,
         self.sim_res,
                        tracer.Cell.CONDUCTOR.value)
            lesser_rad = this_hole.radius - self.thickness # max(self.thickness,
                 self.sim_res)
            tracer.trace_circle(self.hole_mat, x_hole, y_hole, lesser_rad, self.
                sim_res,
                               tracer.Cell.AIR.value)

    def x_y_pos(self, this_line, prev_x, prev_y):
        line_x = this_line.find('X')
        line_y = this_line.find('Y')
        x_str = ''
        y_str = ''
        x = prev_x
        y = prev_y

        if line_x != -1:
            if line_y != -1:
                x_str = this_line[line_x + 1: line_y]
            else:
                x_str = this_line[line_x + 1: -1]

            x = self.number_sig_dig(x_str, self.sig_dig)
            if self.units == "mm":
                x = x * mm_to_mil

        if line_y != -1:
            y_str = this_line[line_y + 1: -1]
            y = self.number_sig_dig(y_str, self.sig_dig)
            if self.units == "mm":
                y = y * mm_to_mil

        return [x, y]
```

```
def number_sig_dig(self, this_string, sig_dig):
    while len(this_string) < sig_dig:
        this_string = this_string + '0'
    ret_val = float(this_string[0:sig_dig])
    if len(this_string) > len(this_string[0:sig_dig]):
        ret_val = ret_val + float(this_string[sig_dig:]) / (10 ^ (len(
            this_string[sig_dig:])))
    return ret_val

def convert_layer_thickness(self, thickness_val, thickness_type):
    if thickness_type == 'oz':
        thickness_rtn = thickness_val * 1.37
    elif thickness_type == 'in':
        thickness_rtn = thickness_val / 1000
    elif thickness_type == 'mil':
        thickness_rtn = thickness_val
    elif thickness_type == 'mm':
        thickness_rtn = thickness_val / mm_to_mil
    return thickness_rtn

def find_units_sig_dig(self):
    units = 'in'
    for line in self.lines:
        if line.startswith('INCH'):
            units = 'in'
            sig_dig = 5
            break
        elif line.startswith('METRIC'):
            units = 'mm'
            sig_dig = 4
            break
    return [sig_dig, units]
```

B.9 tracer.py

```
import numpy as np
from enum import Enum
from shapely.geometry import Polygon, Point
from math import ceil, floor, sqrt


class Component:
    def __init__(self, name, dims, location, heat_generated, side):
        self.name = name
        [self.width, self.length] = dims
        [self.x, self.y] = location
        self.heat = heat_generated
        self.side = side


class Circle:
    def __init__(self, f_id, radius):
        self.f_id = f_id
        self.radius = radius


class Rectangle:
    def __init__(self, f_id, width, height):
        self.f_id = f_id
        self.width = width
        self.height = height


class Oval:
    def __init__(self, f_id, width, height):
        self.f_id = f_id
        self.width = width
        self.height = height


class Line:
    def __init__(self, f_id, x_start, y_start, x_end, y_end, radius):
        self.f_id = f_id
        self.x_start = x_start
        self.y_start = y_start
        self.x_end = x_end
        self.y_end = y_end
        self.radius = radius


class Pour:
    def __init__(self, f_id, points):
        self.f_id = f_id
        self.points = points


class Simulation:
    def __init__(self, resolution, ambient_c, board_orientation, show_process,
        cond_in_plane_k, cond_thru_plane_k,
                diel_in_plane_k, diel_thru_plane_k, conv_coef, rad_coef, rad_pow,
                    comp_htc_coef):
        self.resolution = resolution
        self.ambient = ambient_c
        self.board_orientation = board_orientation
        self.show_process = show_process
        self.cond_k_coef = [cond_thru_plane_k, cond_in_plane_k, cond_in_plane_k]
        self.diel_k_coef = [diel_thru_plane_k, diel_in_plane_k, diel_in_plane_k]
        self.conv_coef = conv_coef
        self.rad_coef = rad_coef
        self.rad_pow = rad_pow
        self.comp_htc_coef = comp_htc_coef


class Cell(Enum):
    CONDUCTOR = -1
    AIR = -2
    INSULATOR = 0


def trace_circle(cond_mat, x_center, y_center, radius, res, cell_value):
    # https://www.redblobgames.com/grids/circle-drawing/
    top = floor((y_center - radius) / res)
    bottom = ceil((y_center + radius) / res)
    left = floor((x_center - radius) / res)
    right = ceil((x_center + radius) / res)

    for y in range(top, bottom):
        for x in range(left, right):
            dx = (x_center / res) - x
            dy = (y_center / res) - y
            dist_sq = dx*dx + dy*dy
            if dist_sq <= ((radius*radius) / res / res):
                cond_mat[x, y] = cell_value


