AUBURN UNIVERSITY

SAMUEL GINN
COLLEGE OF ENGINEERING

SENIOR DESIGN: OPERATIONAL READINESS REPORT

TEAM 8: 3D MODELING OF DETACHED METAL WHISKERS

By

MANAGER: GRAHAM DUKE
SCRIBE: CONNOR MASSEY
ANDREW SMITH
MEREDITH OSBORNE

SPONSOR: DONNA HAVRISIK, ERROL REID
TECHNICAL ADVISOR: DR. GEORGE FLOWERS

TABLE OF CONTENTS

Title Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF OBJECTS

Object Page

LIST OF ABBREVIATIONS

1D	One-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional
CAD	Computer Aided Design
CCA	Circuit Card Assembly
Freq.	Frequency
IBM	International Business Machines Corporation
MCS	Monte Carlo Simulation
MDA	Missile Defense Agency
NASA	National Aeronautics and Space Administration
PCB	Printed Circuit Board
Qty	Quantity
SEM	Scanning Electron Microscope
UI	User Interface
WP	Working Principle

CHAPTER 1
INTRODUCTION

In nearly every industry across the globe, there has been a strange and disastrous phenomenon known as metal whiskering that has long been a severe issue in circuit cards. This is when a soft metal such as zinc, cadmium, or tin spontaneously sprouts microscopic, filament-like structures that can be broken off due to external forces, and potentially cause damage. The threat can range from one whisker to thousands, creating conductive bridges across electrical components which compromise the reliability and longevity of electronic systems. The problem can become especially severe in the aerospace and defense industries, due to the importance of a safe and successful mission and the critical need for reliable electronics [1].

Metal whiskering has been a known phenomenon since the second World War, however the exact cause of growth is still unknown. It has been theorized that certain aspects of the environment can persuade the whiskers to grow. This includes variation in gravitational acceleration, pressure, temperature, mechanical stress and strain, and forces such as prolonged propulsion or heavy impacts. These same forces can also cause the whiskers to detach, leaving them airborne, and potentially falling directly onto the CCA. The damage range can be as small as brief, unnoticeable short circuits to a more excessive high-voltage electrical arcing, which can destroy the CCA. Current preventative measures to avoid whisker growth include the application of special coatings and surface finishes, as well as mixing the metal with other elements to create a more controlled alloy. A concise solution has yet to be found [1].

While whiskers can grow in various metals and dimensions, tin whiskers are most commonly seen in industry when evaluating failures. Tin is a material that is heavily used in CCAs either as a surface coating or when soldering connections on electrical boards. The whiskers can then grow unpredictably and unnoticed out of the tin-plated material. While they

are small in diameter, the lengths they can reach are part of the difficulty when forming ways to combat the issue. They tend to only be a few micrometers in diameter, and up to 25 millimeters (0.98 in.) in length, making them hard to detect. This allows them to reach and create conductive bridges across many different areas on a circuit card, adding to the challenge for engineers attempting to resolve the issue [1].

Metal whiskers in general are incredibly difficult to predict, detect, and account for in designs. In an electronically driven world, the problem will only become more prominent as new technology is developed. The goal of this project is to design and construct a program, utilizing the Unity Physics Engine, that will produce a three-dimensional modeling software that simulates detached whiskers falling onto the user's CCA. The Unity Engine will be used to visually determine whether a connection was made between two or more critical components on the circuit card and statistically assess the chance that bridging will occur using Monte Carlo simulation techniques. In this program, the user will import a 3D rendering of their desired CCA model and define all key boundaries. A user interface will display customizable options for whisker and environmental properties to best simulate detached whisker behavior in the real-world. Although tin is the primary focus in metal whiskering, the user will be able to select various types of whisker-producing metals to better analyze and understand whisker behavior. The success of this program will benefit future teams of engineers to help identify the risks of metal whiskering and mitigate critical failures when designing new CCAs or improving older ones.

CHAPTER 2
DESIGN SPECIFICATIONS

To begin the design process, it is important to first understand the constraints that are present, relate them to an overall objective, and detail the desired mission environment. This section focuses on the range of factors that must be considered to deliver a feasible and functional solution for modeling detached metal whiskers in various potential conditions.

Project Constraints

The constraints for this project include a simplification of the whisker model for smoother simulation, convenience for future users, and working around potential limitations of the Unity Engine. Constraints were given by project sponsors Donna Havrisik and Errol Reid.

- Simulation must model the whiskers as simple cylinders.
    - While whiskers are varying in shape, the simulation will account for a more generic approach to the whiskers in making them cylindrical. The user must be able to input diameter and length distribution parameters to control a randomized set of whiskers to be dropped.
- Simulation should not include whisker growth.
    - The whiskers will be modeled as already detached and free flowing.
- The program must be convenient for users to input their own data.
    - Any user should be able to decide exactly what they want to model regarding the whiskers. Properties such as number of present whiskers, range of lengths, range of diameter, and material properties should be customizable options.
- The interface should be able to take 3D models of CCAs and import them into the Unity Engine for simulation.

    - Users will be able to upload their own models to the program and run simulations to get the most accurate results.
- The program should be able to simulate various environmental effects using the Unity Engine.
    - Various environmental factors will contribute to the landing locations of the whiskers, allowing customization will produce more accurate results. These include gravitational acceleration, vibration, and shock.
- Statistical analysis will be conducted through Monte Carlo simulation techniques.
    - Input data from the user will need to be able to be varied and stored so that the user can analyze probability of certain parameters causing short circuits on the CCA.
- The design must consist of one code and one script language.
    - Users will need to be able to refine the code should they decide to make changes.

Objective

The main objective of this project is to create a program that will run detached whisker simulations to any arrangement of a CCA CAD model imported by the user and under any given circumstances. Users must be able to input specified whisker characteristics, dropping formation and location, boundary conditions, and environmental criteria to fully define their desired simulation atmosphere. The simulation will report back to the user whether a bridge was created and, utilizing the Monte Carlo Simulation technique, assess the risk probability under various parameters on the given CCA.

Mission Environment

The mission environment that this project would be used in varies with each system. The desired design for the program and simulation is to be able to handle these types of changes and, through the user's input, recreate the changes to produce a more accurate model for the risk associated with potential whisker bridging. Environments were specified by the project sponsors:

- Still, controlled areas such as inside computer cases.
- Whiskers will be modeled as initially detached and airborne.
- High vibration and impact environments.
    - Missile and rocket launches; vehicles on rough terrain.
- Satellites in low Earth orbit.
    - Radiation becomes increasingly problematic. The low gravity would also cause whiskers to behave differently than expected on Earth.
- Varying temperature climates
    - Temperature cycling due to exposure to the Sun and deep space or being located near high temperature sections of machinery.
- Interplanetary travel and varying gravity levels across the solar system.
    - Program should consider if a specific CCA is exposed to extreme or low gravity environments and how it would affect the whiskers.

CHAPTER 3
BACKGROUND INFORMATION

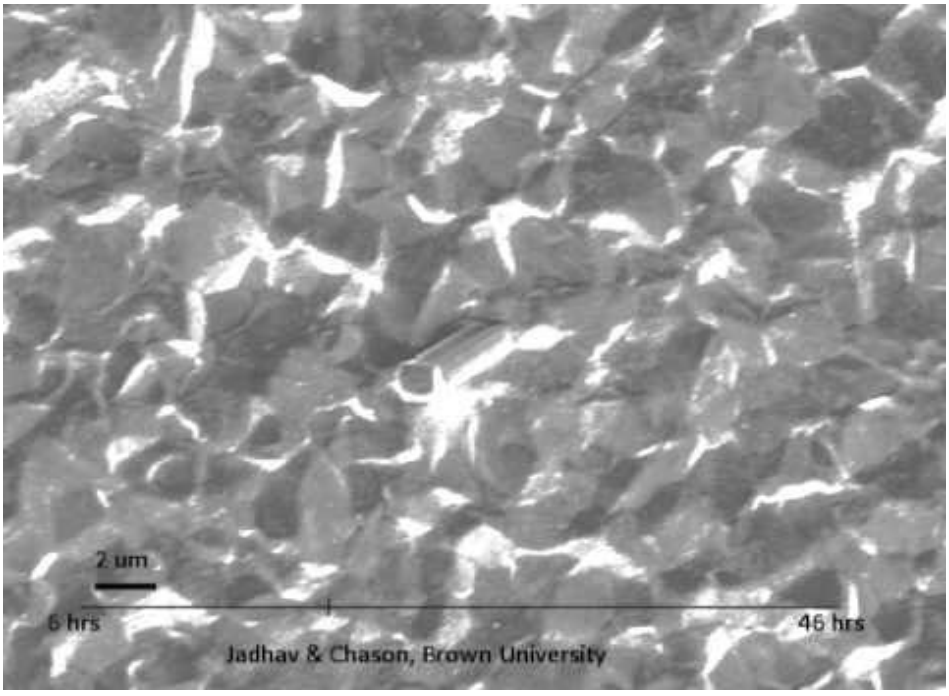
Metal whiskers are a phenomenon that have caused countless electronic failures for nearly a century, yet research on their reason for forming and measures of predictability continue to this day. This section discusses current knowledge on formation, historical cases of exposure, current solutions in avoiding failure, and virtual applications to estimate failure risks.

Whisker Formation

Despite the first reports of metal whiskers being published back in the 1940's, there is still no definitive answer as to why the phenomenon of metal whiskering occurs. Current theories state that compressive stresses applied to the metal may be the main cause of whisker growth. Separate factors can also add to the already applied compressive stress, which in turn increases the risk of whisker growth. These factors include the chemical composition of the metal, the bending of the metal, scratches, or other surface imperfections, and differences in the coefficient of thermal expansion between the metal and its surroundings [2].

Other theories suggest the growth of metal whiskers to be a result of imperfections in the metal's granular structure [2]. When a whisker is growing, long range diffusion of atoms provides the material used for the whisker. Many tests have proven this, revealing that the atoms making up the whisker come from further away, rather than the atoms closely surrounding the growing whisker. The process of an atom's diffusion can occur in two ways: Bulk diffusion and grain boundary diffusion. Bulk diffusion, typically occurring at higher temperatures, is a slow process in which atoms move through the lattice structure of the material and accumulate towards the tip to produce a whisker. Grain boundary diffusion, common at lower temperatures, is a faster process of atoms moving through grain boundaries between the grains and migrating towards the tip to form a whisker [1]. Object 1 reveals a timelapse of a tin whisker's formation.

Following this is Figure 1 representing a sample of zinc whiskers formed on a zinc-plated steel surface, sent to the team by NASA engineer Jay Brusse.



Object 1: Timelapse of a Tin Whisker Forming - YouTube [3]

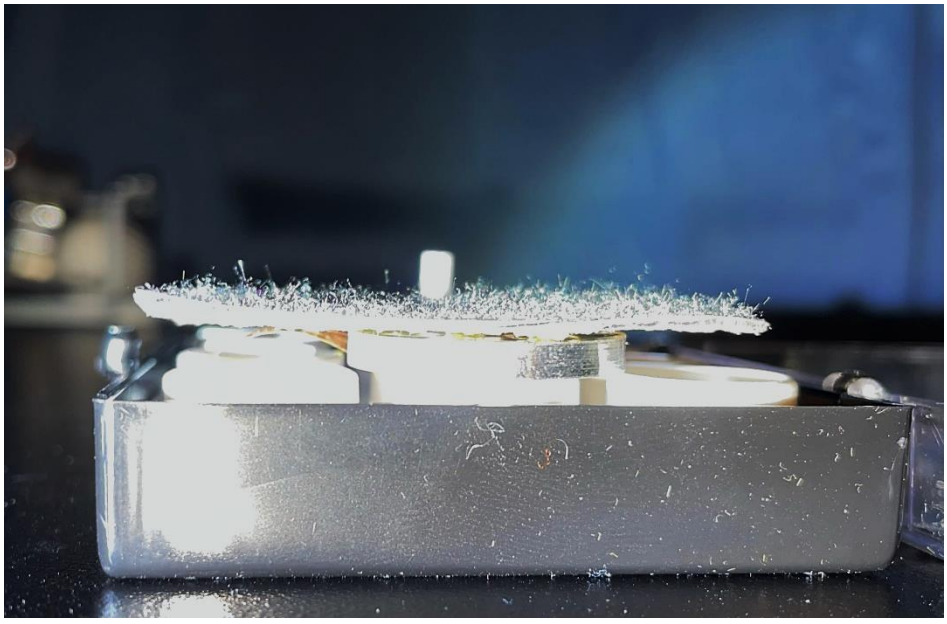


Figure 1: A Sample of Zinc Whiskers

Whiskers can grow on any area where the metal is applied to the surface [4]. Tin whiskers are a common occurrence in electronics due to tin being used as solder in circuit cards. This high rate of tin usage heavily contributes to an increased risk of whiskers growing from tin solder in a circuit card. Furthermore, as circuit cards and electronics become smaller, there is less distance separating circuit card components, which in turn makes it easier for a tin whisker to create a bridge in the circuit card and potentially resulting in catastrophic damage depending on the circuit card's function [2]. Figure 2 represents an example of a circuit card and its many soldering points.

Figure 2: Soldering of a Circuit Card [5]

Research and History

As previously mentioned, the first documented reports of metal whiskers were published in the 1940s, when it was noticed that cadmium whiskers caused repeated failure of electrical components from their application as a surface coating. After it was discovered that cadmium had a propensity to develop whiskers, a switch to the use of tin electroplating began in 1948. However, it was quickly noted that tin acted very similarly to cadmium, developing whiskers

under the same conditions. Methods to help stem the occurrence of metal whisker growth were first introduced in the 1950s, with some of the developed methods still being in use today [6].

While not still commonly used today for health concerns, one method to decrease the chance of metal whiskering was to incorporate a portion of lead in the tin solder, creating a tin/lead alloy. In the 1950s, researchers Eloise Koonce and S. M. Arnold noted that a method of whisker growth is through the addition of material to the base of the whisker, rather than the tip. Tests were also performed by three researchers named R. M. Fisher, L. S. Darken, and K.G. Carroll. Results from their experiments speculated that applied compressive stresses increased both the occurrence of metal whiskers and the rate at which the whiskers grew—a theory that is still considered to this day [6].

Despite the decrease in metal whisker research in the 1960s, important discoveries were still made. One major discovery was that copper undercoats were effective in mitigating the chance of bright tin developing whiskers. In the 1970s, focus shifted back to researching mitigating methods. The first high quality images of whiskers began to be produced through the use of SEM microscopes [6].

The rate at which tin whisker research was conducted continued to decrease throughout the 1980s and 90s. An experiment performed by B. D. Dunn concluded that applied stresses have no effect on whisker growth, contradicting previous theories [6].

In the 2000s, research into both causes of tin whisker growth and mitigation techniques began to increase rapidly. The first three years of the 21st century had more articles published than the previous 60 years combined. The main cause for the drastic increase was due to the European Union’s requirement that lead must be removed from tin films. New techniques and technologies, such as focused iron beam microscopy and micro-focus X-ray diffraction were

developed to help researchers understand more about the microstructure of tin, and how the internal stress level of the metal might affect whisker growth [6].

PanAmSat Galaxy VII

In the past, there have been numerous recorded incidents caused by metal whisker growth. One such incident was the failure of the Galaxy VII satellite. Built by PanAmSat and launched in 1992, the satellite was expected to have a service life of approximately 12 years. However, on November 22, 2000, both satellite's processors failed, causing the satellite to drift off course. The change in course caused the solar arrays of the satellite to lose alignment with the sun, which in turn caused the satellite to lose power. The cause of the processors failing was found to be attributed to metal whisker growth, which caused a short in the satellite's circuit card. The Galaxy IV satellite, launched before the VII, experienced the same failure of its second processor due to metal whiskering. Shown below in Figure 3 is an artist's concept art of the Galaxy VII satellite [7].

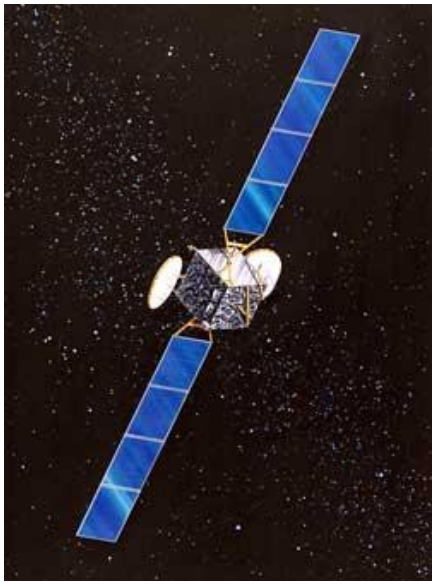


Figure 3: An Artist's Concept of the Galaxy VII Satellite [7]

Millstone Nuclear Reactor Shutdown

Another issue caused by metal whiskers was the failure of a nuclear power plant's safety system. In 2005, the Millstone nuclear power plant in Waterford, Connecticut experienced a sudden shutdown of a nuclear reactor. An alarm was triggered by the reactor's safety system, due to an apparent drop of steam pressure in the reactor. The company shut down the operation immediately. It was later discovered that the pressure of the steam had not actually dropped. The circuit card used for the safety system was inspected for damage, but no signs were found. Upon microscopic inspection, a thin tin whisker was spotted creating a bridge between two components on the circuit card. It was then determined that the metal whisker induced a short, which in turn caused the false pressure reading to occur, triggering the shutdown of the entire reactor [8].

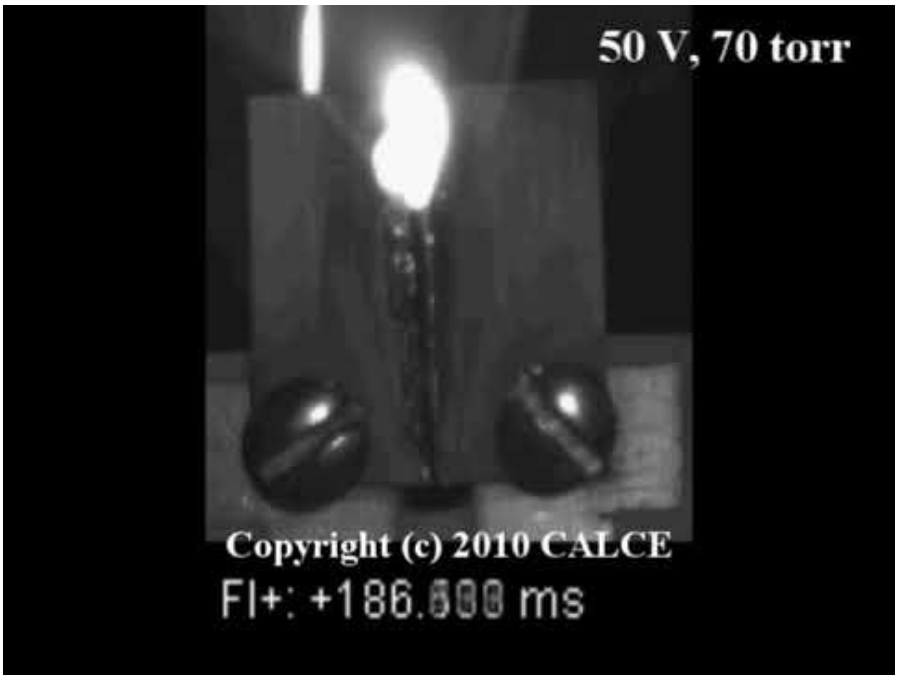
Failure Modes

When metal whiskering occurs in a CCA and results in a conductive bridge between components, there are a few possible outcomes. The first outcome is mechanical contact occurring without an electrical current moving through the bridge. In this scenario, it is most likely that no damage to the CCA will occur. Electrical current that would flow through the bridge is impeded by electrically insulating films. The insulating films can successfully block the flow of electricity unless the film's dielectric strength is broken down by the amount of applied voltage [1]. Dielectric strength of a material is a measure of how much voltage the insulator can withstand without the insulated material becoming conductive [9].

The next possible outcome is an electrical current being established after mechanical contact occurs, with the current running through the whisker being less than the current required

to melt the metal whisker. In this instance, a continuous short of the circuit card occurs. In the case that the current running through the bridged whisker is greater than the current required to melt the whisker, intermittent shorts will occur in the circuit card [1].

The final possible outcome is sustained metal vapor arcing. This outcome occurs either when the current applied to the whisker is large enough to vaporize the metal, or when the voltage is large enough to ionize the metal gas. The voltage required to sustain metal arcing increases as the distance between the two points increases [1]. Shown below is a video of tin whisker induced metal vapor arcing, labeled Object 2.



Object 2: Tin Whisker Induced Metal Vapor Arcing - YouTube [10]

Current Solutions

As previously mentioned, whiskers can be detrimental in a variety of applications. Several solutions are used today to help combat this problem. One involves the use of conformal coating to cover the board. Conformal coating covers the surface where whiskers can grow,

preventing them from breaching the layer. In a study, brass was used as a base medium. Different types of coatings were tested, but brass found success [11, 12]. Parylene and silicone were two other coatings tested in this experiment that also proved to mitigate whisker growth [11].

Another potential solution to this problem is simply cleaning the circuit cards. This idea was analyzed during a study where boards both cleaned and uncleaned were analyzed after five hundred hours. The boards were held at a temperature of 85°C with a set relative humidity. After the time had elapsed, the circuit cards that had been cleaned revealed a shorter length of whiskers than those that were unclean [11]. In any application, shorter whiskers logically have a smaller chance of causing more trouble if they are not able to reach as much of the board assembly.

Adding other material to tin solder to create an alloy has been shown to reduce the chance of whisker formation. Lead and bismuth are both common elements added to tin in the electronics industry. A 60/40 composition of tin/lead alloy was frequently used between 1950-2000 due to its reduction in the chance of whisker growth [6]. The U.S. military has some specifications mentioning a use of 3% by weight lead in alloy composition. Unfortunately, due to health concerns, the electronics industry has slowly moved away from using lead. Bismuth, however, can safely be used in a low quantity and is a common alloy used today. Rockwell and IBM products require the use of at least 0.3% by weight to help reduce whiskers formation [13].

Previous student teams have also accepted the challenge of developing a similar program to the one that this team has been tasked with. 2D and 3D modeling programs from various universities have seen success in simulating whisker behavior. The team will take a new look into creating a prototype simulation.

Simulation and Software

Unity Game Engine

Unity Game Engine is the modeling software that will be used in this project. Compared to similar engines, Unity is easier to learn and apply when looking to modify and replicate real-world physics [14]. In a meeting with team advisor Jake Botello on February 9, 2024, Botello stated that Unity works well with Microsoft Visual Studio, which provides compatibility in giving the user a separate terminal to work in when modifying and developing code. Figure 4 shows an example of the 3D game development window in the Unity system.

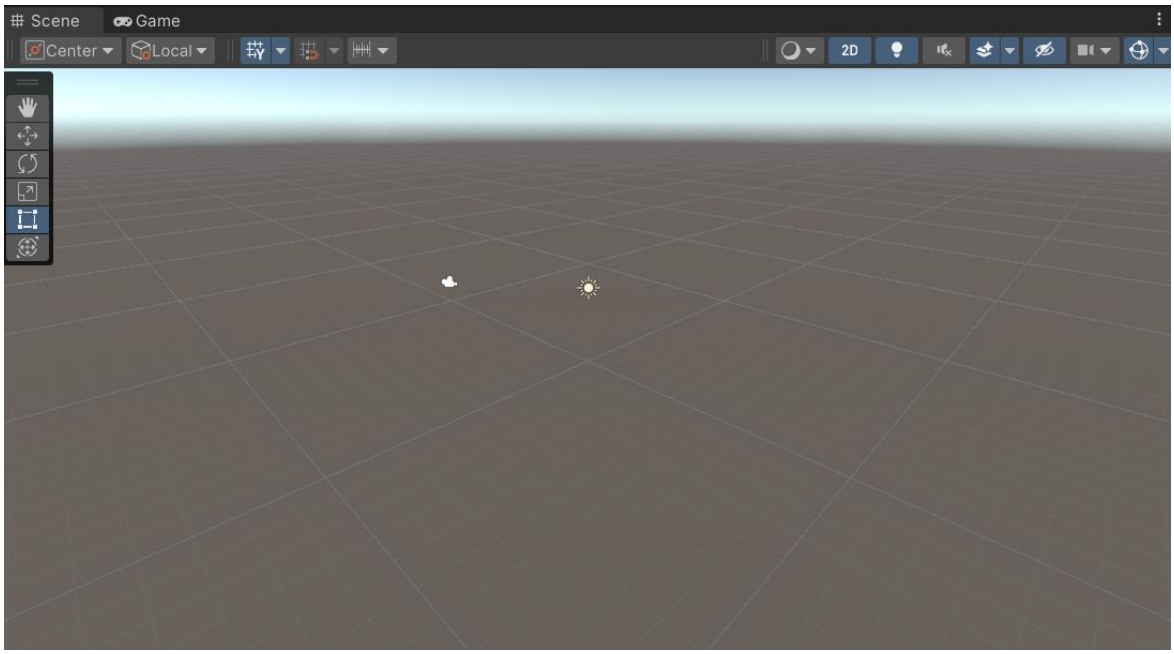


Figure 4: Unity 3D Engine View Window

More capabilities exist around this viewing window to allow the user to create and/or edit objects within this environment. Users can import their desired CCA model into a 3D space and select specific inputs for the simulation to run by. Its physics engine provides a real-world simulation, including the ability to modify gravitational acceleration, an object's mass and

friction, and drag force. As for coding language, Unity supports a few selections, including C#, IronPython, and Rust. C# will be used in the development of the program due to its object-oriented programming language [15]. This means that user inputs can be directly transferred to the back end of the code to make the necessary changes for the particular simulation run. The inputs can also be stored in the program to make necessary calculations.

Monte Carlo Simulation

Monte Carlo Simulation is a type of simulation process that is typically used in situations where variables are randomized and unknown. These simulations have applications in a variety of different fields, including finance estimation, risk assessment, and decision making. In such a simulation, inputs are iterated, and their results are stored. The combination of these results forms an output distribution. Through a degree of randomness in the system, gathered data can then be averaged to obtain a generalized expected outcome [16]. This allows the probability and likelihood of an occurrence to be estimated due to various influencing parameters.

For purposes of this project, the technique will be used to predict short circuiting probability as user input changes. Sets of whiskers will be released onto the imported CCA model in the simulation, providing a degree of randomness to the probability calculation. Figure 5 displays results from a conducted experiment measuring whisker dimensions obtained from a sample of tin-plated brass, which was left in ambient storage for approximately 11 years [1]. In total, 187 detached whiskers were taken from random locations on the sample and measured by their length and thickness. Results from this experiment revealed a lognormal distribution of whisker dimensions, represented by the following figure [17]. Therefore, in applying MCS, lognormal distributions can be applied to the whisker dimensions to accurately represent detached whiskers encountered in the real-world.

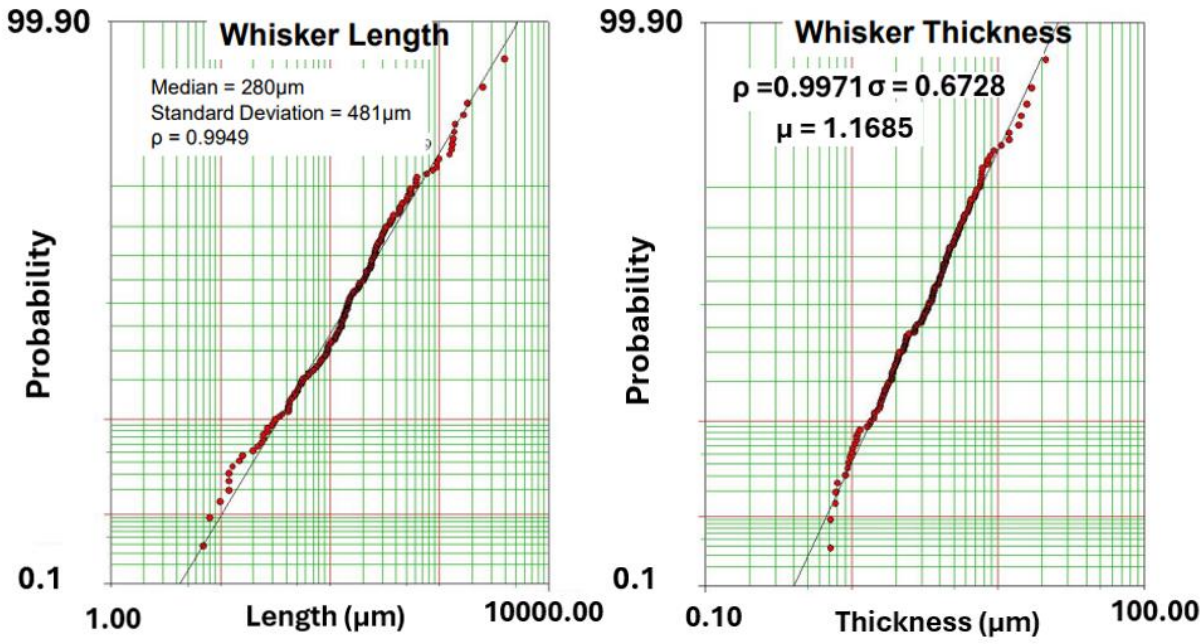


Figure 5: Probability of Whisker Dimension Growth [1]

Governing Equations

To properly execute a desired simulation environment under various input parameters, it is necessary to define equations for these parameters and any forces present. As the program acknowledges an input of lengths, diameters, and a material selection, equations and material property values embedded into the program will perform various calculations in the background. The following calculations will influence the nature and interaction of the whiskers as they perform in simulation. With a stored value of density (ρ), the mass (m) of each whisker can be determined from Equation (1):

$$m = \rho \cdot [\pi \cdot (d/2)^2 \cdot L] \quad (1)$$

where d is the whisker diameter and L is the length of the whisker. With a stored value of the metal's coefficient of friction (μ_f), the friction force of the whisker (F_f) can be determined from Equation (2):

$$F_f = \mu_f \cdot F_n \tag{2}$$

where F_n is the normal force of the whisker on its surface, defined by Equation (3):

$$F_n = m \cdot g \cdot cos(\theta) \tag{3}$$

where g is the gravitational acceleration, and θ is the angle (in degrees) of the whisker relative to the CCA surface. Lastly, with a stored value of the material resistivity (ρ_m), (R) of each whisker can be determined from Equation (4):

$$R = \frac{\rho_m \cdot L}{(\pi/4) \cdot d^2} \tag{4}$$

More equations will continue to be explored in the design process as the team navigates the various potential of parameter manipulation available in Unity's 3D Physics engine.

Future Applications

As mentioned previously, metal whiskers present a large problem to engineers and consumers alike. Even though they are small, they can cause significant failures in a variety of different systems. However, the success of the team's solution would allow engineers to identify modes of failure to mitigate any potential risks. This system could then be applicable in a variety of environments. Companies producing and handling precious equipment, such as MDA and NASA, can import their designed CCA models and assess detached whisker behavior should their devices produce whiskers. By doing so, unintentional and potentially catastrophic mission failures due to whisker-induced malfunctioning can be minimized. The program can even be modified for various environments, with applicable forces and climates taken into consideration.

This can benefit situations where the CCA is introduced to various planets or levels of the earth, with changing gravitational accelerations, pressure levels, and temperatures that are not common.

CHAPTER 4
DESIGN CONCEPTS

The desired outcome of this project is to develop a 3D program in Unity that allows users to import their CCA model, visually evaluate conductive bridges from spawned whiskers, and statistically analyze bridging frequency and probability. This requires several individual inputs due to the various real-world circumstances where bridging can occur. This section breaks down the function of the program, user inputs and returned outputs, and an analysis of various operational methods. A final operational method is determined based on these results.

Functional Decomposition

There are many variables to consider when simulating metal whisker behavior, such as whisker dimensions, drop location, boundary constraints, and component arrangement on the CCA model. Whiskers can also grow in small or large quantities, affecting the likelihood of bridging. Their small size makes it difficult to view, and their lightweight nature means that they do not fall directly down from the point of detachment. Therefore, it is necessary to consider these variables in the modeling program to ensure that accurate results can be properly evaluated in predicting conductive bridging probabilities. To do so, both the input parameters and output simulation must respectfully represent the user's desired testing atmosphere.

Input parameters are related to characterizing the whiskers, specifying necessary conditions, and importing the CAD model for testing. In characterizing whiskers, defining properties such as length, diameter, material, and unit type must all be considered. For practical simulation purposes, it is key that the user enters values for μ and σ for both length and diameter of the whiskers. This specifies the dimensional distribution of whiskers. Arrangement is also a key input that defines the number of whiskers and how they are placed prior to dropping. The

location of where these whiskers are dropped also plays a factor in simulation results, as components in CCA's can come in various arrangements. Conditions that affect whisker behavior are the boundary conditions that limit their movement and external forces they will be subjected to, such as gravity, vibration, and mechanical shocks. Importing a CCA model is also necessary so that the user can address the potential effect of whiskers on their design.

The output simulation allows the user to visually analyze bridging results and statistically evaluate bridging frequency. To visually analyze results, it is necessary for the bridged area to be highlighted on the CCA model. It is also valuable to display a percentage of the probability that the generated whiskers cause a bridge in the current simulation for visual justification. Monte Carlo Simulation will be available as the program runs background calculations, so that the user can view graphs based on any input parameters they vary.

Figure 6 shows a functional decomposition for the designed program to be evaluated by.

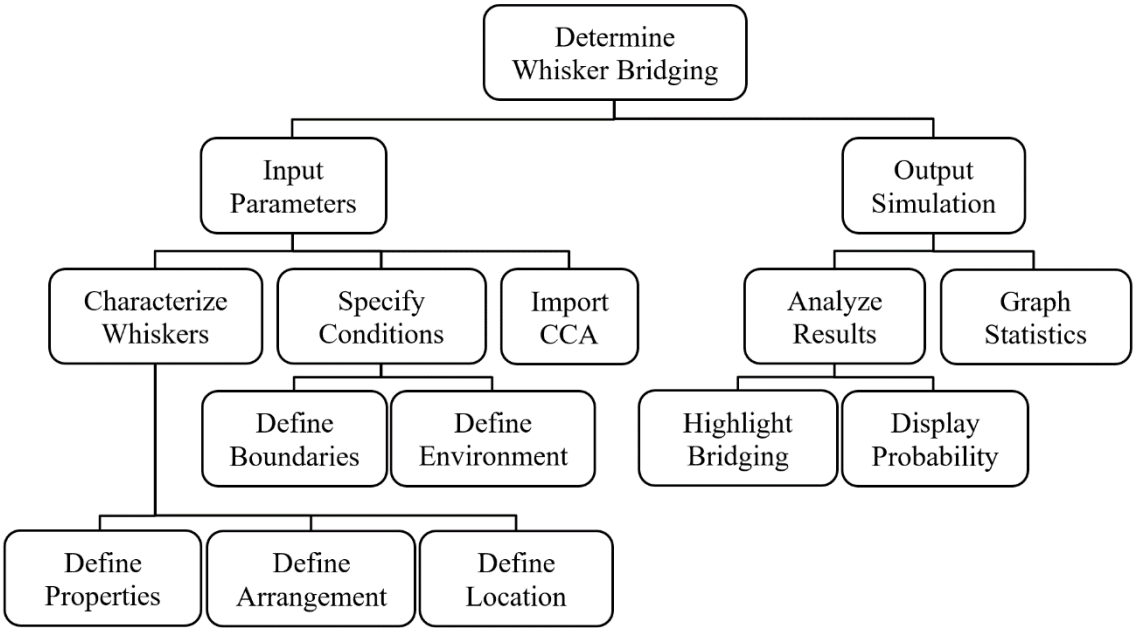
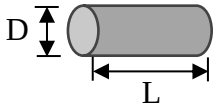
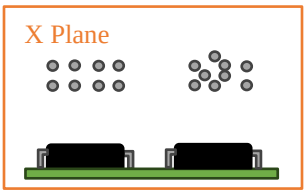
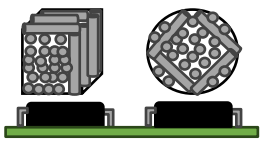
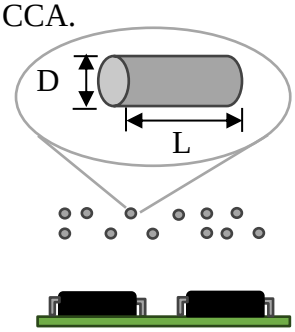


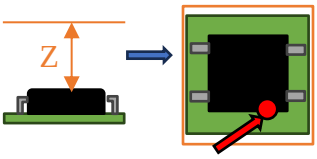
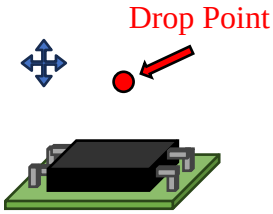
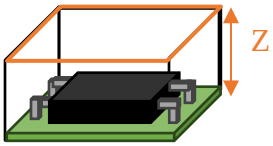
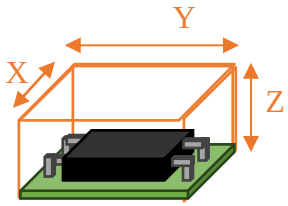
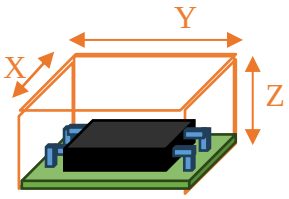
Figure 6: Functional Decomposition of Metal Whisker Program

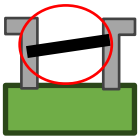
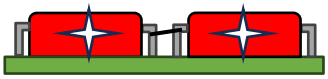
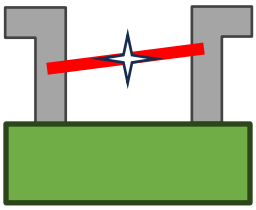
Morphological Matrix

A morphological matrix was created to analyze the design of various user interfaces to match user satisfaction for their specific application. Table 1 represents a variation of Working Principles (WP) that were considered for each subfunction in the functional decomposition.

Table 1: Morphological Matrix

Subfunction	WP1	WP2	WP3
Define Properties	**Entered Properties** User enters desired whisker dimensions and material properties, responsible for units (Imperial/Metric, scaling). Length μ: 10 mm σ: 0.05 mm Density: 7.31 g/cm^3	**Selection of Properties** User selects unit type (Imperial/Metric), whisker dimensions, and material from a list with properties stored. Unit Type: Metric Length: μ: 10 mm σ: 0.05 mm Material: Tin	**Manual Properties** User selects unit type (Imperial or Metric), draws whisker(s), and enters material properties. Unit Type: Metric Density: 7.31 g/cm^3 
Define Arrangement	**Automated Arrangement** User can enter 2D array(s) for whiskers to be generated normal to X, Y, or Z plane. Spacing is determined by user. Arrays can be randomized. 	**Clumped Arrangement** User defines number of whiskers to generate, which spawn and form within a volume (cube or sphere) to be dropped. 	**Manual Arrangement** User draws (can copy and paste) whiskers to desired locations around CCA. 

Subfunction	WP1	WP2	WP3
Define Location	**Coordinate Location** User enters the coordinates (x,y,z) for the drop point of whiskers based on an origin. Model displays preview before confirmation. X Coordinate: 8 cm Y Coordinate: 9 cm Z Coordinate: 4 cm	**Top-Down Location** User defines distance from a surface parallel to the Z plane. User view becomes normal to Z plane to click and drag specific drop point. 	**Manual Location** User will click and drag the specific drop point in the 3D model. 
Define Boundaries	**Automatic Boundaries** After importing CCA model, program creates X and Y plane boundaries, and asks user for Z plane boundary. Program acknowledges CCA components locations. 	**Semi-auto Boundaries** After importing CCA model, user enters X, Y, and Z plane boundaries from origin to surround model. Program acknowledges CCA components. 	**Manual Boundaries** After importing CCA model, user enters X, Y, and Z plane boundaries from origin to surround model. User is asked to highlight and describe CCA components. 
Define Environment	**Automatic Environment** User can select from a preset list of common environmental options that contain properties stored in program. Environment: (Select) Earth Mars Space	**Semi-auto Environment** User can select from a preset list of common environmental options and make changes to desired properties to fit needs. Environment: Mars Pressure: 6.52 mbar Temperature: 20°C	**Manual Environment** User is responsible for entering all environmental properties prior to simulation. Created environments can be saved for future use. Name: Test 1 … Save

Subfunction	WP1	WP2	WP3
Highlight Bridging	**Highlight Dataset** Whisker number(s) in dataset that did not cause a bridge are in black text. Whisker number(s) that bridged are in red text. Each whisker can be clicked on to view its location in model. Data Whisker #1 Whisker #2 Whisker #3 	**Highlight Component(s)** Component(s) in the CCA flash to indicate a short circuit. Clicking on the component shows whiskers attached to it. 	**Highlight Whisker(s)** Fallen whisker(s) in the CCA flash to indicate conductive bridging. 
Display Probability	**Text Object Display** A numerical value of the probability of bridging will be displayed in a text box in the game window for the finished simulation. 20% 	**Console Display** A numerical value of the probability of bridging will be displayed in the console below the game window.  Console Simulation 15: 20%	**Stored Display** Probabilities for each simulation will be stored in a separate program (such as Excel) to be accessed.  Microsoft Excel Simulation 13 0% Simulation 14 0% Simulation 15 20%

Preliminary Solution

Define Properties: Selection of Properties

One of the most historically critical mistakes to make in design is improper conversion of units. This can either be related to mixing Imperial and Metric units or forgetting to adjust values

to the proper scalar (i.e. millimeters to meters). While on the surface it seems a simple mistake, the outcomes could be devastating. By allowing users to deliberately select their unit system, and embedding scalar conversions into the software, this error is removed completely. It is also more feasible for the user to input variables for the whisker, such as length and diameter distributions, material, and quantity. This allows the software to acknowledge and create the specific set of whiskers on its own.

Define Arrangement: Automated Arrangement

While it is difficult to fully simulate the behavior of many whiskers in a confined space, automating their arrangement in arrays is the closest one can get to representing a large number of detached whiskers in real-world situations. It is impractical to generate a clump of whiskers to disperse over an area and can be monotonous to manually arrange them in space when looking to in large quantity. Arrays provide the benefit of quickly generating a formation of whiskers and ensure that the same whisker cylinder is evenly spaced to user desires. Random arrays also provide a closer approximation to real-world behavior of detached whiskers in a confined space, generating a random number of whiskers that are distributed with various distances between them.

Define Location: Coordinate Location

This working principle provides the benefit of giving the user full access to the three-dimensional space by entering the x, y, and z coordinates to specify a drop point relative to the CCA. The user will be able to see a preview of the generated whiskers in the view window as they rest at their defined location. This allows the user to make adjustments to the coordinates

before confirming their drop location. While the top-down location constrains the user to a specified distance parallel to the Z plane, this is not a common function available to Unity and would require deeper coding experience than necessary to perform the task. Manually locating the drop point is also not feasible as there are no constraints present, leaving it up to the user to fine tune their approximate location from different viewing angles.

Define Boundaries: Manual Boundaries

Due to the various applications of CCAs in industry and their own individual arrangement of circuit components, it is important to give the user freedom to determine the boundaries by which whiskers are contained to. Automated X, Y, and Z planes based off an imported CCA CAD model can be beneficial in certain circumstances. However, the user should be able to limit the boundaries and allow as much freedom or restriction of whisker projectile as much as they desire. This may provide the user with more information on whisker behavior in their specific model as they vary these boundary conditions. Furthermore, in asking the user to highlight and describe CCA components, the program is immediately aware of how the whiskers can affect the board. This includes highlighting resistors, capacitors, inductors and inputting corresponding values. This ensures that the program understands the components on the CCA that are present and creates collision detection points. The user also does not have to trust that the program acknowledges them on its own, as errors can arise. Automating this within the program would not be an easy task to perform due to variance in CCA board component arrangement.

Define Environment: *Automatic Environment*

CCAs are exposed to various types of environments. In giving the user a selection of

environments to choose from, properties of this environment can be automatically generated into the interface. This provides the user with a simple way to adjust the environment that their CCA encounters and quickly analyze the difference in results. While having the ability to change environmental parameters gives the user more accessibility, it is too tedious to make the user responsible for entering every property to the exact expectations. Properties such as pressure and temperature also do not affect the results of detached whiskers falling as much as gravitational acceleration does and are therefore obsolete in giving the user access to change.