def trace_line(cond_mat, x_start, y_start, x_end, y_end, radius, res, cell_value):
    # draw circles from start to end with some increment
    dx = x_end - x_start
    dy = y_end - y_start
    x_trace = x_start
    y_trace = y_start
    x_trace_prev = 0
    y_trace_prev = 0
    leng = sqrt(dx*dx + dy*dy)
    for p in range(0, floor(leng)):
        if floor(x_trace_prev) != floor(x_trace) or floor(y_trace_prev) != floor(
            y_trace):
            trace_circle(cond_mat, floor(x_trace), floor(y_trace), radius, res,
                cell_value)
        x_trace_prev = x_trace
        y_trace_prev = y_trace
        x_trace = x_trace + dx/leng
        y_trace = y_trace + dy/leng


def trace_rectangle(cond_mat, x_center, y_center, width, height, res, cell_value):
    top = floor((y_center - height/2) / res)
    bottom = ceil((y_center + height/2) / res)
    left = floor((x_center - width/2) / res)
    right = ceil((x_center + width/2) / res)

    for y in range(top, bottom):
        for x in range(left, right):
            cond_mat[x, y] = cell_value


def trace_oval(cond_mat, x_center, y_center, width, height, res, cell_value):
    if width > height:
        # left right circles
        width_sm = width - height
        trace_circle(cond_mat, floor(x_center - width_sm / 2), y_center, floor(
            height / 2), res, cell_value)
        trace_circle(cond_mat, ceil(x_center + width_sm / 2), y_center, floor(
            height / 2), res, cell_value)
        trace_rectangle(cond_mat, x_center, y_center, width_sm, height, res,
            cell_value)
    else:
        # top bottom circles
        height_sm = height - width
        trace_circle(cond_mat, x_center, floor(y_center - height_sm / 2), floor(
            width / 2), res, cell_value)
        trace_circle(cond_mat, x_center, ceil(y_center + height_sm / 2), floor(
            width / 2), res, cell_value)
        trace_rectangle(cond_mat, x_center, y_center, width, height_sm, res,
            cell_value)


def trace_pour(cond_mat, points, res, cell_value):
    pour_points = np.array(points) / res
    poly_pour = Polygon(pour_points)
    bounds = poly_pour.bounds

    for y in range(int(bounds[1]), int(bounds[3])):
        for x in range(int(bounds[0]), int(bounds[2])):
            P = Point(x, y)
            if poly_pour.contains(P):
                cond_mat[x, y] = cell_value
```

B.10 current_tracing.py

```
from pathfinding.core.grid import Grid
from pathfinding.finder.a_star import AStarFinder
from pathfinding.core.diagonal_movement import DiagonalMovement
import numpy as np


class ElectricLoad:
    def __init__(self, name, current, path_start, path_end):
        self.name = name
        self.current = current
        self.path_start = path_start
        self.path_end = path_end


def find_current_path(path_start, path_end, mat_network):
    map_grid = Grid(matrix=np.transpose(mat_network))
    start = map_grid.node(path_start[0], path_start[1])
    end = map_grid.node(path_end[0], path_end[1])

    finder = AStarFinder(diagonal_movement=DiagonalMovement.always)

    [path, runs] = finder.find_path(start, end, map_grid)
    return path


def find_straight_path(path_start, path_end):
    path = []
    cursor_loc = np.asarray(path_start)
    path.append([cursor_loc[0], cursor_loc[1]])
    this_step = np.zeros([2], dtype=int)

    while not (path_end[0] == cursor_loc[0] and path_end[1] == cursor_loc[1]):
        this_step[0] = np.sign(path_end[0] - cursor_loc[0])
        this_step[1] = np.sign(path_end[1] - cursor_loc[1])
        cursor_loc = cursor_loc + this_step
        path.append([cursor_loc[0], cursor_loc[1]])

    return path


def find_middle_path(path_start, mat_network, short_path):
    steps = np.diff(np.transpose(np.asarray(short_path)))
    steps_trans = np.transpose(steps)
    p_orth = np.transpose([steps[1], -1 * steps[0]])
    way_points = []
    concat_path = []
    concat_steps = []
    way_points_back = 1

    cursor_loc = path_start
    for step in range(0, int(np.size(steps) / 2)):
        center_cell = cell_at_width_center(cursor_loc, mat_network, steps_trans[
            step], p_orth[step])
        way_points.append(center_cell)
        cursor_loc = cursor_loc + steps_trans[step]
    way_points.append(np.array(short_path[-1]))