Highlight Bridging: Highlight Whisker(s)

The arrangement of components in CCAs can be very complex, and in combination with hair-like cylinders spread around, it can be difficult to identify conductive bridging at a glance. However, the scale of the whiskers can be increased for visual representation, but still retain the same hitbox that a normal whisker should represent. Any whisker that causes bridging during the simulation can then be highlighted by changing its color. Users can then directly view this whisker and the conductive points it has bridged with. While access to a dataset that links to the bridging whisker seems beneficial, it can become tedious for the user to sift through larger sets of generated whiskers to find which resulted in bridging. Additionally, Unity does not provide a direct function to bring the user to a specific view based on an input; thus, requiring a deeper understanding of code than necessary. Highlighting the bridged component works, but without identifying the specific whisker(s) to cause the problem, it is difficult to determine which specific whisker was the culprit.

Display Probability: Text Object Display

The probability is a key value to display for simulation, as it represents the number of whiskers that bridge out of a distribution of a collection specified by the user. In keeping the simulation process clear and concise, it is beneficial to display the probability for the current simulation in the same location as the simulation itself (i.e. the game window). This value directly serves as a visual justification that the probability of bridging is represented by the visual bridges created on the circuit card seen while in the game window. Displaying this value in the command window requires the user to be able to escape out of the game window itself and traverse the Unity editor to navigate the console. Additionally, storing the probabilities for each simulation in a separate application, such as Microsoft Excel, is a tedious process when looking to access the file. It is more intuitive for the user to remain in the game window and not have to source their desired results externally.

Program Operation

The program's operation consists of two primary simulations: a physical simulation and a statistical simulation. The physical simulation represents both what the user can see and interact with relative to the 3D space, as well as background calculations in the script that determine physical whisker behavior. The statistical simulation represents Monte Carlo methodology in the form of graphs, similarly due to background calculations in the script that determine probabilities due to various inputs. Figure 7 reveals a simplified workflow of the operational procedure in the program to achieve these results for the user.

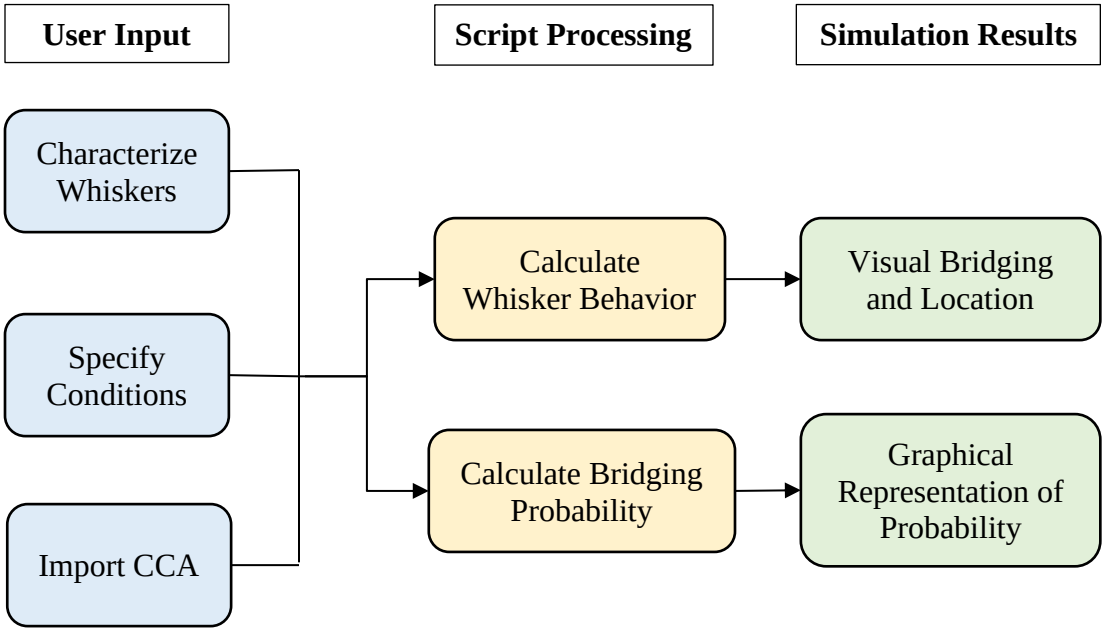


Figure 7: Program Operational Methodology

Physical Simulation

The user will start by creating a new file. The software will ask the user to import the CAD model of the desired CCA in a specific file format for it to be properly rendered into the program. An origin will appear at a corner of the model, and the user will be asked to enter X, Y, and Z distances from the origin to define the boundaries for simulation. The user will click any model components to be considered for simulation, and enter their respective values (resistance, capacitance, voltage, etc.). All values will be stored in the script and the simulation is then ready to be defined.

In defining the simulation, a window will open asking the user to enter the unit type, length/diameter mu and sigma values, quantity, material of the whisker, and number of iterations to simulate. The combination of whisker quantity, length/diameter mu, and length/diameter sigma allows the program to generate a random set of whiskers defined by these criteria. The

properties of this material, stored in the program, will be used in the script to determine the characteristics of the whiskers to properly simulate real-world behavior. These will be calculated using governing Equations (1) through (4), which will determine the mass, normal force, friction force, and resistance of each whisker. In the same window, the user will enter all environmental properties for the simulation to consider and turn on any environmental forces present. Further down, the user can generate several 2D arrays normal to a specified plane with desired dimensions and spacing between each whisker. Additionally, random 2D arrays can be generated by a provided whisker quantity from the user, creating an unorderly arrangement of whiskers. Values in creating the arrays will be stored within the program script.

The user will then identify the X, Y, and Z coordinates to locate a dropping point for the whiskers. A preview of whiskers ready to drop will be shown to the user and allow the user to fine tune their coordinates to their desired location. In confirming the coordinate inputs in the user window, the program can begin.

Depending on the CCA geometry, whisker characteristics, and drop location, the program will take time for all whiskers to find their landing. After completion, whiskers that have created a conductive bridge will highlight by changing to another color. This will allow the bridged whiskers to immediately stand out among whiskers that did not bridge and catch the eye of the user. A text object revealing the probability of the current simulation will be displayed in the corner of the game window to validate the success of the simulation. The dimensions of each bridging whisker will be stored and applied into graphical representation to view Monte Carlo results. Scrolling below the dataset will reveal an option to view Monte Carlo results based on the user's input parameters and the formulation used in the program script. This simulation file can be saved and accessed later at user leisure.

Statistical Simulation

The input data will be used in the background of the program as it simulates the physical probability and frequency of specific dimensions of detached metal whiskers landing and creating short-circuits in the model. Monte Carlo Simulation is known to find its strengths in randomization. Therefore, variation in the whisker characteristics and conditions, in conjunction with a large quantity of data recordings, will produce valuable results to the user. The following is a current list of preliminary simulations in applying Monte Carlo methodology to user inputs.

Simulation 1

Repeated simulation revealing the frequency that each length of whiskers from a distribution caused bridging. This will be a 2D bar graph measuring the frequency of each length (y-axis) vs. the length of whiskers that bridged (x-axis).

Simulation 2

Repeated simulation revealing the frequency that each diameter of whiskers from a distribution caused bridging. This will be a 2D bar graph measuring the frequency of each diameter (y-axis) vs. the diameter of whiskers that bridged (x-axis).

Simulation 3

Repeated simulation revealing the frequency that each length:diameter ratio of whiskers from a distribution caused bridging. This will be a 2D bar graph measuring the frequency of each ratio (y-axis) vs. the ratio of whiskers that bridged (x-axis).

Simulation 4

Repeated simulation revealing the frequency that each resistance of whiskers from a distribution caused bridging. This will be a 2D bar graph measuring the frequency of each resistance (y-axis) vs. resistance of whiskers that bridged (x-axis).

Simulation 5

Repeated simulation with only variation in drop location. This will be a 3D plot measuring the probability of short circuits (z-axis) vs. the length of the CCA (x-axis) and width of the CCA (y-axis). A color code legend will indicate the whisker length/range of lengths.

Table 2 represents each simulation and their respective variation, plot type, axes measurement, and legend (if necessary).

Table 2: Monte Carlo Simulations Due to Input Variation

Simulation Number	Variation	Plot Type	Axis Measurement			Legend
			X	Y	Z	
1	Whisker Length	2D	Length Range	Frequency of Bridging	N/A	N/A
2	Whisker Diameter	2D	Diameter Range	Frequency of Bridging	N/A	N/A
3	Whisker L/D Ratio	2D	L/D Ratio Range	Frequency of Bridging	N/A	N/A
4	Whisker Resistance	2D	Resistance Range	Frequency of Bridging	N/A	N/A
5	Drop Location	3D	CCA Length	CCA Width	Probability of Short Circuit	Color Range of Whisker Lengths

These simulations will help the user determine the frequency of specific whisker dimensions creating conductive bridges due to various user inputs, allowing them to better understand the consequences present relative to their CCA model. Results will vary depending on the user's CCA model, as the dimensions and arrangement of components on the circuit card will determine the likelihood that a detached whisker—of varying properties itself—creates a conductive bridge to short circuit the system. Monte Carlo Simulation will successfully aid in this random uncertainty and provide results that help to predict and eliminate failures due to metal whiskering.

CHAPTER 5
DESIGN DESCRIPTION

To design a 3D simulation in Unity, it is important to understand the creative potential of the software in attempting to implement design selections. The desired capabilities range from importing the CCA CAD layout from the user, interpreting variation from user inputs, and generating necessary data to analyze bridging probability and frequency. It is known that Microsoft Visual Studio allows one to modify game simulation interactions in Unity through the integration of C# coding language. This section provides details on how Unity will be modified both in its game features and using C# in Visual Studio to meet the selected design criteria and display valuable results to the user. Additionally, simulations in both Unity and MATLAB were produced to validate the application of the team's design approaches.

CCA Interpretation

In the simulation design, CCAs will be implemented into the Unity interface. As previously mentioned, the user must be able to import their design into the simulation to evaluate the user's CAD model in the simulation. Manual post processing from the user will be necessary to allow Unity to register its conductive components. The team will look to automate this process in the next stage of development, as this would make the overall user experience better. Therefore, the manual application of importing CCA CAD models for Unity interpretation is as follows.

File Processing

In inputting the user's CCA model, Unity requires the user to import a .obj file of their CAD model into the game window. For the present implementation of the team's design, the user will simply insert their desired files into the display editor. They will first upload the file to

the Assets folder. When they are ready for the file to be put into the simulation, they will drag and drop the file from the Assets folder into the simulation window. Figure 8 shows what this window looks like in the editor.

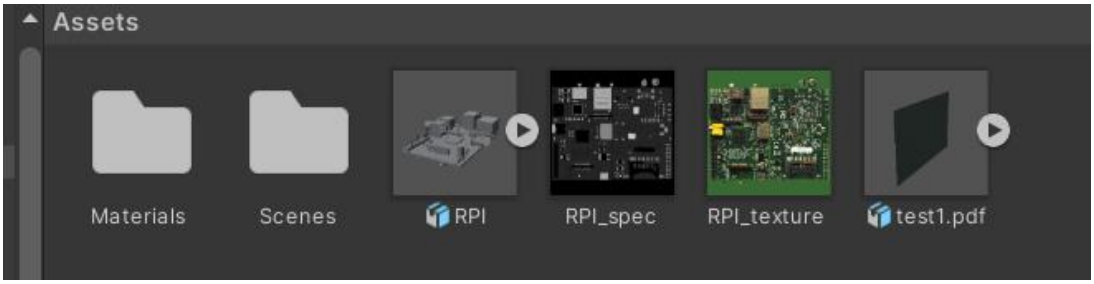


Figure 8: Asset Folder in Unity

When the user drags their file into the interface, Unity will automatically create individual GameObjects for necessary parts of the design and group the GameObjects associated with the imported file together in the Hierarchy of the game editor. These GameObjects are referred to as the "building blocks for scenes in Unity" [18]. These files will take the geometric properties of the CCA into consideration and represent them in the proper scale. Due to the fact that CCAs can contain very small components, the ability to accurately define each component's geometric properties is crucial in developing an accurate simulation.

In recognizing the geometry of the CAD model, the user must also identify the volumetric constraints that the whiskers will be bounded by. This is beneficial for cases when the circuit card is completely enclosed in its real-world application, or if it is attached along a surface—both of which will output different results for bridging probability.

Material Processing

Conductive components arranged on the CCA must be registered into Unity to identify potential locations for whisker bridging. In Unity, materials may be applied to selected GameObjects to assist in identifying these conductive points. While a tedious process, the

current approach is to allow the user to manually identify the conductive points on the CCA and establish them as Colliders, allowing Unity to register and perform an operation when contact is made. Automated processes in identifying these conductive points will be looked at as the team progresses to make this more seamless.

For the next stage of development, materials in Unity will be used as a visual identification for the user [19]. Ideally, the simulation will automatically register the materials of each component on the circuit card. As previously mentioned, the team will look to automate this process as development progresses. Materials in Unity have different properties that can be altered by the user. After selecting the desired material, it can be applied by dragging and dropping it onto the GameObject within the game window [20].

Test Models from Altium

To evaluate the feasibility of trying different circuit cards to the simulation in Unity and analyzing different probabilities of bridging, it is necessary to test multiple CCAs of varying designs. Some CAD models can be found online, one source the team has used is a Raspberry Pi CAD model that provides a complex arrangement of electrical components [21]. However, other models can be designed and saved through a program known as Altium.

Altium allows for “advanced product design, collaboration, and CCA design tools”. However, it has been described as “easy to use for new designers and students”, thus making it quick and easy to design and test different arrangements of components on a circuit card. This software has a limit to the size of the boards it can make. Specifically, the dimensions are 100 x 100 inches (2540 x 2540 mm) [22].

In designing a CCA, an electrical schematic of the circuit card must first be drawn in 2D. The designer can implement various elements, such as resistors and capacitors, and arrange them

to their liking while connecting them with wire. The schematic shown below in Figure 9 is used to power an LED, and the design itself was created copying an online tutorial circuit [23]. D1 represents an LED and P1 represents a board connector. Specifically PCB connectors work to transfer power from one PCB to another [24].

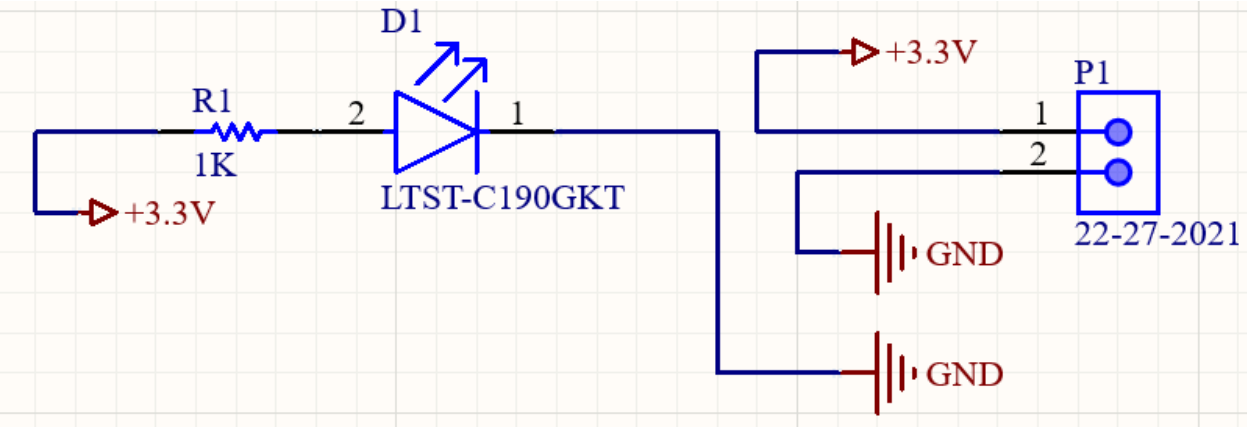


Figure 9: Altium 2D CCA Schematic

These files can then be transformed into a 3D model within the editor window [25]. Figure 10 reveals the same board generated into a 3D model.

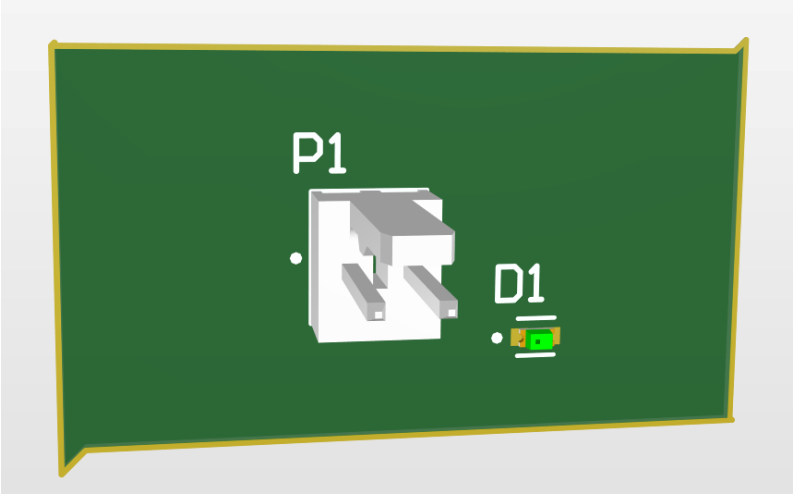


Figure 10: Altium 3D CCA Model

Altium allows these files to be saved as a .obj file, which can be properly translated into Unity [26]. Ideally the team will create a handful of these with various component arrangements to test in the 3D simulation.

Random Whisker Dimensional Distribution

Monte Carlo Simulation is known to find its strengths in estimating the probability of an outcome by generating a random set of samples based on input distributions. Due to limited knowledge of whisker growth, it is impossible to predict the exact dimensions of metal whiskers after they have detached. However, known measurements from previous incidents of whisker bridging on circuit cards provide a rough estimation of the lengths and diameters whiskers can attain. Through the application of normal and lognormal distributions of whisker dimensions, the probability of a specified number of whiskers bridging on a given CCA can be determined to best represent real-world expectations. Microsoft Visual Studio allows Unity to be modified in game simulation interactions and will therefore be used.

Normal distributions are a common approach to representing the expected outcome of a set of random variables. Samples are spread relatively evenly around their average (or mean), and result in a bell-shaped curve. With an input of the mean (μ), standard deviation (σ), and quantity to generate, a set of samples that fit the specified criteria can be formed [27]. Unity and MATLAB both contain built-in functions to create normal distributions based on these inputs. However, while normal distributions are a typical analysis for generating a set of detached whiskers, it is known that the dimensions of detached metal whiskers are better represented by lognormal distributions.

As seen in Figure 5, metal whiskers follow a lognormal distribution in both their length and thickness. Lognormal distributions relate to normal distributions in that if the logarithm of

each point in a dataset was taken, the data would fit a normal distribution [28]. Figure 11 reveals this relationship, where a lognormal distribution is represented by X, and after taking its natural log, a normal distribution, represented by Y, is returned [28].

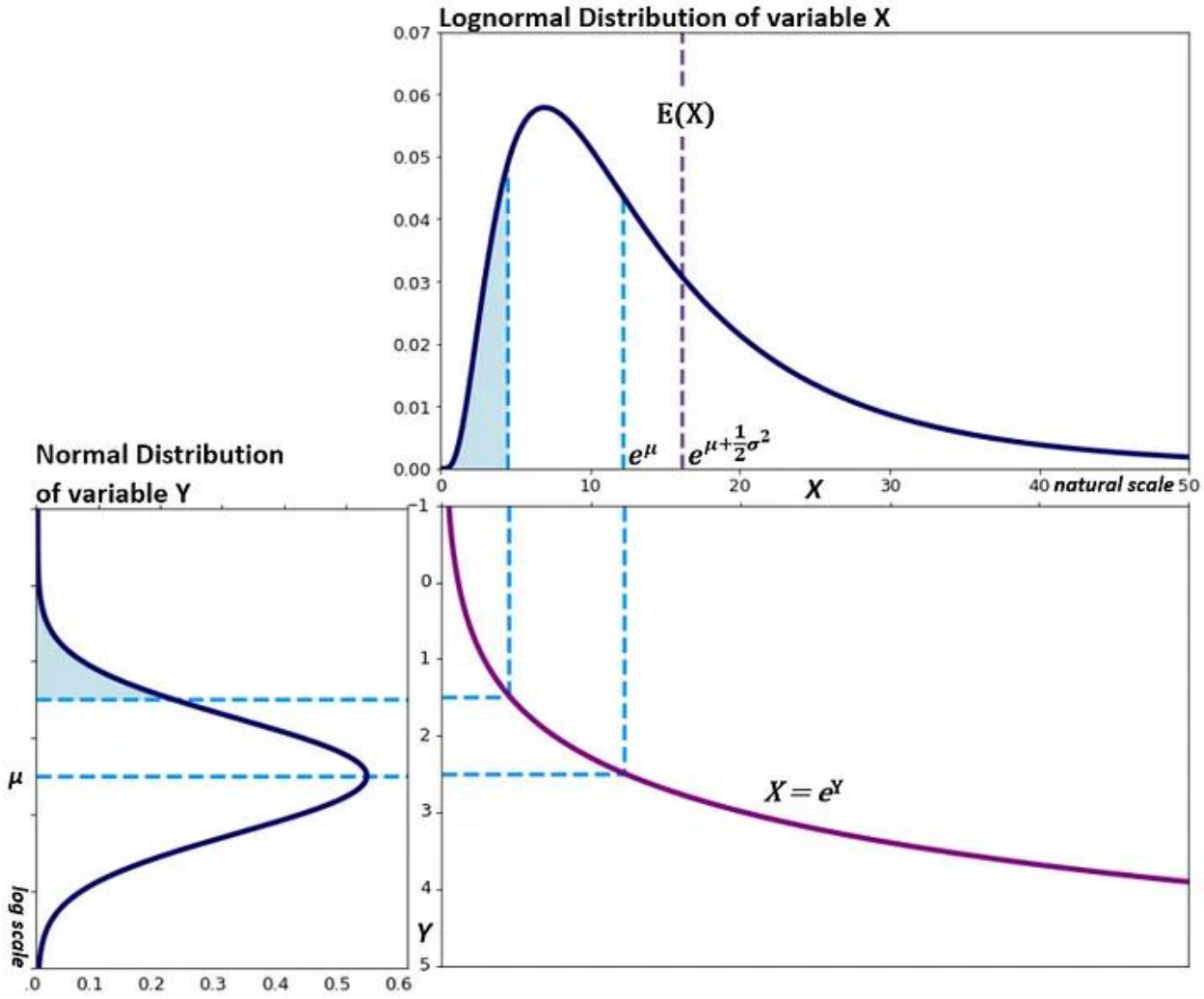


Figure 11: Normal and Lognormal Relationship [28]

Lognormal distributions differ from normal distributions in that samples are not evenly distributed about a mean value. Additionally, variables mu (μ) and sigma (σ) are respectively redefined as the location and scale parameter of the distribution, instead of the mean and standard deviation [28]. These input values are provided by the user to generate the distribution, as they are the result of calculations obtained by various real-world measurements of detached

whiskers. Therefore, it is assumed that the user has already calculated the location and scale parameters of a distribution they desire to simulate. To create this lognormal distribution, the user's location parameter (μ), scale parameter (σ), and quantity of samples to generate are the necessary inputs [28]. Visual Studio has built-in functions using C# to accept these input parameters, and from them, generate a normal distribution. Based on the relationship established in Figure 11, the exponential function of each sample that is generated in the normal distribution can then be taken to produce a lognormal distribution, using Equation (5):

$$X_{log} = \exp\left(X_{norm}\right) \tag{5}$$

where X_{log} represents the lognormal distribution of samples, and X_{norm} represents the normal distribution of samples. Upon creating the lognormal distribution, the location and scale parameters are used to compute the median, mean, mode, and variance [28]. These properties will be displayed to the user to reveal the success of the lognormal distribution they have created.

The median represents the central tendency of the generated lognormal distribution where samples cluster symmetrically and is beneficial when there are anomalies in the samples that skew the mean [29]. This can be seen in Figure 5, where the samples cluster in lower values, and begin to pull away at higher values. The mean represents the average value, while the mode represents the value with the highest occurrence in the distribution. Variance reveals the dispersion of data, with a higher variance indicating that the samples span further around the median, and lower variance indicating that the samples are closer to the median [29]. According to Pavlovic, the median, mean, mode, and variance can be calculated using Equations (6) through (9):

$$Median = \exp(\mu) \tag{6}$$

$$Mean = \exp(\mu + \frac{\sigma^2}{2}) \quad (7)$$

$$Mode = \exp(\mu - \sigma^2) \quad (8)$$

$$Variance = [\exp(\sigma^2) - 1] \cdot \exp(2\mu + \sigma^2) \quad (9)$$

These properties will be displayed to the user to reveal the success of the lognormal distribution they have created. Lastly, the individual probability that each iteration of generated whiskers formed a bridge on the CCA can be calculated as a percentage using the Equation (10):

$$Individual\ Probability = \frac{\#\ of\ bridged\ whiskers}{\#\ of\ whiskers\ generated} \cdot 100 \quad (10)$$

Whereas the overall probability of the simulation across all iterations can be calculated as a percentage using Equation (11):

$$Overall\ Probability = \frac{\#\ of\ bridged\ iterations}{\#\ of\ iterations\ ran} \cdot 100 \quad (11)$$

These probabilities give the user a risk assessment for the various scenarios or inputs they adjust in the simulation, giving them information on what design changes must be made to prevent whisker bridging and short circuits in their system. In addition to variations in whisker dimensional distribution, external forces on the circuit card can also affect bridging probability.

External Force Modeling

To make the simulation more accurately model how the metal whiskers will land on the board, thus giving a more accurate risk assessment, three external forces will be implemented into the program. These external forces are commonly found in many scenarios where a CCA is being used. The first external force that will be implemented is gravity. In real-world applications, circuit cards can operate in a variety of environments, such as Earth with standard gravity, outer space where no gravity is present, and other celestial bodies with varying levels of gravitational force, such as the Moon, or Mars. Figure 12 displays a graphical comparison of

gravitational accelerations between Earth, the Moon, and Mars at certain elevations, indicating that whiskers will behave differently depending on the selected gravity.

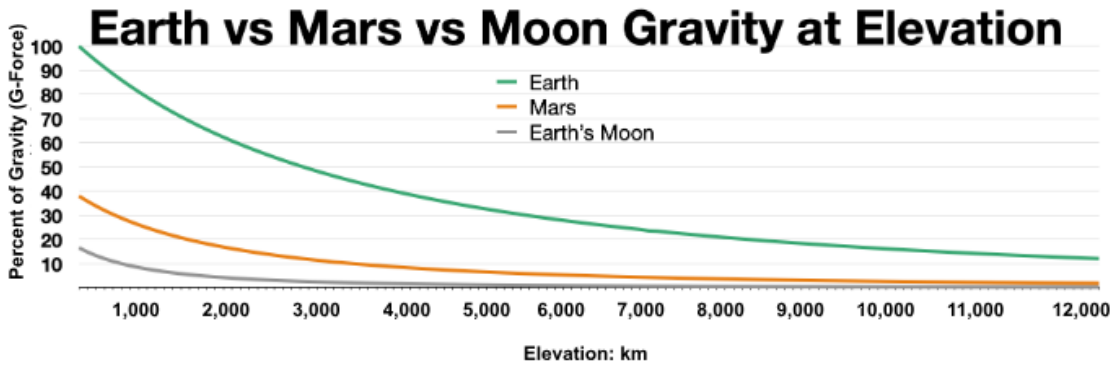


Figure 12: Comparison of Earth, Moon, and Mars G-Force at Varying Elevations [30]

The team will incorporate gravity into the code by applying a force to each simulated object. This applied force, recognized as the ConstantForce component in Unity, will model the force of gravity that corresponds to the possible operating environments. Using C# code within Visual Studio, different gravitational acceleration values will be stored, and can be applied to the simulation as needed. When using the simulation, the user will be able to select which environment the CCA will be operating in using a drop-down box found on the main screen. The gravitational acceleration that corresponds to the selected environment will then be applied to the falling whiskers accordingly.

The second external force that the team plans to implement into the code is a constant vibration applied to the circuit card. In many operating environments, a circuit card could experience a vibration with some amplitude and frequency, which could potentially affect how a falling metal whisker could land on the circuit card. In the designed 3D simulation, a vibration will be able to be applied in any user specified direction. The team will model vibration by applying a force to the circuit card and the generated conductors. The applied force will be

modeled as a sinusoidal force and will be applied using the AddForce Unity function [31]. The user will be able to input their desired amplitude and frequency, along with how long they want the vibration to be applied for. The function for the sinusoidal wave that the applied force will follow is represented by Equation (12):

$$y = A \cdot \sin(\omega \cdot t) \tag{12}$$

where y is the force applied to the circuit card (in Newtons), A is the amplitude of the applied force (in Newtons), ω is the frequency of the sinusoidal movement of the circuit card (in Hertz), and t is the total time that the vibration will be applied for (in seconds). Any vibration the user desires to apply to the circuit card in the simulation will follow the previously mentioned sinusoidal form and oscillate within the bounds of the amplitude for a given time duration and frequency.

The third external force that the team will implement is a mechanical shock, similarly applied to the circuit card. A piece of equipment with circuit cards could potentially undergo high impact actions or partake in maneuvers that would produce high amounts of G-force in a short amount of time. In the complete simulation, the user will be able to apply a shock to simulate the potential quick and harsh forces a circuit card might experience. When using the simulation, the user will be able to input the amount of force the shock will apply to the circuit card, along with the direction that the shock is applied in. As mentioned in applying vibration, the shock will be incorporated into the code using the AddForce Unity function. The team's technical advisor, Dr. George Flowers, recommended that shock should be modeled using a half-sine pulse applied for one hundredth of a second. This applied shock will take the same form from inputs mentioned in Equation (12). However, the way mechanical shock will be implemented into the code will be vastly different than the way vibration will be applied.

As previously mentioned, vibration will be modeled as a sine wave. Over the entire user specified time, the amplitude will be reached multiple times, thus meaning that the maximum specified force will be applied to the CCA in multiple instances over the entire time vibration is applied. However, mechanical shock will be modeled as a half sine wave since it is a single jolt to the system. When applied, the CCA will only experience the maximum amount of force once, rather than multiple times [33]. Figure 13 shows the difference between a half sine-pulse and a complete sinusoidal function.

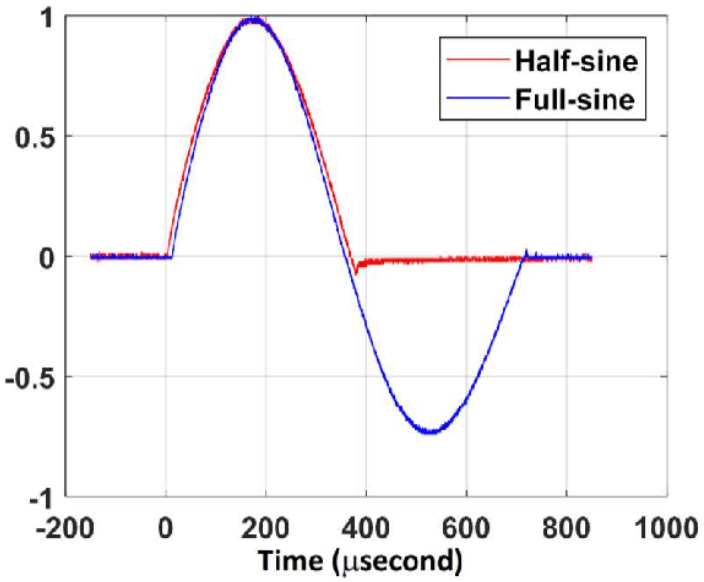


Figure 13: Half-Sine vs. Full-Sine Wave [32]

As seen in the figure, a mechanical shock would only be applied once, as the half sine pulse only reaches its amplitude a single time. In contrast, the full sine pulse reaches its amplitude in multiple instances, revealing oscillation. This means force would be applied to the CCA multiple times in alternating directions, resulting in vibration.

User Interface

In successfully allowing the user to add and adjust input parameters to define the simulation, text boxes must be available to the user that transmit values from user inputs into the

C# code used in Visual Studio. These input boxes will remain in the Unity game window where the imported CCA model will be located. This is for simplicity, so that the user does not have to edit the code or navigate Unity's editor to make changes. The following inputs detail how the user's values will reflect into the code to adjust simulation outcomes.

Lognormal and Normal Distribution Selection

As mentioned, metal whiskers are modeled by lognormal distributions as they are encountered in the real-world and are therefore a required distribution method for the simulation to employ. However, if time permits, the team's sponsor has asked for the ability to select between lognormal and normal distribution in the case that the user would like to analyze results from the latter. This would require the input selection to turn "on" or "off" a script in Unity containing the distribution parameters, which may not be directly feasible.

Mu and Sigma for Whisker Dimensions

Depending on the distribution selection, the meaning of mu and sigma for both the length and diameter of the whisker cylinders will differ. As previously mentioned, normal distributions require a mean (μ) and standard deviation (σ) to characterize a set of whiskers to generate. However, lognormal distributions require a location (μ) and scale (σ) parameter to characterize whiskers. This input will still ask the user for a mu and sigma (in mm), but depending on the distribution type selected, values will be used to calculate the selected distribution in its corresponding script.

Number of Whiskers

The input number of whiskers directly correlates with the distribution of whiskers as the final input necessary to generate the desired number of whiskers and their dimensional variation using the normal distribution function.

Gravity Selection

Gravitational acceleration influences the weight of the whiskers as they fall. Selections between Earth, the Moon, or Mars must be available to the user to vary results under different conditions.

Whisker Material Selection

The selection between tin, zinc, or cadmium whiskers gives the user the freedom to analyze the variance in probability and frequency of bridging on their imported CCA model. With material properties of coefficient of friction, density, resistivity, and the previous inputs of whisker volume and gravitational acceleration, Equations (1) through (4) can be applied to adjust whisker mechanics in the simulation as they fall and interact with other game objects.

Coordinate Drop Location

For the user to determine where their whiskers would like to drop from, X, Y, and Z coordinate inputs (in mm) will be made available. The user will be able to verify these inputs as a preview of the whiskers will be shown at the set location, allowing them to be adjusted until the user is ready to start the simulation.

External Force Selection

External forces for vibration and shock will be available to the user, requiring the corresponding inputs from Equation (12). These inputs will be accepted by Unity and visually display the impact of their application to the simulation as the generated whiskers fall and interact with other objects on the circuit card under these external force influences.

Figure 14 displays a concept on how the user interface will be integrated into the game window of Unity.

Figure 14: Unity UI Concept

C# Functions

Written code to control the program is necessary for the Unity simulation to function as desired. Through compatibility between Microsoft Visual Studio and Unity, C# can be applied in the Unity Engine to translate user inputs into adjustments to simulation conditions. While C# and the Unity Engine have some useful built-in functionalities, many of the functions must be hard coded to obtain the simulation objective. These functions include many of the algorithms and

methodologies that allow the user to have full control over the simulation. Inputs include controlling whisker dimensions, potential conductor placement, gravity and vibration control, and the Monte Carlo Simulation to assess the risk of bridging. The following functions all work simultaneously inside the engine to produce a functional 2D simulation that can assess many different environments and CCA layouts to give the user the best possible outlook when concerned with metallic whisker bridging. This 2D simulation is described later in the report and allowed the base design of the code to be tested and confirmed before being applied to a 3D simulation.

Make Conductors

This is one of the first functions that gets called by the program when running. Make Conductors is the main code that controls the generation of the conductors into the simulation. This code works by taking in the user input for the number of conductors needed, then utilizes the built-in Random Unity function to produce a randomized spawn location for every conductor and places them onto the board to simulate a 2D model of a CCA. These conductors act as the main points of interest and contact points when determining if a bridge has been made in the simulation.

Conductor Control

This function has the purpose of making sure that the conductors being spawned in by the user do not have any overlap. As previously mentioned, the conductors will be placed into the scene in randomized locations. A common occurrence was that some conductors would be spawned into close proximity or inside of other conductors, which would cause false connections

to be registered. To do this, components called colliders were attached to the base conductor. Colliders are the boundaries of an object that can be set and applied to any object required. This allows Unity to know if an object has made physical contact with any other part that is currently in the scene. Conductor Control checks the spawned in conductors, registers which ones have overlapped by utilizing the colliders attached to each one, then removes the overlapped objects from the scene. It then counts the number of conductors remaining and places more into the scene if needed. This process is repeated until the desired number of conductors are in the simulation with no overlapping objects.

Distribution Select

This function's purpose is to allow the user to select which type of distribution to apply to the simulation in terms of generating the whiskers. The user will be able to select from two main distributions: normal and lognormal. Having this will let the user have more control over the whisker generation, in turn providing a more realistic simulation. The dimensions of the whisker cylinders will be based off of those parameters within the selected distributions.

Length Distribution Generate

The purpose of this function is to generate randomly selected values from the selected distribution for the length of each whisker cylinder. To call this function into action, the user must first input the mu and sigma of the whiskers into an input field present in the simulation. The code will then take the input field and convert the values into integers, allowing for the numbers to be processed. If a normal distribution is selected, the user's values then flow into a built-in Unity function called RandomFromDistribution. This will then create a value from the

normal distribution based off of the inputs and assign it to a whisker. If a lognormal distribution is selected, and user's inputs will be created from the same normal distribution function, then applied to Equation (5) to generate the lognormal distribution.

Diameter Distribution Generate

Similarly to the previous function, the purpose of Diameter Distribution Generate is to create values from the selected distribution that will be applied to the diameter of the whiskers. The user inputs mu and sigma for the diameter, then follows the same process as the Length Distribution Generate, generating the values from the desired selection. The numbers can then be assigned to a whisker and further spawned into the simulation.

Material Selection

This code will be the final input to characterize the whiskers, which is their material. To achieve this, parameters such as whisker density, coefficient of friction, and resistivity will be integrated into the C# code. Similar to the gravity control script, the user will be prompted to choose the whisker material, enhancing the simulation's accuracy by accommodating different whisker types.

Make Whiskers

The purpose of Make Whiskers is to generate or spawn in all of the desired whiskers at the appropriate lengths and diameters. To do this, another user input is required so the user can tell the program how many whiskers they want to have simulated. This function then calls the previously listed Length Distribution and Diameter Distribution to produce values for each

whisker that is desired. It will then call the function listed below, Coordinate Location, to determine where the whiskers will spawn.

Coordinate Location

This function's purpose is to be an option for users to control where the whiskers will spawn into the simulation. The user will be prompted to enter a pair of coordinates, which will then tell the program the area where to spawn in the whiskers in. This function will be more vital moving into the 3D simulation as having the third axis will require more precise spawning locations that the user can control for each run. The users will import their CCA, where an origin will be set at the corner edge of the board. The user then enters in appropriate coordinates based on the size of their circuit cards. By enabling users to set the spawning location for each simulation, it allows the user to fine-tune the parameters and the simulation to their specific needs and scenarios. Finally, the code will spawn the whiskers with properties derived from previous computations at the desired locations, enabling the simulation to continue.

Gravity Control

This function is one of the most important, as it controls the force of gravity for the whiskers across the simulation. This works by using the Unity component: ConstantForce. This allows for any object with this component to have a constant force applied that can be set or manipulated into any magnitude or direction needed. To effectively model gravity in the simulation, all whiskers will utilize this component to be applied in the negative y-direction, and the code in Gravity Control would then change the value of the constant force depending on what

the user desired. Users can choose from Earth, Moon, or Mars gravity to apply to the simulation at any time.

External Force Generation

This function serves as the control for the shock and vibration forces being applied to the whiskers. It will be attached to two buttons present in the simulation that the user can press to apply either a brief shock or impact to the CCA, or a vibration force over a set period of time. The code will also take in an input for the user to set the amount of time the vibration will occur. To apply the vibration force, the Unity function ConstantForce will be used. The code will oscillate the magnitude of the force quickly, effectively modeling the vibration of an object. Another Unity function titled AddForce will be used to generate the shock or impact force. This adds one set force to an object for a brief moment, allowing for the user to create an impact to their CCA as many times as needed. This is to better simulate more aggressive environments that CCAs might be in like rocket or missile launches, allowing for more customization of the simulation for each user. The code for this function has been developed in a small-scale application, but not yet implemented into the 2D simulation mentioned towards the end of this report due to time constraints. In appropriately applying Equation (12) for both shock and vibration, this code will be further evolved in the upcoming 3D simulation.

Run Simulation

Run Simulation is the function that starts the entire process and triggers the other previously listed functions in the proper order. This function also acts as the Monte Carlo Simulation method for determining the odds of bridging. It will take in a user input for how

many iterations are wanted and run the simulation as many times as requested by the user. It will then generate the whiskers, allow them to fall, count the connections made, and then clear them to start again. This will produce a percentage of bridging that is printed to the user, which is the total risk of bridging across the whole CCA. The whiskers are allowed to fall and settle over the course of 20 seconds, and the time scale for the simulation is turned to 50 times the normal speed to simulate all of the iterations without making users wait for the entire period of time.

Whisker Control

This part of the code allows the whiskers to behave as needed, while also properly counting the total number of connections being made throughout the simulation. This is done once again by utilizing a collider. Whisker Control takes the colliders applied to each whisker, and states that if the whisker remains in contact with two or more conductor colliders, it will register that whisker as bridged and increase the total connections made counter by one. The function will also change the color of the connected whisker from the standard orange color to a bright white color to highlight to the user which whiskers made a connection, providing a more obvious distinction between bridged and non-bridged whiskers. In addition, to make the simulation more visually appealing, the whiskers objects will be visually larger than the user's inputs. The colliders of the whiskers will be the appropriate sizes based off the distribution values, but the whiskers in the simulation will appear larger, allowing users to see the whiskers and their connections more clearly when running through the simulation.

Dimension Storage

The primary function of this code is to save the dimensions of whiskers that form a bridge and store them into a list, enabling exportation to other platforms like Excel or MATLAB for more in-depth analysis. It achieves this by collecting each dimension as a bridge is formed and storing it within a Unity game object list. This list can then be easily converted into float or decimal values and further transferred to an external program. This approach is necessary as Unity lacks the capability for advanced numerical analysis or the generation of informative graphs or charts. By exporting the dimensions to a separate program, the user can generate useful visualizations, such as plots depicting whisker length versus the number of connections.

Total Restart

This function's main purpose is to allow the user to restart their simulation with new whisker dimensions and conductor layouts while also clearing the data collected from the previous run. It accomplishes this task by searching through the scene for the whisker and conductor objects, and then deleting them from the scene. It then sets the values for the total connections made back to zero. This allows users to rapidly run through multiple different scenarios and environments that could be more geared towards their own desires and needs. All these functions culminate into the Unity simulation, allowing it to work properly and efficiently.

These functions were applied to a 2D simulation to test the design and outcome of simulation results based on user input. This allowed the team to analyze their operation on a smaller scale to determine which core mechanics will be further developed and translated into a 3D simulation.

Simulation Results

Upon completing the Unity simulation based on all user defined inputs, the user will be able to access the probability of bridging encountered during each iteration, and the probability of bridging across all iterations of the simulation. This will be accomplished using Equations (10) and (11) respectively and will provide the user with an estimation of the best- and worst-case scenarios that are present on their selected circuit card CAD model. Along with this, for lognormal distributions only, the resulting median, mean, mode, and variance of whisker dimensions that were generated will be displayed in the game window, and respectfully calculated using Equations (6) through (9).

Additionally, the Dimensional Storage function will store the dimensions of distributed whiskers in Excel, tracking the number of times a whisker's approximate length and diameter formed a bridge. This information will be stored in a specifically named file with a designated location. To reveal to the user the frequency that whiskers of specific dimensions will bridge their CCA, MATLAB will be used to access the data within the Excel file. In calling to the specific rows and columns in the Excel file using MATLAB, the MATLAB script will read these values and plot a total of five histogram figures:

- Frequency (y-axis) of each conductor pair (x-axis) that is bridged.
- Frequency (y-axis) of lengths of whiskers (x-axis) that create a bridge.
- Frequency (y-axis) of diameter of whiskers (x-axis) that create a bridge.
- Frequency (y-axis) of length:diameter ratio of whiskers (x-axis) that create a bridge.
- Frequency (y-axis) of electrical resistance of whiskers (x-axis) that create a bridge.

To obtain these results, the only responsibility of the user is to open the MATLAB .m file and make a numerical change to the Excel file they are calling to in the script. This is necessary

because each completed simulation in Unity will need to create a unique Excel file with a consistent name but iterate the number as new simulations are run. Running the MATLAB script will finalize the desired whisker simulation outputs. While this approach requires three separate applications to complete, the team will investigate other methodologies to accomplish this same solution and streamline the entire process.