    for way_point_no in range(1, len(way_points)):
        intermediate_path = find_straight_path(way_points[way_point_no-
            way_points_back], way_points[way_point_no])
        intermediate_steps = np.diff(np.asarray(intermediate_path), axis=0)
        l_path = len(intermediate_path)
        if l_path > 1:
            if len(concat_steps) > 0:
                step_direction_change = np.diff([concat_steps[-1],
                    intermediate_steps[0]], axis=0)
                step_change_dist = np.linalg.norm(step_direction_change)
                if abs(step_change_dist) <= 1:
                    concat_path.extend(intermediate_path[:-1])
                    concat_steps.extend(intermediate_steps)
                    way_points_back = 1
                    if way_point_no == (len(way_points) - 1):
                        concat_path.extend([intermediate_path[-1]])
                else:
                    way_points_back = way_points_back + 1
            else:
                concat_path.extend(intermediate_path[:-1])
                concat_steps.extend(intermediate_steps)
                way_points_back = 1
        elif l_path == 1:
            way_points_back = way_points_back + 1

    return [tuple(x) for x in concat_path] # concat_path


def cell_at_width_center(start_loc, mat_network, step, p_orth):
    # move from the current location in the path orthogonally until out of trace
    cursor_loc = start_loc

    trace_width = find_trace_width(cursor_loc, mat_network, step, p_orth)
    cursor_loc = cell_at_edge_of_width(start_loc, mat_network, p_orth)

    step_size = np.linalg.norm(step)
    step_progress = step_size

    while step_progress <= (trace_width / 2):
        cursor_loc = cursor_loc - p_orth
        step_progress = step_progress + step_size

    return cursor_loc


def set_res_values(start, end, mat_network, cell_thickness, material):
    short_path = find_current_path(start, end, mat_network)
    middle_path = find_middle_path(start, mat_network, short_path)
    acceptable_gap = np.linalg.norm(np.asarray(short_path[0]) - np.asarray(
        middle_path[0])) * 5
    if np.linalg.norm(np.asarray(short_path[-1]) - np.asarray(middle_path[-1])) <=
         acceptable_gap:
        path_taken = middle_path
    else:
        path_taken = short_path
    res_mat = calc_resistances(path_taken, start, end, material, cell_thickness,
        mat_network)
    return res_mat


def show_path(start, path, matrix):
    steps = np.diff(np.transpose(np.asarray(path)))
    steps_trans = np.transpose(steps)

    cursor_loc = start
    matrix[cursor_loc[0], cursor_loc[1]] = -matrix[cursor_loc[0], cursor_loc[1]]
    for step in range(0, int(np.size(steps) / 2)):
        cursor_loc = cursor_loc + steps_trans[step]
        matrix[cursor_loc[0], cursor_loc[1]] = -matrix[cursor_loc[0], cursor_loc
            [1]]
```

```
def calc_resistances(path, path_start, path_end, material, cell_ht, mat_network):
    steps = np.diff(np.transpose(np.asarray(path)))
    steps_trans = np.transpose(steps)
    p_orth = np.transpose([steps[1], -1 * steps[0]])
    mat_resistance = np.zeros(mat_network.shape)

    rho = material_rho_lookup(material)

    path_loc = path_start
    for step in range(0, int(np.size(steps)/2)):
        cursor_loc = path_loc
        trace_width = find_trace_width(cursor_loc, mat_network, steps_trans[step],
            p_orth[step])

        # with width calculated, assign all cells along path to this width or
            resistance
        this_resistance = rho / (trace_width * trace_width * cell_ht)
        set_res_for_trace_width(cursor_loc, this_resistance, mat_resistance,
            mat_network, p_orth[step])

        # move to next step
        path_loc = path_loc + steps_trans[step]

    # for each cell in network that doesn't have a resistance, find the closest
        cell that does and set equal to that
    no_res_cells1 = mat_resistance == 0
    no_res_cells2 = mat_network == 1
    no_res_cells = np.argwhere(no_res_cells1 * no_res_cells2)
    res_cells = np.argwhere(mat_resistance != 0)

    for no_res_cell in no_res_cells:
        idx = np.linalg.norm((res_cells-no_res_cell), axis=1).argmin()
        this_res_loc = res_cells[idx]
        mat_resistance[no_res_cell[0], no_res_cell[1]] = mat_resistance[
            this_res_loc[0], this_res_loc[1]]

    clipped_res_mat = clip_res_matrix(mat_resistance, mat_network, path_start,
        path_end, steps)
    smoothed_res_mat = smooth_matrix_non_zero(clipped_res_mat, 10)

    return smoothed_res_mat


def find_trace_width(path_loc, mat_network, step, p_orth):
    trace_width = 1
    cursor_loc = cell_at_edge_of_width(path_loc, mat_network, p_orth)