Test Simulation Designs

While the ideas constructed in previous sections detail the team's methodology, it is beneficial to prove that these designs are feasible under potential restrictions to the Unity program. In testing the previous design implementations to achieve the desired outcome of this project, two individual simulations were developed: a 2D simulation in Unity relating user inputs to physical outputs, and a 1D simulation in MATLAB analyzing bridging frequency. These allowed the team to directly understand the successes of the conceptualized design and mitigate obstacles that would have been encountered later in the design process of the 3D simulation.

Unity

To confirm that the previously listed C# functions would work properly according to user input, a 2D simulation for detached metal whiskers was developed to test the code and ensure that design parameters would be viable to implement. One notable omission from the 2D simulation that was discussed in the C# functions was the incorporation of the ability to select log-normal distribution. This choice was aimed to simplify the whisker dimension generation process by avoiding potential coding complexities that could compromise the program. Instead, a normal distribution was utilized. However, the 3D simulation will feature the inclusion of the

log-normal distribution, along with the External Force Generation and the Material Selection functions.

Additionally, a logistical constraint with Unity prevented a previous project constraint from being met. The program was not able to be contained inside a single script due to how Unity handles written code. For Unity to work efficiently, the program needs to run the entirety of its code from game objects. A script can be written to control or manipulate a certain aspect, then attached to an object which allows the script to run. In terms of this simulation, four scripts were written that contain the previously mentioned functions for the program to run as desired. All scripts written in C# for the simulation.

In beginning the 2D Unity simulation, first the users will open Unity and select the program. This will open the scene and load in all functions and scripts needed to run the simulation. The users will then see the main screen, showing the user interfaces and CCA set up. Figure 15 below shows the initial screen users see as they load up the scene.

Figure 15: Unity 2D Initial UI

The next step to run the simulation will be to enter in values for each input field listed. That is, the mu and sigma values for whisker length and width, the number of conductors needed, the number of whiskers to be dropped, and finally the number of iterations desired for the simulation. For the dimensional inputs, the users must take into consideration the scale of the simulation: 10 units is equal to 1 mm. This is noted in the top left of the interface. To make the simulation as realistic as possible, the CCA was set to be 1000 units long, meaning that it is around 3.93 inches or 100 mm long. The conductors were set to be 2.5 mm wide.

Once the users have entered in their desired inputs, they can now press the 'Set Conductors' button. Once pressed, this will call the Make Conductors and Conductor Control functions, which set the conductors in place for the simulation. Figure 16 below shows the user interface after entering in the inputs needed and spawning in the conductors, colored in red.

Figure 16: Unity 2D Spawned Conductors

Next, the user will be able to select the gravity levels for the environment, utilizing the 'Select Gravity' dropdown box. Selecting this will call the script 'Gravity Control' and set the gravity force value to either the Earth, Moon, or Mars levels depending on the selection of the

user. This dropdown, shown in Figure 17, can be changed throughout the simulation to the user's desire.

Figure 17: Unity 2D Gravity Selection

Finally, the user can enter in the number of iterations they want the simulation to run through. A higher number of iterations would develop the most accurate results for determining the odds of bridging, but any number of iterations can be chosen. Once the user has the simulation set up in the desired manner, the simulation can then begin. Clicking the 'Start' button will call the function Run Simulation. This generates the requested number of whiskers by utilizing the Make Whiskers function, and places them into the scene above the CCA. Then, the simulation will continue as the whiskers get pulled down by the gravity force and potentially encounter the conductors. Figure 18 demonstrates the whiskers (in orange) falling onto the CCA after the user has begun the simulation.

Figure 18: Unity 2D Simulation of Whiskers Dropping

As the whiskers are falling, the Whisker Control script will start checking for contact between the whiskers and the conductors. If a connection is registered, the total connections counter displayed to the user will increase. Figure 19 below shows that the whisker has made a connection and become highlighted in white for visual distinction.

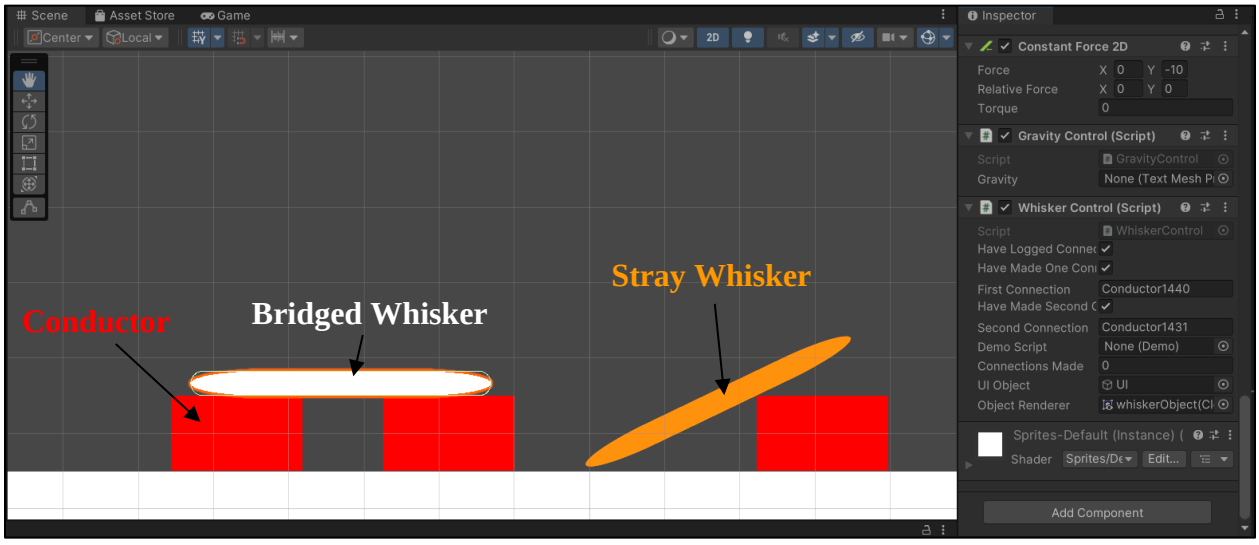


Figure 19: Unity 2D Bridged Whisker

The simulation will then continue moving through each simulation desired by the user and documenting the number of total connections made, and storing the whisker dimensions of each whisker that created a connection by calling the Dimension Storage function. Once the process is complete, the odds of bridging probability will be calculated using Equations (10) and (11) and printed to the user. Figure 20 below shows the simulation after the final run, using exaggerated dimensions of 20 mm lengths and 3 mm widths for clarity.

Figure 20: Unity 2D Post Simulation UI

After the simulation is completed, the user can then press the 'Restart' button if desired, which invokes the Total Restart function. The user can then run through different layouts and whisker dimensions for varied results. Table 3 shows the simulation modeling a 2D CCA with 15 conductors placed into the scene with various values for the user inputs as the user attempted to find the probability of bridging across multiple environments.

Table 3: Unity 2D Simulation Results

Simulation Number	Length [mm]		Width [µm]		Whiskers	Iterations	Envir-onment	Probability [%]	
	Mean	Std. Dev.	Mean	Std. Dev.				Min.	Max.
1	10	2	5	2	50	10	Earth	4	24
2	10	2	5	2	200	10	Earth	4.5	26
3	20	5	10	5	200	20	Earth	6.5	38
4	20	5	10	5	200	20	Moon	7	37
5	20	5	10	5	200	20	Mars	6	58

As seen in the table, the whisker bridging probability was effectively modeled. As expected, the bridging probabilities were at the lowest when the total number of whiskers and the dimensions of the whiskers were at a minimum. Increasing these values raised the probability of bridging. However, these values are not directly controllable in a real-world setting due to how whiskers form; instead, they are integrated into the simulation to more accurately replicate the real-life scenarios encountered by the circuit cards. These probabilities indicate that the modeled CCA would exhibit volatility, making it an unsustainable layout for a potential circuit card. The user could then restart the simulation to generate a new layout featuring increased spacing between conductors or fewer conductors, potentially reducing the risk of bridging.

MATLAB

While the integration of storing bridged whiskers from the Unity simulation into Excel for MATLAB to access was not yet implemented into code due to time constraints, a similar simulation was executed in MATLAB to validate the ability to display graphical results in bridging frequency. Two simulations were created: a single simulation of normal distribution of whisker dimensions, and an iterated simulation performing the same concept but as many times as the user desires.

The purpose of the single simulation was to test the ability to create whiskers and conductors in vector notation and analyze bridging results by graphical representation. Four "conductors" were created as vectors with a length of 20 integers, with each value incrementing by one. Similarly, five "gaps" were created as vectors with the same length as the conductors to separate them by. For example, conductor one was represented by a vector spanning numbers 20 through 39 along the x-axis, and conductor 2 spanning numbers 58 through 77 as a gap separates both conductors. This was applied to the remaining conductors and gaps to create a one-dimensional circuit card. Figure 21 displays the resulting circuit card, where the length of the gaps are denoted by a black line, and the length of the conductors are denoted by black squares.

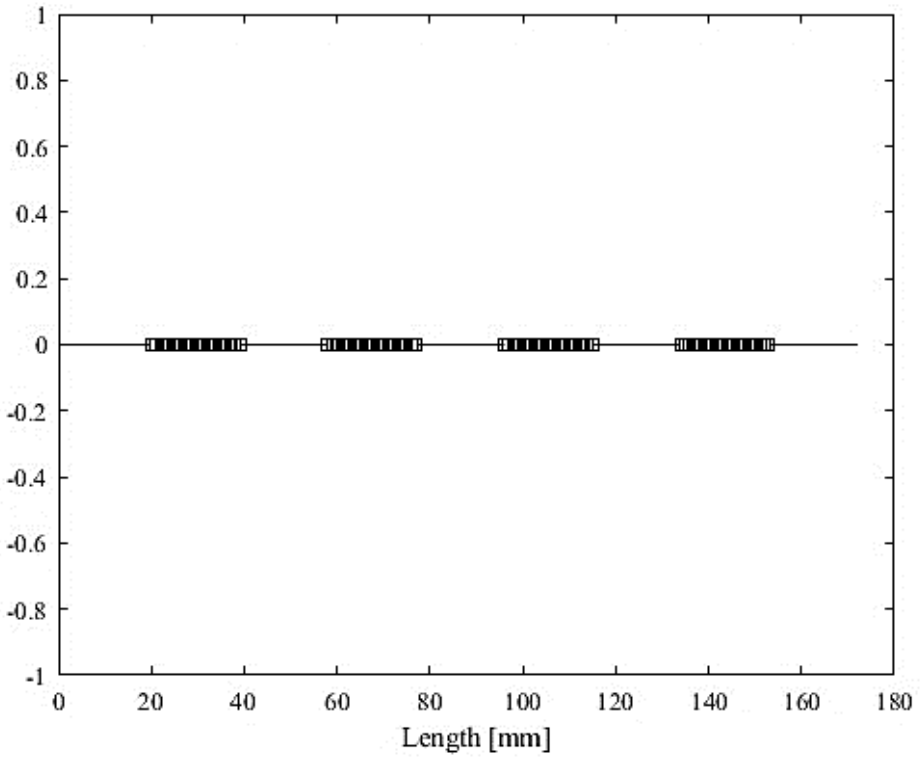


Figure 21: MATLAB 1D Circuit Card

Five whiskers were then generated using the normal distribution function which accepted inputs for mu, sigma, and number of whiskers to generate. These whiskers, similar to the conductors and gaps, were also vectors of integers incremented by one, and were randomly

generated based on the normal distribution inputs. Boundaries were also set using the full-scale length that the series of conductors and gaps covered along the x-axis, ensuring that the randomly generated whiskers could not have integers extending outside of the circuit card's bounds. Figure 22 reveals the random distribution of the five whiskers, denoted by stars, along the length of the circuit card. Each whisker generated has its own color for a clear distinction from the others and for visual understanding of its length.

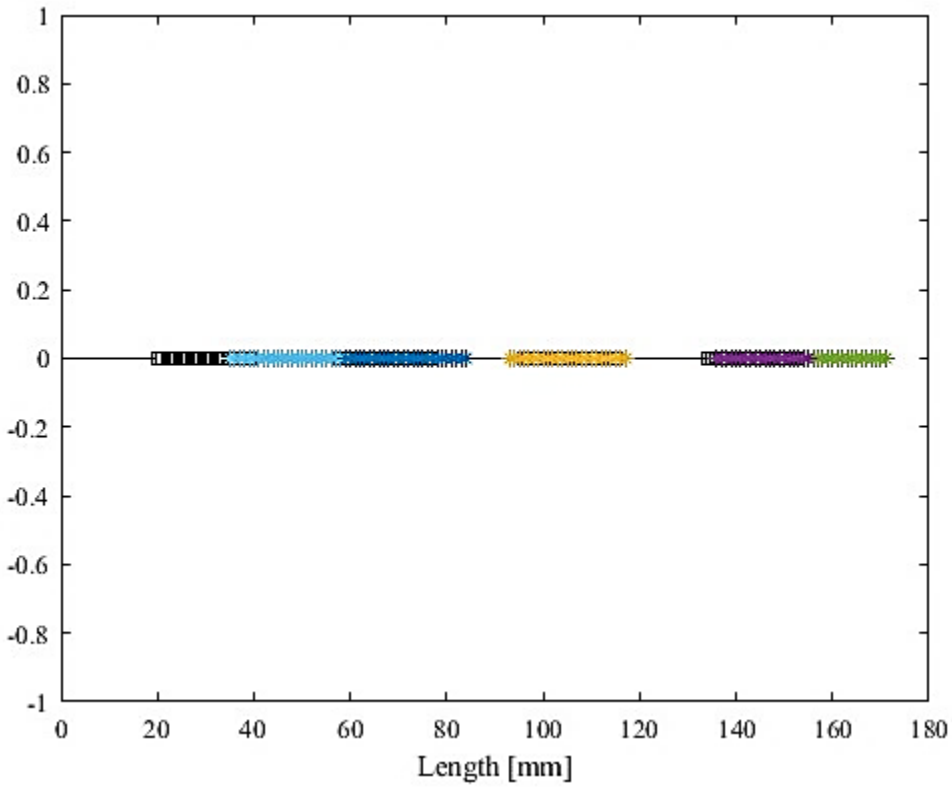


Figure 22: MATLAB 1D Whiskered Circuit Card

As seen in the figure, whisker one (light blue) has formed a bridge between conductors one and two. To register this contact, the code checks each whisker's vector to see if any of its values are shared by two conductors that are next to each other. If this is returned as true, then the code returns to the user that the whisker has formed a bridge between the specified

conductors. If false, the code tells the user that the whisker did not form a bridge. These results are displayed to the user in the command window for verification.

The iterated simulation applied the same principles as the singular simulation, but embedded within a loop that ran from one, up to the number of iterations the user desired. Additionally, this simulation tracked the length of whiskers that created a bridge in each iteration, tabulated them, and displayed the frequency of all generated whisker lengths that managed to bridge any set of conductors. Figure 23 displays the resulting whisker length bridging frequency of a simulation ran for 1000 iterations, using the same initial conditions as described in the single simulation.

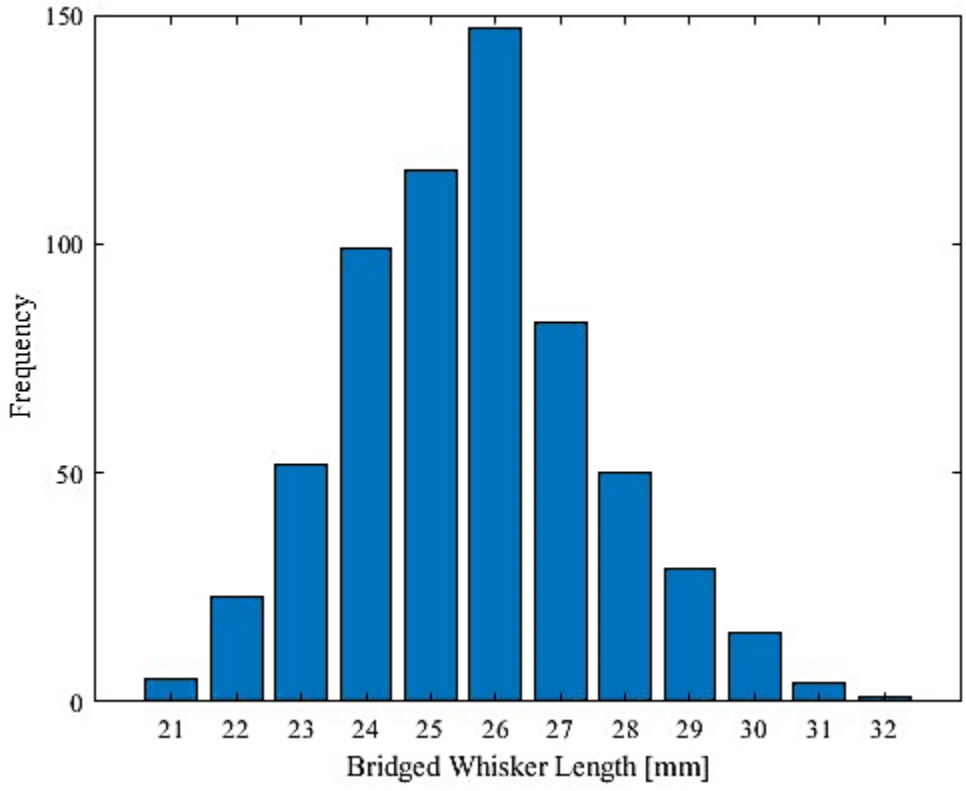


Figure 23: MATLAB 1D Iterated Simulation

A normal distribution can be seen, where the mean length to cause bridging are whisker vectors with 26 integers.

CHAPTER 6
SIMULATION CONSTRUCTION

Upon integrating the designs described in the previous chapter, a customizable 3D simulation in Unity was successfully developed. This simulation modeled the behavior of detached metal whiskers generated using statistical distribution, encountering various circuit board designs and external forces, for the user to assess bridging probability under potential real-world scenarios. This section details the implementation of the previous detailed designs, step-by-step instructions on simulation use, testing performed for authentication, sources of error, and proposals for further development. Required programmatic questions addressing the comprehensive challenges and considerations of the project can be found in Appendix A.

Design Implementation

When implementing the previously described designs, the team experienced successful implementations of many of the designs based off their initial description. However, the team also dealt with other functions of the project that required adjustments to best fit the team's design goals. This section provides detail on how the design descriptions outlined in the previous chapter were used in developing the 3D simulation, and any key modifications made to accommodate design implementations.

CCA Processing

To design and import various CCA's into the simulation, the Altium software was used. The overall implementation method remained as the team anticipated, with the user taking an .obj and .mtl file created from Altium and then inserting the files into Unity. To identify the various conductive materials on a given CCA, the team incorporated an automated feature instead of a manual selection. This meant that the user would no longer have to select each

conductive surface on the circuit board, and instead input a material name into a textbox (e.g. Copper). A script then compares this user input to the various materials present on the uploaded CCA. If a match is found between the input material name and material name within the board's object hierarchy, a conductive tag is applied to all components on the board with the same name. This was accomplished in part using the list functionality in C#, which allows elements to be added one at a time without knowing the total number of elements, similar to a dynamic array [34]. For the simulation, a script copies each component on the board as a GameObject and stores them within a list. Depending on the name of that GameObject, a particular tag is applied to each surface on the board for collision identification. A delegate and listener are used when taking the inputs from the conductive material text box. They work to run a function that applies the necessary conductive tag to the corresponding GameObject, thus registering bridging [35]. The function is set up to run when the input is not empty, by modifying code found online [36]. The team used the help of NASA engineer Timothy Mondy to assist in learning Altium, as well as Auburn graduate student Jake Botello for help with Unity.

The user also needs to understand the overall layout of the imported CCA to best understand which parts are conductive. In the case of boards designed in Altium, with the exception of copper, material names appear as "mat_19", "mat_20", etc. on the board in Unity. This means that the user must look ahead of time to note the material names of the possible conductive points and double check other components to make sure all instances are correct. To assist with this, an external software known as Blender can be used to convert the file to a .fbx file type. Then, either a separate project in Unity would need to be created to view the board, or the .fbx file needs to be hidden during the simulation. While the team solely used .obj and .mtl files for testing, the use of .fbx files may assist in a different problem that occurs in the

simulation. This area overall will be explained in the Sources of Error and Further Development section within this chapter.

Additionally, if the traces on the physical CCA are covered, they will need to be removed from the CAD file prior to importing the board in the simulation. Otherwise, Unity will treat the traces as an exposed conductive surface, which will reduce accuracy in the results. Shown below is Figure 24, which shows an example of a CCA created in Altium. For this figure, the previously mentioned and copied tutorial circuit was modified to work with in development.

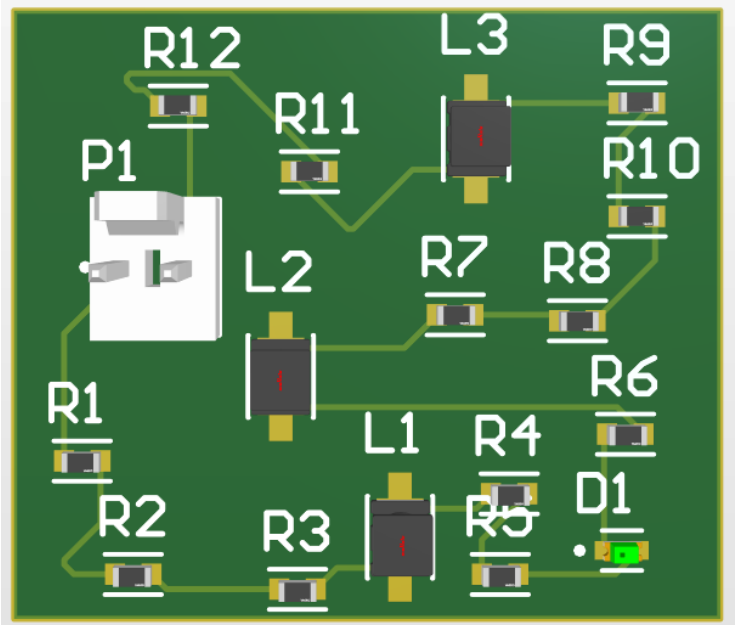


Figure 24: Example Altium CCA

Distribution Selection

The process of implementing the normal and lognormal distributions into the simulation followed the same concepts outlined in the previous chapter. In the simulation, the user can select normal or lognormal distributions for their whiskers from a drop-down menu. An internal script recognizes which selection the user has made, references the corresponding function, and plugs in the respective mu and sigma inputs to calculate whisker dimensions. Figure 25 reveals

the hard-coded toggle between normal and lognormal distributions, which functions on a switch statement to change between the two distribution cases.

```
1 reference
void DropdownValueChanged(TMP_Dropdown change)
{
    switch (change.value)
    {
        case 0:
            uiScript.distributionType = DistributionType.Lognormal;
            break;
        case 1:
            uiScript.distributionType = DistributionType.Normal;
            break;
    }
}
```

Figure 25: Distribution Selection Script

The normal distribution uses a built-in function available to C# called RandomFromDistribution, which requires the mean (mu) and standard deviation (sigma) based on the user's input for length and diameter. For a lognormal distribution, a function within the script applied Equation (5), using inputs for location (mu) and scale (sigma) parameters to generate a normal distribution, and taking the exponent of the results to obtain a lognormal distribution. Figure 26 reveals the relative functions in the script used to create normal and lognormal distributions, which is set to return the normal or lognormal value for the corresponding whisker dimension.

```
private float GenerateLogNormalValue(float mu_log, float sigma_log)
{
    float normalVal = RandomFromDistribution.RandomNormalDistribution(mu_log, sigma_log);
    float logNormalVal = Mathf.Exp(normalVal);
    return logNormalVal;
}

private float GenerateNormalValue(float mu_norm, float sigma_norm)
{
    return RandomFromDistribution.RandomNormalDistribution(mu_norm, sigma_norm);
}
```

Figure 26: Distribution Functions

When incorporating the two distributions into the code, the team opted to not display the mean, median, mode and variance values to the user, to declutter the UI and remove unnecessary information. These parameters were referenced in the previous chapter and could be calculated using Equations (6) through (9). However, their purpose is to give the user an understanding of the shape and traits of a lognormal distribution, and do not affect how they are generated. Additionally, the parameters were not required by the team's sponsor.

Whisker Scaling

Due to the incredibly small physical nature of metal whiskers, the team decided to apply a scale of 10 to all boards imported by the user. By increasing the board's size, the user can easily see the generated whiskers, which are typically measured in microns.

Another important issue the team had to consider was how to ensure the whiskers generated were at the appropriate scale for the imported circuit board. Unity is not intended to be used as CAD software and does not have measuring tools to validate the size of game objects. Upon various tests, the team concluded that Unity text boxes accept numerical inputs measured in centimeters. This was done by importing a 3-inch x 3-inch board from Altium and creating a single whisker to match its length and width. To create this board, particular parameters were set in Altium, and then the board was cut to size [37]. The dimension tool in Altium was also used with these boards, and the units had to be properly flipped [38,39]. The whisker was adjusted until its length and width were set at 7.62 in its scale. This indicated that inputs were in centimeters, as 3 inches is 7.62 centimeters. Figure 27 below shows the test board (left) and whisker (right) from a top-down perspective, revealing that their lengths and widths match under centimeter input.

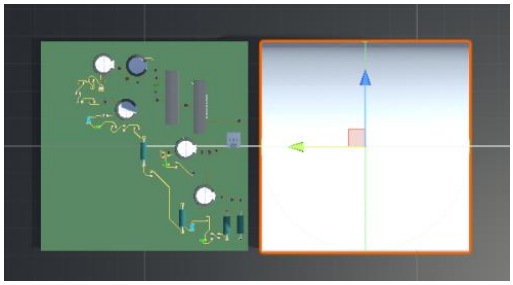


Figure 27: Whisker Scaling Validation

However, whiskers are measured in microns, therefore requiring the text boxes to accept values in this dimension. Unity does not have a feature to change the input from centimeters to micrometers. Therefore, to create whiskers under micrometer inputs, internal code divides the input value by 1000. Although the conversion between centimeters and micrometers is a factor of 10000, the scale of the simulation has been increased by a factor of 10, requiring a decrease in the conversion factor under this larger scale.

Material Selection

In the simulation, the team successfully implemented the ability for the user to change whisker material types. When running the simulation, the user chooses their desired material in a dropdown in the simulation menu, with the options being tin, zinc or cadmium. When a material is chosen and the simulation is run, a script automatically applies the material properties of density and resistivity to the simulation. This allows for certain values like resistance, measured in ohms, volume, measured in cubic micrometers, and mass, measured in kg, of the generated whiskers to be calculated. The equations used by the team in a script for the calculations was Equation (1), which calculates the mass of the whisker, and Equation (4), which calculates the resistance of the whisker. Equations (2) and (3) were not used, due to Unity having a built-in feature that applies a set coefficient of friction to any game object and handles the frictional

forces encountered. Figure 28 displays the dictionary within a script that is referenced based on the whisker material selected, and the material property needed, in order to be integrated into the whisker's physics.

```
3 references
public Dictionary<MaterialType, MaterialProperties> materialProperties = new Dictionary<MaterialType, MaterialProperties>()
{ //density (kg/um^3), resistivity (ohm*um), coefficient of friction (unitless)
    { MaterialType.Tin, new MaterialProperties(7.3e-15f, 1.09e-1f, 0.32f) },
    { MaterialType.Zinc, new MaterialProperties(7.14e-15f, 5.9e-2f, 0.6f) },
    { MaterialType.Cadmium, new MaterialProperties(8.65e-15f, 7.0e-2f, 0.5f) }
};
```

Figure 28: Whisker Material Dictionary

Drop Location

To allow the user to define the drop location of their whiskers, a coordinate input was implemented. As described in the previous chapter, the user defines the volumetric bounds that their whiskers are to be randomly placed within. A central reference point, or origin, of this system is positioned at coordinates (0,0,0), and users can input specific values that the system utilizes to establish an invisible rectangular prism with which whiskers can spawn within. When a user inputs a value for each axis, the coordinate system calculates a spawning range extending from both negative and positive directions of that value from the origin. For example, if the user entered a value of 3 cm in the x-direction, 2 cm in the y-direction, and 3 cm in the z-direction, the generated whiskers would be set to spawn within a 6 cm length, 6 cm width, and 4 cm height range above the circuit board. In this way, users can flexibly adjust their coordinate parameters and modify the spawning boundaries based on their circuit board's geometry. Located below is Figure 29, which accepts the coordinate values from the menu on the user interface and creates the volumetric constraint of the whiskers spawn location where they are dropped from.

```
// Generate dimensions and spawn position
float diameter = WidthDistributionGenerate() / 1000;
float length = LengthDistributionGenerate() / 1000;
float spawnPointX = UnityEngine.Random.Range(-float.Parse(xCoord.text)*10, float.Parse(xCoord.text)*10);
float spawnPointY = UnityEngine.Random.Range(1, Convert.ToInt32(yCoord.text)*10);
float spawnPointZ = UnityEngine.Random.Range(-float.Parse(zCoord.text)*10, float.Parse(zCoord.text)*10);
Vector3 spawnPos = new Vector3(spawnPointX, spawnPointY, spawnPointZ);
```

Figure 29: Drop Location Calculation

External Force Application

For the three external forces intended to be applied in the simulation, a few design changes were necessary to ensure their function. The first external force, gravity, was modeled in the 3D simulation in the same way it was implemented in the 2D simulation, by changing the gravitational constant within the simulation's physics settings. Figure 30 displays the gravitational drop-down selection in the user interface.

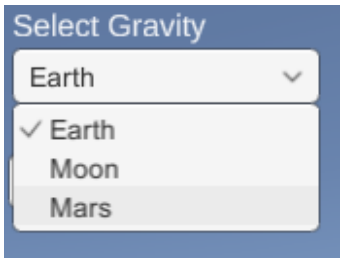


Figure 30: Gravitational Acceleration User Interface

The second external force, an applied vibration, required modification to represent a true sine wave in the simulation. The vibration was still modeled using a sinusoidal function; however, the board's movement was determined based on a position calculated by a script using Equation (12), rather than by applying a force directly to the board to change its position. Beyond this, the vibration was implemented as expected, with the user being able to pick an X, Y or Z direction, and adjust inputs of amplitude, frequency and vibration. Additionally, a toggle is available to the user to allow them to apply vibration in the simulation and be used in each iteration. Figure 31 displays the vibration inputs available in the user interface.

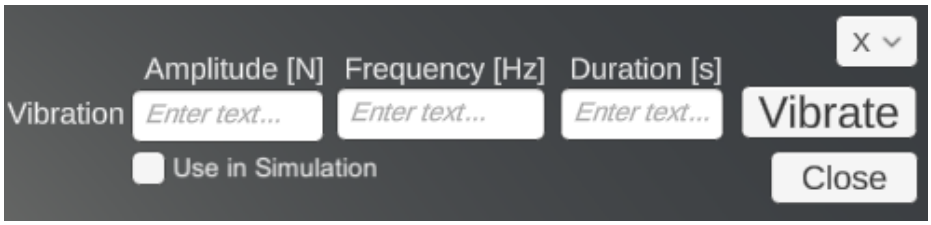


Figure 31: Vibration User Interface

The third external force, shock, also faced changes with how it was incorporated into the 3D simulation. While a half-sine pulse caused a jolt in the board's position, the whiskers were not affected whatsoever and showed no signs of movement. The team's technical advisor, Dr. George Flowers, suggested that the team model the applied shock using a modified version of the vibration script, with constant values set for duration and frequency. The duration and frequency values were tuned until constants of 0.1 seconds and 5 hertz respectively met the desired results of an applied shock. Similarly, shock could be applied in the X, Y, or Z direction, with an adjustable input for its amplitude, and a toggle to apply this force in the simulation to be used per iteration. Figure 32 displays the shock inputs available in the user interface.

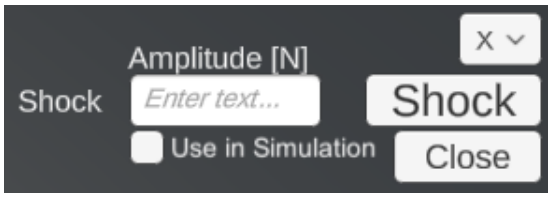


Figure 32: Shock User Interface

Results and Accessibility

Upon interpreting results after the simulation has completed, users can process a few results in the physical simulation by analyzing the circuit board. Any whisker that has caused a bridge will have its color changed to red, in contrast to the initial white color of the generated whiskers. This simple change allows the user to quickly see the bridged whiskers on the CCA

under the last iteration ran. Figure 33 shows a whisker that has bridged and turned red, while several others have not bridged and remain white.

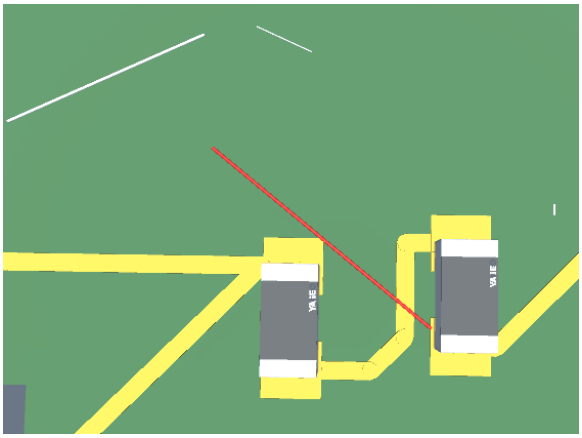


Figure 33: Example Bridged Whisker

Another way the team allows the user to visually analyze the data is by toggling a heatmap feature. If chosen to be used in the simulation, the heatmap allows the user to see which areas of the board are more likely to cause a bridge to occur. As whiskers are dropped over a series of iterations, all conductive surfaces realized by the simulation will change color along a gradient depending on how many bridges have occurred. This gives the user a quick and easy way to understand which sections of the board are more prone to bridging. Once a bridge is detected, all conductive surfaces on the CCA that contributed to the bridge change color and increase in its gradient as it encounters additional bridging incidents. Figure 34 reveals the heat map feature applied to a circuit board that has encountered significant bridging, with a gradient revealing the frequency of bridging for each conductive surface. This board was not designed by the team. It was downloaded from a site that Jay Brusse mentioned [40].

Figure 34: Heat Map Feature

Furthermore, statistical results of the simulation can be generated to reveal bridging probabilities and bridged whisker frequencies. These results allow the user to see how adjusting certain aspects, such as whisker geometry, quantity, or applying various external forces, can affect the board under bridging instances. For the user to generate these results, a directory path and file name are required to capture the raw Unity data into a .csv file for Excel processing. This design has deviated from the original approach of incorporating MATLAB to generate results. While MATLAB extensions are available to Virtual Studio Code, the C# code used to handle the physical interactions within the simulation cannot communicate with MATLAB code in transferring bridging results. Both languages can be used in the simulation, but the team found no way of communicating between the two because they do not share the same approach in data handling.

The data captured in the saved Excel .csv file includes the length, width, resistance, and iteration number belonging to all whiskers generated, as well as all whiskers that created bridging. Figure 35 displays these values from a portion of data captured by a simulation ran for five iterations.

Copy and paste new Unity data below.							
2344.882	1.848655	95.22385	1	273.2997	8.589521	0.514089	1
320.8697	4.340045	2.36416	1	186.033	2.423198	4.396923	1
1085.211	3.487889	12.38015	1	193.3856	3.619274	2.048886	1
102.6456	16.63976	0.05145	1	55.25309	6.138229	0.20352	1
236.6911	19.22683	0.088859	1	281.829	1.452754	18.53269	1
67.14388	13.55089	0.050747	1	389.0316	2.011397	13.34522	1
40.35767	1.670242	2.007723	1	98.05896	1.303999	8.003304	1
88.04597	1.221839	8.184995	1	957.6295	1.841145	39.20649	1
36.73108	8.801983	0.065797	1	142.2036	14.10397	0.099212	1
13.54542	3.152975	0.189099	1	261.551	1.880877	10.26059	1
186.033	2.423198	4.396923	1	397.5871	6.018121	1.523516	1
254.749	2.291744	6.731577	1	227.8579	2.132149	6.956101	1
584.2444	3.833191	5.518363	1	631.0462	3.964317	5.572641	2
273.2997	8.589521	0.514089	1	38.13288	0.666463	11.91472	2
169.9955	2.478171	3.841596	1	631.0462	3.964317	5.572641	2
143.7395	3.896694	1.313772	1	631.0462	3.964317	5.572641	2
115.6845	3.273762	1.49802	1	90.54319	1.356419	6.829748	2
227.8579	2.132149	6.956101	1	108.0807	1.073343	13.0199	2
25.59343	3.146816	0.358692	1	93.94727	1.936148	3.478114	2
48.14874	1.954614	1.74904	1	61.61744	8.4147	0.120771	2
389.0316	2.011397	13.34522	1	57.63481	6.382478	0.196355	2
140.7167	3.316131	1.775902	1	344.6671	1.575585	19.26871	2
57.80341	6.901667	0.168416	1	44.83537	3.357326	0.55204	2
299.5237	2.412891	7.139904	1	82.0699	3.743707	0.812675	2
191.2258	5.989491	0.739781	1	68.56454	2.533356	1.482667	3

Figure 35: Example Saved Unity Data

Starting from the left, the first four columns represent the length, diameter, potential resistance, and iteration number belonging to all generated whiskers, and the following four columns represent the same parameters but for all bridged whiskers. These values are then pasted into a macro-enabled excel file, titled “WhiskerResults.xlsm”, which calculates the individual and overall probabilities using Equations (10) and (11) respectively. This is accomplished by using the iteration number assigned to each whisker to determine the number of whiskers that bridge per iteration and the number of iterations that encounter bridging. Figure 36 shows the results of applying a built-in macro using the two probability equations for the data shown in Figure 35.

ALT + F8 -> GenerateProb	
Overall Probability	
100.00%	
Individual Probability	
Iter #	Probability
1	24.00%
2	24.00%
3	32.00%
4	22.00%
5	28.00%

Figure 36: Example Calculated Probabilities

These probabilities allow the user to assess the likelihood that bridging will occur under the input parameters and make adjustments as necessary.

Lastly, the data can then be interpreted using histograms to understand bridged whisker frequency. Plots for the length, diameter, length to diameter ratio, and resistance can be created to help anticipate expected whisker properties for those that may bridge on the user's circuit board. Figure 37 displays the set of histograms generated from the data shown in Figure 35.

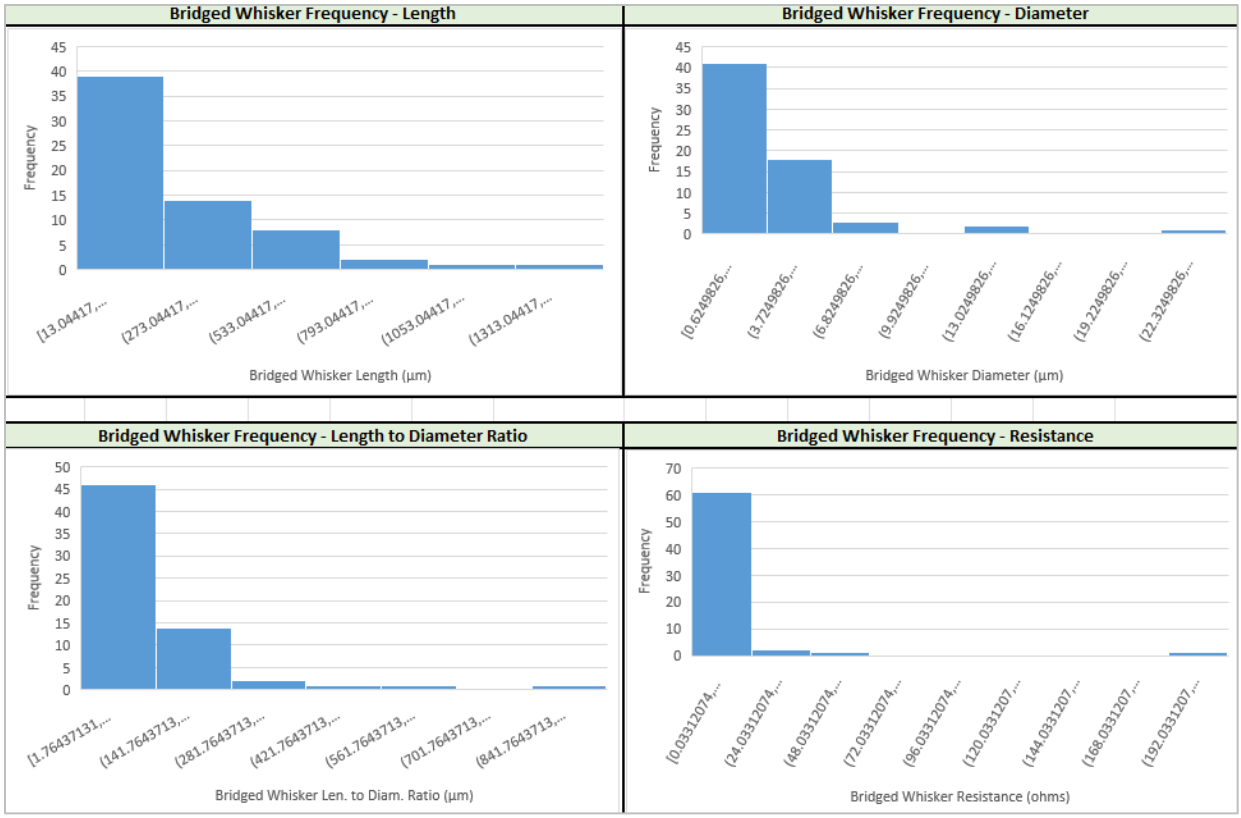


Figure 37: Example Frequency Histogram Results

Method of Operation

The following section provides a brief overview of the simulation developed for modeling detached metal whiskers. This simulation is designed to accommodate a wide array of scenarios, offering extensive and customizable inputs to ensure the highest accuracy in results. Users can tailor various parameters to reflect real-world conditions closely and analyze various circuit boards easily. To achieve the best results, users are encouraged to familiarize themselves with the simulation's interface and features, as detailed in this overview. For enhanced guidance, which includes step-by-step instructions and advanced usage tips, refer to Appendix B, Simulation User Manual. This manual contains in-depth information and detailed instructions on utilizing the program to its fullest potential.

Accessing the Simulation

This simulation was stored using GitHub. GitHub is a powerful version control and software management tool that enables developers to collaborate efficiently. This also allows ease of access to the simulation by the users, as the repository the program is stored in can be cloned to the user's personal device at any time. This allows any updates being pushed out to be obtained quickly and efficiently. The following sections explain briefly how to gain access to this program through GitHub.

Initially, users should create a GitHub account if one is not already created. This will allow access to repositories and what the tool has to offer. Next, users should look to download a program called GitHub Desktop. This app is a desktop version of GitHub that allows easier and more straight forward access to repositories along with any updates that could come out later in the program's timeline.

Once GitHub Desktop is downloaded, open it and log in. From there, look to the top left of the app for the File tab. Click this, and find the option: Clone Repository. Figure 38 demonstrates this.

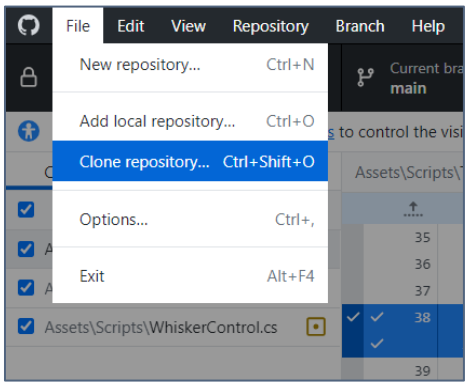


Figure 38: Cloning Repository

This will open a new screen. Clicking on the tab titled URL inside the screen and entering the following URL to the input field will tell the app which file to access. Defining a directory for the folder to save into and clicking Clone will copy the program files to the user's device.

https://github.com/connorm0088/3DStuff.git

Appendix B, Simulation User Manual, contains more details about cloning from GitHub along with other methods to complete the task.

Unity Setup

The Unity Editor and Hub should be downloaded to use the simulation files cloned in the previous section. Both programs can be obtained through the official Unity website. It is recommended that version 2022.3 of the Unity Editor be downloaded, as this was the version used when the program was being developed.

Once downloaded, opening the Unity Hub app will present the main interface. Clicking the 'Add' button will open up the file explorer on the user's device. Figure 39 shows what this should look like for a new user. The file explorer should be used to find the recently cloned repository, and selecting the file will add the program to the user's Editor, allowing access to the simulation.