    # move in opposite direction until out of trace counting to find width
    while mat_network[cursor_loc[0], cursor_loc[1]] == 1:
        cursor_loc = cursor_loc - p_orth
        # if step is diagonal, add diagonal distance to trace width
        if np.linalg.norm(step) > 1:
            trace_width = trace_width + np.sqrt(2)
        else:
            trace_width = trace_width + 1
    return trace_width


def set_res_for_trace_width(this_location, this_resistance, mat_resistance,
    mat_network, p_orth):
    edge_cell = cell_at_edge_of_width(this_location, mat_network, p_orth)
    cursor_loc = edge_cell
    # with width calculated, assign all cells along path to this width or
        resistance
    while mat_network[cursor_loc[0], cursor_loc[1]] == 1:
        mat_resistance[cursor_loc[0], cursor_loc[1]] = this_resistance
        cursor_loc = cursor_loc - p_orth


def cell_at_edge_of_width(start_loc, mat_network, p_orth):
    # move from the current location in the path orthogonally until out of trace
    cursor_loc = start_loc
    while mat_network[cursor_loc[0], cursor_loc[1]] == 1:
        cursor_loc = cursor_loc + p_orth

    # move back one location, start trace width count
    cursor_loc = cursor_loc - p_orth
    return cursor_loc


def clip_res_matrix(in_matrix, mat_network, path_start, path_end, steps):
    steps_trans = np.transpose(steps)

    out_matrix = in_matrix

    # start location
    cursor_loc = path_start
    step = 0
    cursor_loc = cursor_loc - steps_trans[step]
    start_step = np.rint(np.mean(steps_trans[1:5], axis=0))
    start_step = start_step.astype(int)
    p_orth = [start_step[1], -1 * start_step[0]]
    cursor_loc = cursor_loc - start_step
```

```
# check if the step is diagonal, if so, split up step movement to two half
    steps keeping orthogonal direction
if np.prod(start_step) != 0:
    half_step_1 = [start_step[0], 0]
    half_step_2 = [0, start_step[1]]
    while mat_network[cursor_loc[0], cursor_loc[1]] == 1:
        set_res_for_trace_width(cursor_loc, 0, out_matrix, mat_network, p_orth)
        cursor_loc = cursor_loc - half_step_1
        set_res_for_trace_width(cursor_loc, 0, out_matrix, mat_network, p_orth)
        cursor_loc = cursor_loc - half_step_2
else:
    while mat_network[cursor_loc[0], cursor_loc[1]] == 1:
        set_res_for_trace_width(cursor_loc, 0, out_matrix, mat_network, p_orth)
        cursor_loc = cursor_loc - start_step

# end location
cursor_loc = path_end
step = len(steps_trans) - 1

end_step = np.rint(np.mean(steps_trans[-5:-1], axis=0))
end_step = end_step.astype(int)
p_orth = [end_step[1], -1 * end_step[0]]
cursor_loc = cursor_loc + end_step
# check if the step is diagonal, if so, split up step movement to two half
    steps keeping orthogonal direction
if np.prod(end_step) != 0:
    half_step_1 = [end_step[0], 0]
    half_step_2 = [0, end_step[1]]
    while mat_network[cursor_loc[0], cursor_loc[1]] == 1:
        set_res_for_trace_width(cursor_loc, 0, out_matrix, mat_network, p_orth)
        cursor_loc = cursor_loc + half_step_1
        set_res_for_trace_width(cursor_loc, 0, out_matrix, mat_network, p_orth)
```

```
            cursor_loc = cursor_loc + half_step_2
    else:
        while mat_network[cursor_loc[0], cursor_loc[1]] == 1:
            set_res_for_trace_width(cursor_loc, 0, out_matrix, mat_network, p_orth)
            cursor_loc = cursor_loc + end_step

    return out_matrix



def smooth_matrix_non_zero(in_matrix, radius):
    [nrow, ncol] = in_matrix.shape
    out_matrix = np.zeros_like(in_matrix)

    for row in range(0, nrow):
        for col in range(0, ncol):
            mat_val = in_matrix[row, col]
            if in_matrix[row, col] > 0:
                nei_row_start = max(row - radius, 0)
                nei_col_start = max(col - radius, 0)
                nei_vals = [in_matrix[row, col]]
                for nei_row in range(nei_row_start, min(row + radius, nrow)):
                    for nei_col in range(nei_col_start, min(col + radius, ncol)):
                        if not (nei_row == row and nei_col == col):
                            if in_matrix[nei_row, nei_col] > 0:
                                nei_vals.append(in_matrix[nei_row, nei_col])
                ave_val = np.average(np.asarray(nei_vals))
                out_matrix[row, col] = ave_val

    return out_matrix



def material_rho_lookup(material):
```

```
# https://www.thoughtco.com/table-of-electrical-resistivity-conductivity
    -608499
switch = {
    'Copper': 6.61e-4,
    'Aluminum': 8.98e-4,
    'Gold': 9.61E-04,
    'Silver': 6.26E-04,
    'Nickel': 2.75E-03
}
# units: ohm mil
return switch.get(material)
```