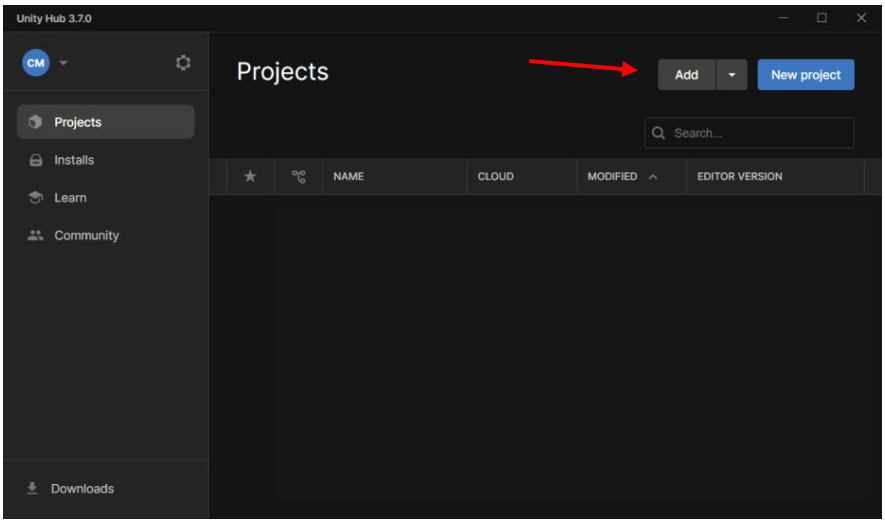


Figure 39: Unity Hub Interface

Opening the project will present users with an empty Editor. It is not necessary for new users to be familiar with the interface layout that is provided by Unity. For more in-depth instructions and descriptions of each aspect of the Editor, refer to Appendix B for the official User Manual. For the simulation to be fully imported into the project, users must navigate to the Project tab which is located on the bottom bar of the Editor. On the left side of the tab, there is a folder icon labeled Assets. Clicking the dropdown next to this folder shows a new file labeled Scenes. Inside there will be an item titled SampleScenes. Double clicking this will fully import the simulation into the Editor. Figure 40 shows how to access this folder and the item inside.

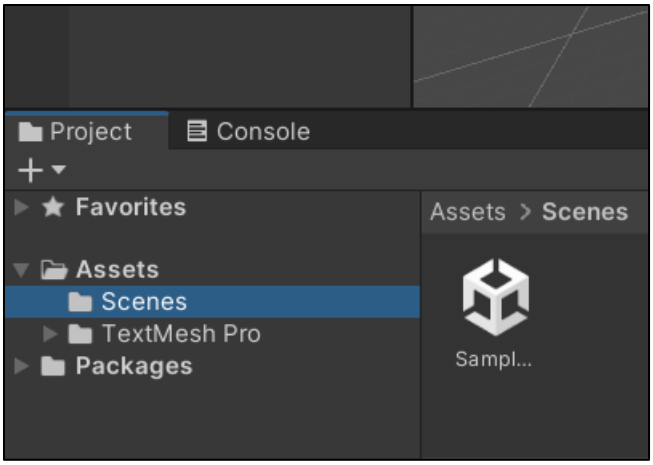


Figure 40: Sample Scene

Importing Circuit Boards

Next, users will need to import the CCA planned to be used in the simulation. Boards are to be imported as a .obj or .fbx file. Many circuit card modeling programs offer this capability. Altium was used to generate most boards tested while the program was being developed. To import the board into the simulation, users must open the folder containing the 3D version of the circuit board of choice. This can be done through the Unity Editor. The Project tab will have a new folder labeled Boards, given the simulation had been setup correctly. Dragging and dropping the desired board into the Boards folder will import the object into the simulation. Figures 41 and 42 demonstrate this process. When first bringing in the .obj and .mtl files into the simulation, it is important to drag both files together to ensure proper display [41].

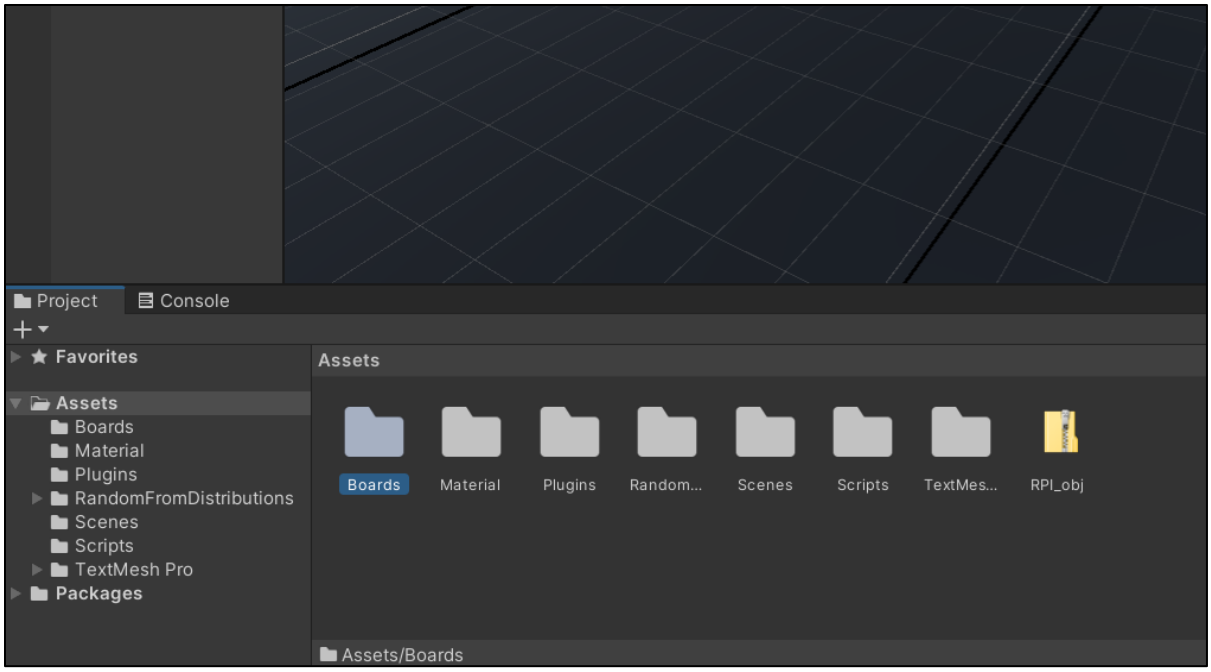


Figure 41: Boards Folder

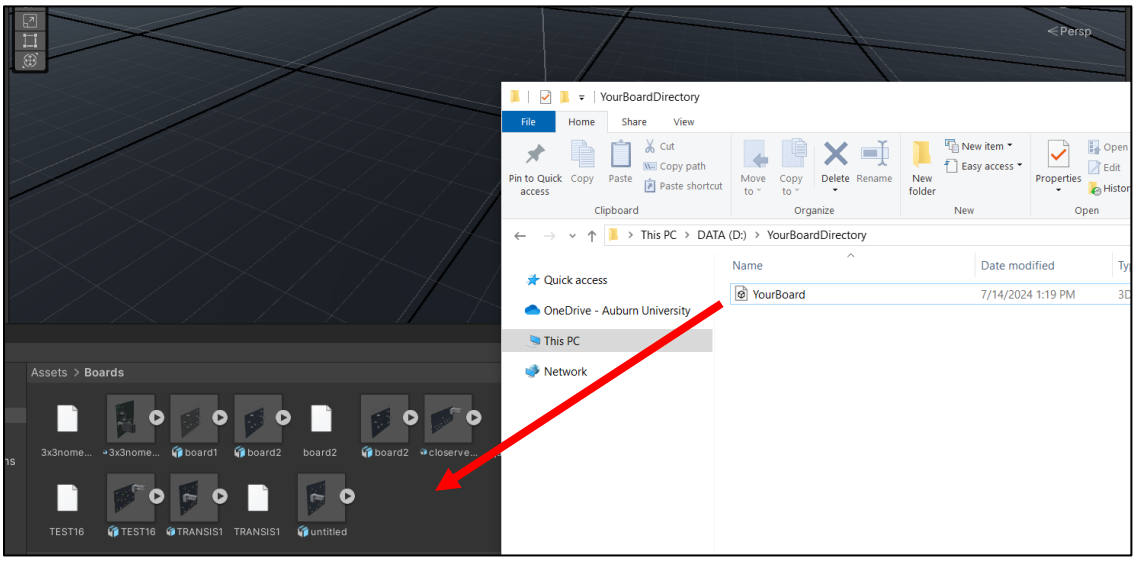


Figure 42: Importing CCA

Preparing Imports

For the user's CCAs to be properly analyzed in the simulation, some minor edits have to be made in order for the program to appropriately register the board. To do this, users will need to select their board by clicking on it and then looking into the Inspector window that shows up on the right side of the screen. There, an Add Component button will be seen. Clicking this button will allow different Unity components to be added to the board by bringing up a search bar where the names of certain objects can be typed into. In the case of the simulation, a specific script will need to be applied: Trigger Control. This script contains most of the code that scans objects for materials and allows different components to be set as conductive. Figure 43 demonstrates this step.

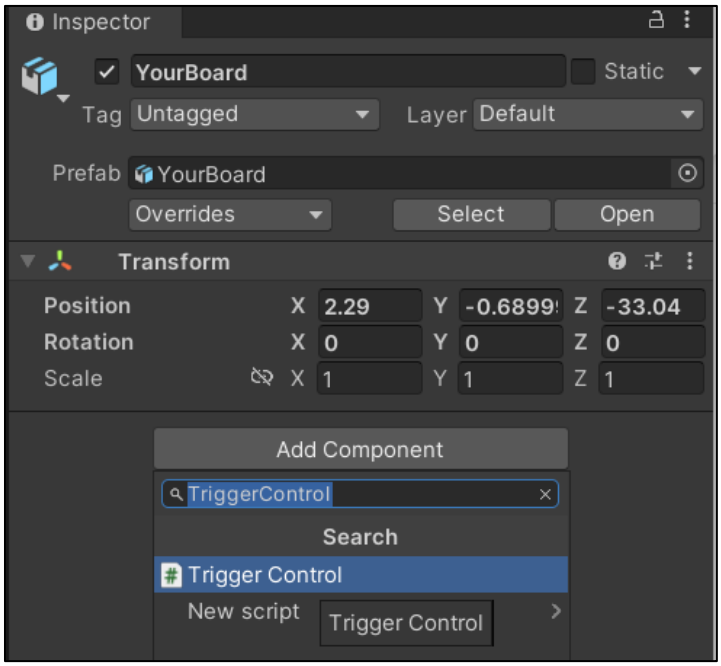


Figure 43: Adding Components

Once this script is added, a new field under the title 'Trigger Control' will appear. Users should then find the input fields labeled 'Material Input' and Trigger Material'. Next to these fields will be a small circle icon which when clicked will bring up a similar search bar as when

clicking Add Component. Search for the names of both respective fields and click them to fill in the input fields. Figure 44 shows a clear depiction of this step.

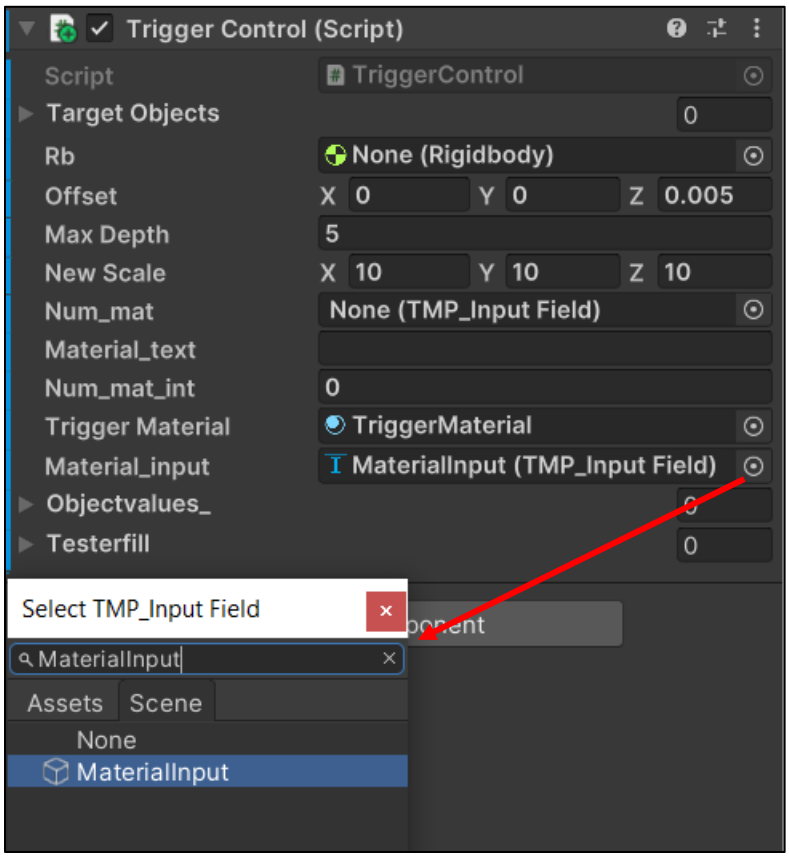


Figure 44: Preparing the Board

The board that is desired to be analyzed should now be set to run for the simulation. The previous steps will not have to be repeated for this board but will be for newly imported ones.

Running the Simulation

Once the boards are prepared for the simulation, users will look to the top of the Editor, and click the play button to begin. Once loaded in, holding down the left mouse button allows the W, A, S, D, Q, and E keys to be used to fly around the area. W, A, S, and D move the camera forward, left, backwards, and right, while the Q and E keys move the camera down or up. Preset

camera positions can be selected using the buttons in the user interface, seen as Top Down, Standard, and Side view. Figure 45 provides a clear look at the official user interface.

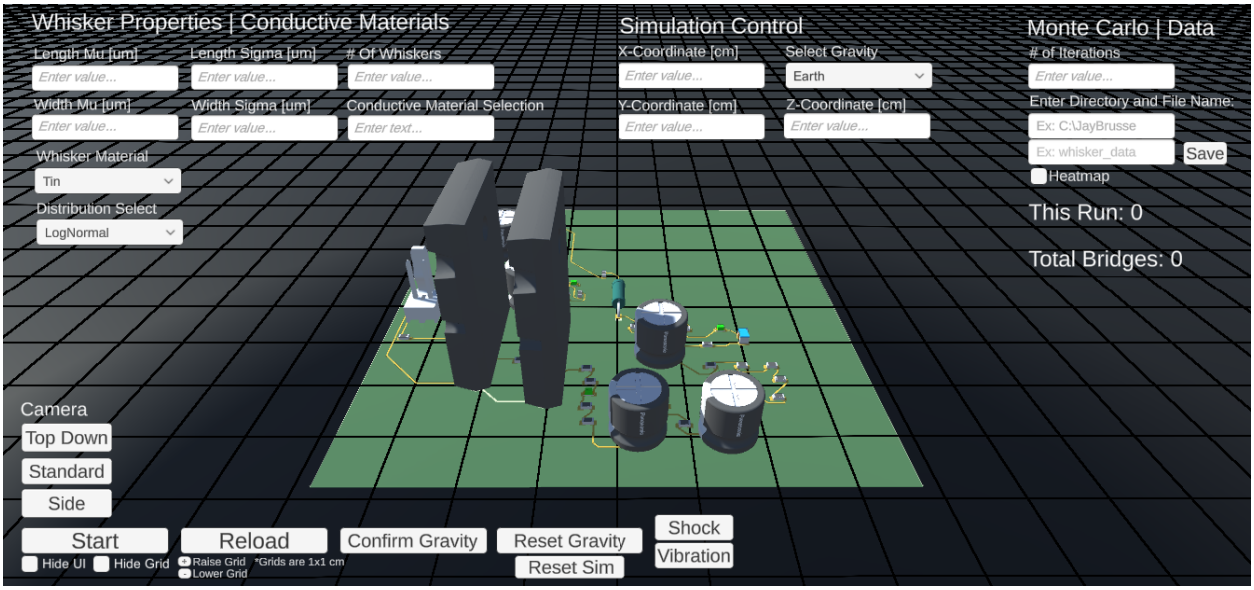


Figure 45: User Interface

To begin running the simulation, the inputs seen in Figure 43 must be filled in, then the Start button can be pressed. For an extensive and detailed explanation on each input and how to use them, look to Appendix B, User Manual. Once the Start button has been pressed, the whiskers will be generated and floating over the board. Users can then change any specific parameters and press Reload to see where the whiskers are spawning and the dimensions to make sure they are properly covering the board along with being correctly sized. Once the simulation is ready to run, the Confirm Gravity button can be pressed, and the simulation will generate whiskers across the user defined number of iterations which will then drop down with the selected gravitational acceleration. All whisker dimensions will be stored into a csv file in the directory given by the user along with the file name. Any whisker that bridges will be highlighted red in the scene and will also be tracked into the same file for further analysis. Refer to Figure 46 for a depiction of this feature.

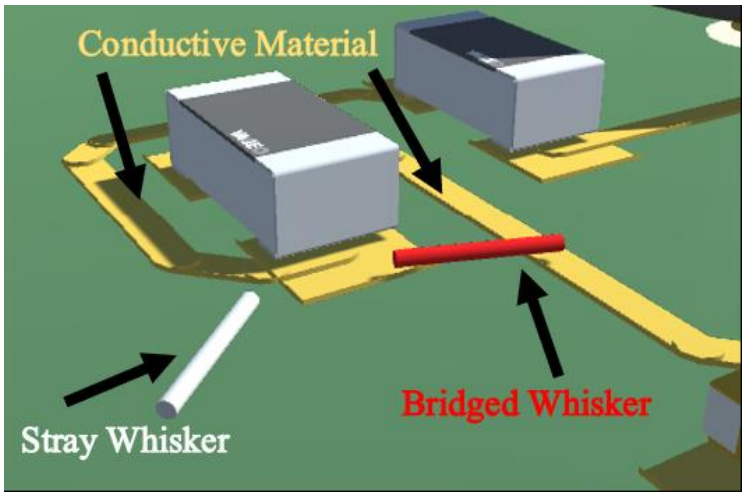


Figure 46: Bridged vs. Stray Whisker

External forces, such as mechanical shock or vibration, can be applied by pressing the "Shock" or "Vibration" button on the interface. This action opens a smaller section with customizable inputs. For vibration, adjustments can be made to amplitude, frequency, and duration. For shock, the amplitude/intensity is adjustable. Each force features a dropdown menu to select the direction for the forces to be applied in. Additionally, a toggle switch within each force's interface allows these forces to be applied continuously across multiple simulation iterations. Figure 47 shows the vibration interface with some sample inputs already typed in:

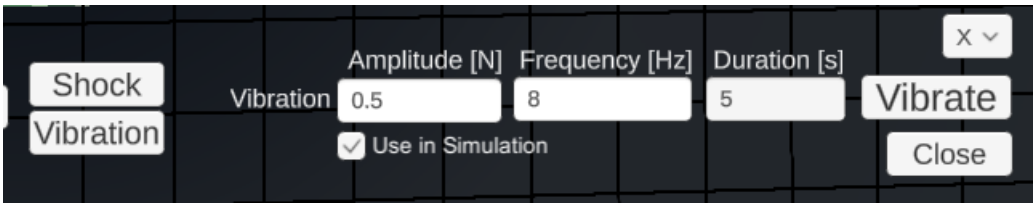


Figure 47: Sample Vibration Input

Additionally, a heatmap can be seen by pressing the Heatmap Toggle button near the top right of the interface. This changes the color of all of the conductive components on the board to a gradient ranging from green to a dark red depending on how many times the conductor

contributed to forming a bridge. The heatmap seen in Figure 48 provides a visual that shows the highest risk areas where the most concentrations of whisker bridges occurred.

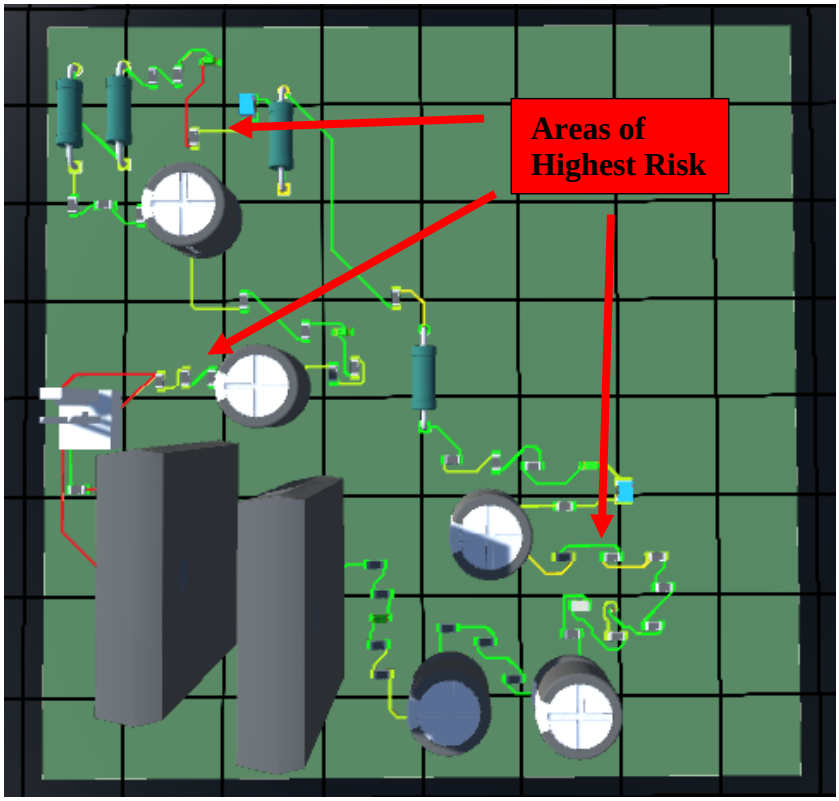


Figure 48: Sample Heatmap

Data Processing

Once the simulation is completed, the csv file created can be viewed and further analyzed. Navigating to where the file was saved and opening it will show data in the format seen in Figure 49. It can be seen that the dimensions of the whiskers along with the resistance and the iteration they belong to are listed, with the same parameters capture for whiskers that formed a bridge.

All Whiskers				Bridged Whiskers			
Length (um)	Width (um)	Resistance (ohm)	Iteration	Length (um)	Width (um)	Resistance (ohm)	Iteration
5500.664	151.3164	0.03334108	1	5499.993	148.587	0.03457302	1
5500.777	150.6189	0.03365131	1	5500.922	150.2488	0.03381817	1
5499.958	149.7035	0.03405903	1	5501.795	150.1787	0.03385513	1
5500.992	149.6109	0.03410761	1	5498.949	150.8678	0.03352919	1
5499.831	150.6828	0.03361695	1	5499.732	149.6539	0.0340802	1
5500.442	148.7595	0.03449571	1	5502.174	149.8021	0.03402789	1
5499.725	149.9621	0.03394021	1	5499.625	149.8744	0.03397935	1
5500.098	149.9509	0.03394758	1	5499.877	151.0531	0.03345262	1
5500.473	149.5945	0.03411185	1	5499.977	149.3731	0.03420996	1
5497.817	150.4017	0.03373037	1	5499.055	150.1084	0.03386997	1
5501.496	150.4387	0.03373636	1	5500.166	149.5694	0.03412142	1
5500.687	149.5997	0.03411079	1	5499.003	150.3931	0.03374154	2
5497.783	149.8971	0.03395764	1	5498.267	150.1608	0.03384146	2
5499.492	150.496	0.0336984	1	5501.301	150.5085	0.0337039	2
5500.704	150.5976	0.03366037	1	5501.881	150.9467	0.033512	2
5499.759	150.7462	0.03358824	1	5498.669	150.372	0.03374894	2
5501.197	150.593	0.03366545	1	5500.64	149.2015	0.03429285	2
5500.395	149.9039	0.03397069	1	5502.168	150.5374	0.03369628	2
5502.018	149.9339	0.03396711	1	5500.912	149.6023	0.034111	2
5500.601	149.7112	0.0340595	1	5500.173	150.3027	0.03378933	2
5499.558	150.3568	0.03376124	1	5499.691	150.6411	0.03363475	2
5498.945	148.4898	0.03461167	1	5498.708	150.2798	0.03379063	2
5501.795	150.1787	0.03385513	1	5498.628	151.3692	0.0333055	2
5499.492	149.6872	0.03406354	1	5497.918	150.3175	0.03376881	3
5499.663	150.234	0.03381709	1	5499.193	151.0179	0.03346404	3
5500.308	149.9406	0.03395355	1	5498.81	149.7552	0.03402841	3

Figure 49: Raw Simulation Data

This data can then be copied into a .xlsm file titled “WhiskerResults”, which was contained inside the simulation repository obtained from GitHub previously. Inside this file are instructions that will produce histograms, and the risk assessment involved with the analyzed board. Once the raw data has been copied into the “WhiskerResults” file, the user should a format similar to the data depicted in Figure 50.

Copy and paste new Unity data below.							
298.625	2.33234	4.12385	1	797.613	6.43327	1.44774	1
234.075	2.40527	3.03942	1	577.434	5.68211	1.34353	1
150.199	4.48344	0.56131	1	214.628	1.82064	4.86408	1
34.504	2.90437	0.30728	1	5437.34	3.31164	37.2445	2
45.5618	2.78521	0.44121	1	409.633	3.55443	2.43566	2
160.746	6.94118	0.25063	1	332.493	2.71252	3.39468	2
19.0187	1.23628	0.93478	1	81.5415	1.63816	2.2826	2
258.642	4.47146	0.97177	1	56.5992	2.54944	0.65416	2
189.303	1.28257	8.64483	1	270.461	1.65912	7.38095	2
1531.53	1.42842	56.3865	1	402.077	2.92299	3.53522	3
62.7893	2.7378	0.62928	1	133.358	2.11385	2.24198	3
71.608	8.19879	0.08002	1	172.296	3.61279	0.99163	3
725.45	2.91612	6.40852	1	301.436	6.50395	0.53531	3
46.8027	3.40158	0.30386	1	204.567	1.97747	3.92988	3
72.6412	7.02549	0.11056	1	54.8071	1.14971	3.11476	4
1393.06	2.4075	18.0551	1	19.2166	9.14976	0.01724	4
200.653	3.41422	1.29308	1	173.018	4.32173	0.69589	4
1070.39	4.31052	4.32758	1	8.10114	3.31514	0.05537	4
652.171	4.27984	2.67466	1	161.056	1.20088	8.38957	4
577.766	2.84501	5.36224	1	41.2818	1.08942	2.61295	4
46.8904	3.46618	0.29319	1	165.95	1.68048	4.4144	4
30.3718	1.44439	1.09361	1	461.17	6.2458	0.88807	4
142.188	6.68722	0.23885	1	30.1957	1.96854	0.58536	4
129.773	5.46161	0.32682	1	153.531	2.74972	1.52539	4
74.2305	2.90219	0.66205	1	288.011	4.14752	1.25775	5
19.8655	3.833	0.10157	1	476.62	1.58571	14.2393	5
66.8773	3.76375	0.35465	1	716.574	5.38013	1.85967	5
287.827	3.75607	1.53259	1	325.767	3.15344	2.46094	5

ALT + F8 -> GenerateProb	
Overall Probability	
Individual Probability	
Iter #	Probability

Figure 50: Using WhiskerResults.xlsm

As stated in the instructions in the file, pressing the ALT key along with the F8 key will open the Excel macro list, and users will be able to select to generate frequency plots along with the probabilities of bridging across the whole simulation and per iteration. Once the macros have been run, the results will be formatted in a readable manner. Here, overall probability is the odds that an iteration would have formed a bridge, and the individual probabilities are the odds of bridging per iteration. A 100% overall probability means each iteration had a bridge be detected. Figures 51 through 52 show the probability and frequency charts once a simulation has been completed.

ALT + F8 -> GenerateProb	
Overall Probability	
100.00%	
Individual Probability	
Iter #	Probability
1	0.60%
2	1.20%
3	1.00%
4	2.00%
5	1.80%

ALT + F8 -> GenerateFreq			
Length	Diameter	L:D Ratio	Resistance
797.613	6.433271	123.982497	1.44774
577.4344	5.682107	101.623289	1.343526
214.6283	1.82064	117.886183	4.864083
5437.337	3.31164	1641.8865	37.24451
409.6334	3.554434	115.245747	2.43566
332.4931	2.712521	122.577152	3.394678
81.54153	1.638159	49.7763221	2.282595
56.59921	2.549437	22.2006702	0.6541598
270.4611	1.659118	163.014987	7.380952
402.0768	2.922991	137.556633	3.535218
133.3579	2.113854	63.0875642	2.241976
172.2962	3.612794	47.6905686	0.991634
301.436	6.503949	46.3466119	0.535307
204.567	1.977467	103.449008	3.929879
54.80706	1.149706	47.6705001	3.114762
19.21662	9.149758	2.1002326	0.01724328
173.0178	4.321728	40.0344029	0.6958857
8.101142	3.315135	2.44368389	0.05537399
161.0557	1.200879	134.114844	8.389572
41.28175	1.089418	37.8933981	2.61295
165.95	1.680482	98.7514296	4.4144

Figure 51: Probability Results

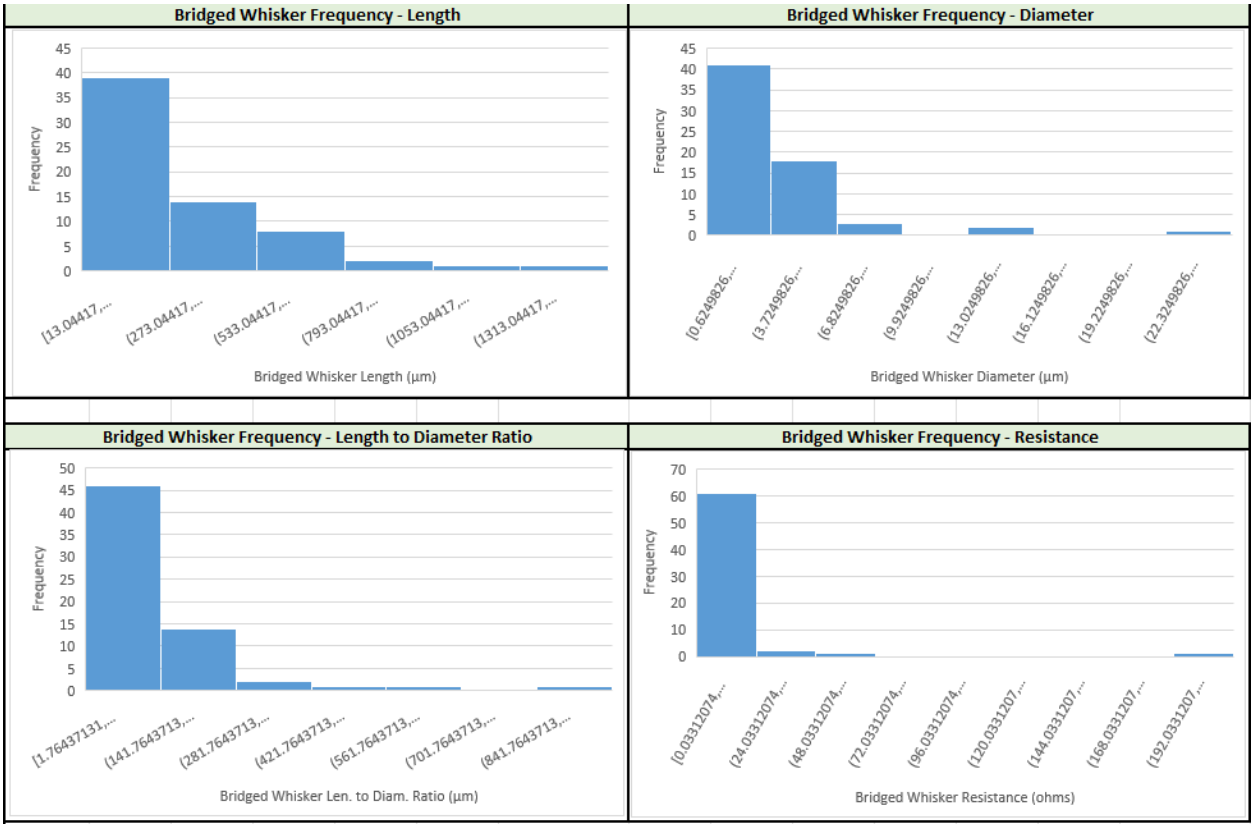


Figure 52: Frequency Results

Additionally, new data can be saved for the same board by simply changing the File Name input, pressing Save, and running additional simulations.

Testing and Analysis

To ensure that the simulation performs as expected, testing of each design element and boundary constraint is necessary. The simulation must be robust enough to handle changes to design elements such as the circuit board in use, whisker distribution, or external force, while accurately reflecting results due to these variations. However, while the user should be able to fully customize the simulation to their needs, errors can arise for extreme or unexpected input values. These errors can result in various processing problems, such as decreases in frame rate, memory overloading, or render overloading—all of which typically require the program to be forced quit. Several hard-coded functions were added to account for user variation of design elements while also making sure that boundaries were in place to prevent performance issues. These features will be tested to validate their implementation and ensure that the user does not encounter any obstacles while running the simulation.

Varied CCA Designs

When the user imports their circuit board model, it is important to make sure the board is being appropriately scaled to real-world expectation. This is crucial, as it is known that metal whiskers are measured in micrometers, requiring the imported boards to be at the proper scale for accurate bridging results. Additionally, conductive materials must also be registered, regardless of the number of conductive surfaces or components used (resistors, capacitors, transformers, etc.).

In verifying appropriate scale, two blank circuit boards were created in Altium: a 5 cm x 5 cm board and a 5 in x 5 in board. The same process mentioned earlier was used to get the board to a particular size. Since whisker dimensions are represented in micrometers and Unity operates under the metric system, any boards designed using imperial units must be imported and adjusted to match the metric system accordingly. To aid in this, the team implemented a 1 m x 1 m grid with individual 1 cm x 1 cm units spanning the full area. Using the grid feature and adjusting the camera to a top-down perspective, both boards were measured to verify their conversion. Figure 53 reveals the 5 cm x 5 cm board, which correctly measures five units in both length and width upon visual observation.

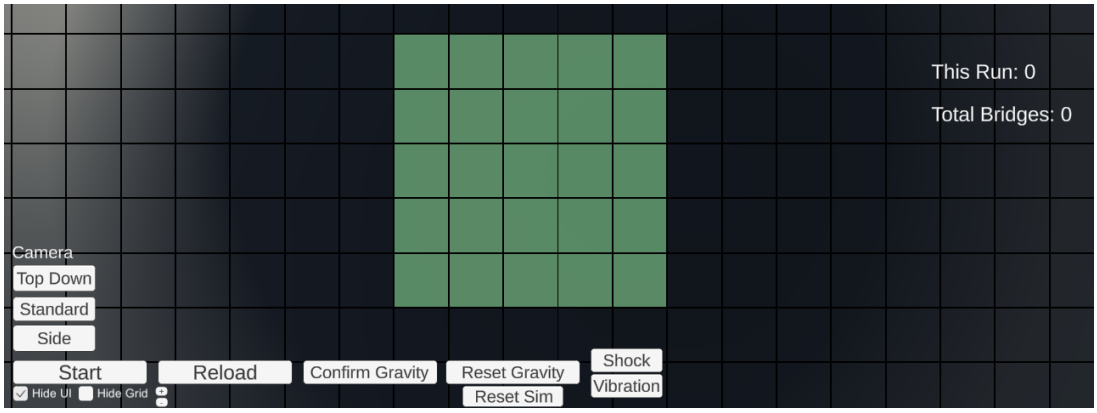


Figure 53: Test Scale for 5 cm x 5 cm Board

Figure 54 reveals the 5 in x 5 in board, which is equivalent to a 12.7 cm x 12.7 cm board.

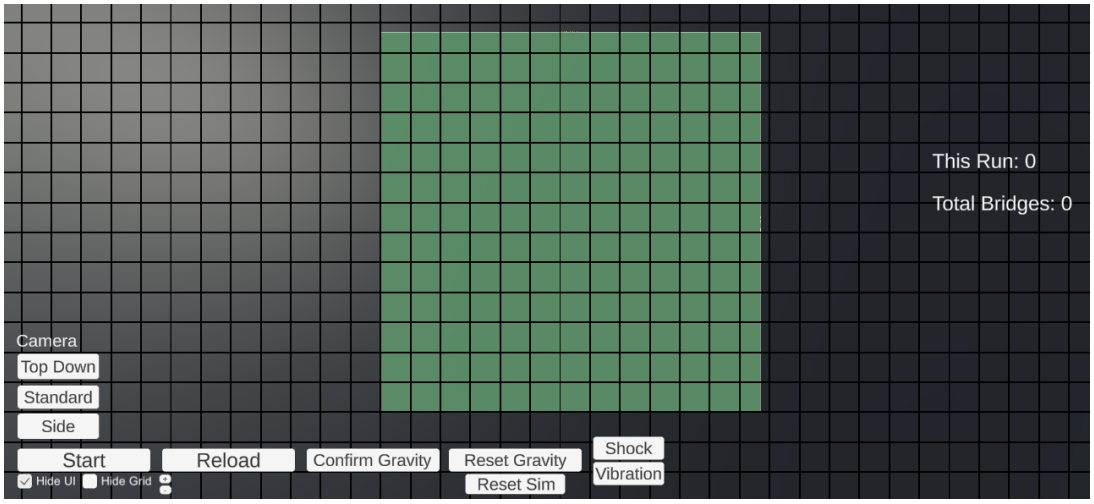


Figure 54: Test Scale for 5 in x 5 in Board

Upon visual observation, the board correctly reads approximately 12.7 units in both its length and width. Therefore, the simulation appropriately scales the boards without any user adjustment.

In testing the ability to register conductive materials on the circuit board, a circuit board in Altium was populated with various placement of capacitors, diodes, and resistors. It is known that the tinned-copper leads of capacitors, copper pads that the diodes and resistors rest on, and exposed copper traces on the circuit board are conductive. The automatically generated name of the material was edited to tinned copper. To account for these conductive surfaces, the associating names for these materials were entered into the conductive material input feature. These materials were then scanned throughout the board and registered as conductive triggers for the simulation to interact with. Figure 55 reveals a close view of the capacitor leads, the diode and resistor pads, as well as the exposed traces after their respective conductive materials were identified.

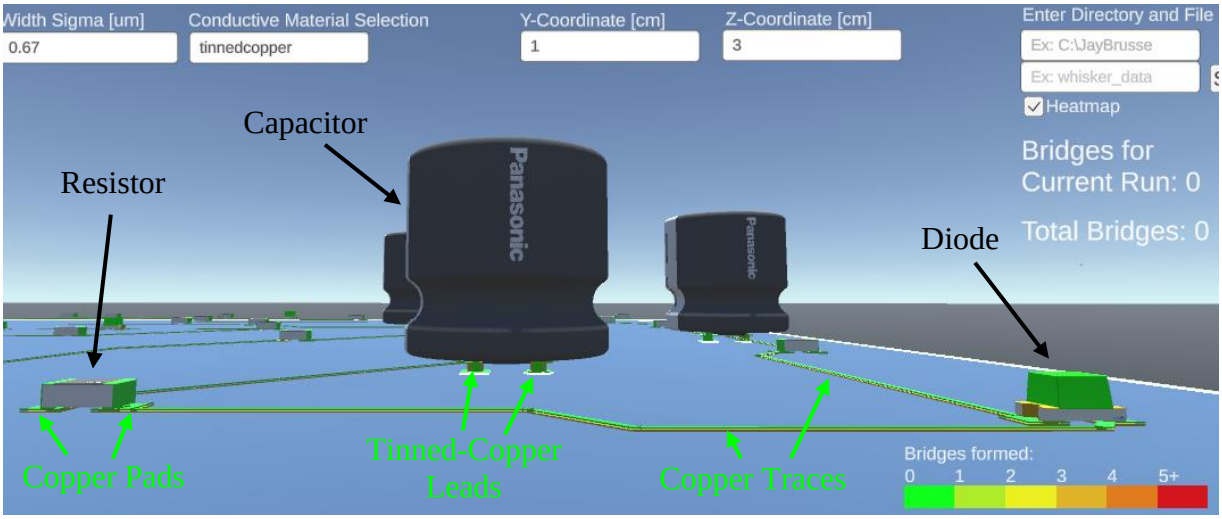


Figure 55: Test Conductive Material Input

From this figure, it is evident that the desired conductive materials were appropriately handled and registered as conductive by the simulation. This is proven by toggling the heat map, which highlights all conductive surfaces that can be bridged by whiskers. The conductive materials are shown highlighted in green, since the simulation has not started, and zero bridges have occurred yet.

Distribution Validation

Generating whiskers is one of the primary design elements within the simulation, and it is necessary to make sure that they correctly represent normal and lognormal distributions based on which one is selected. To test this feature, a normal and lognormal simulation were both ran under the same mu, sigma, and quantity inputs. The resulting whiskers were stored into excel files to create a histogram of their lengths and assess their graphical distribution. Figure 56 represents the normal distribution, which has been selected from the dropdown. Figure 57 displays the resulting histogram created from the lengths of all whiskers generated.

Figure 56: Test Inputs for Normal Distribution

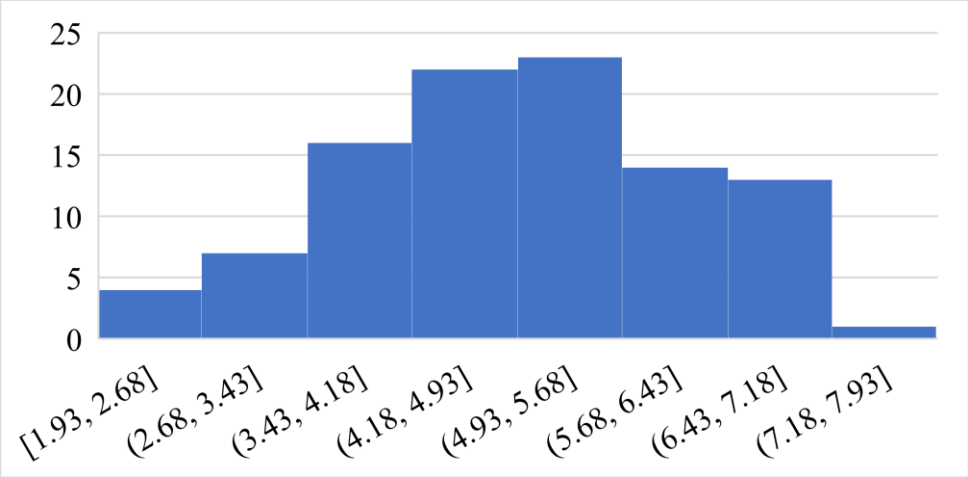


Figure 57: Test Results for Length Normal Distribution

Figure 58 and Figure 59 represent the same approach, but for a lognormal distribution.

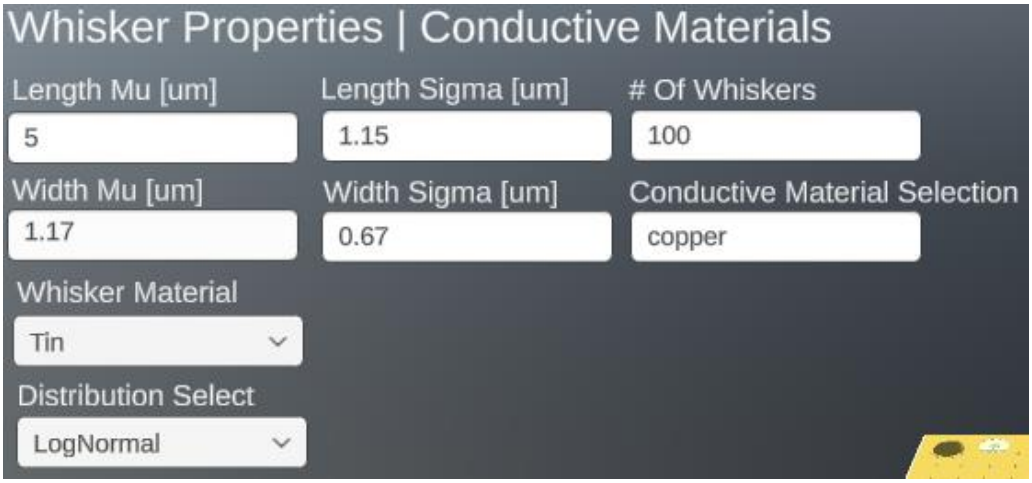


Figure 58: Test Inputs for Lognormal Distribution

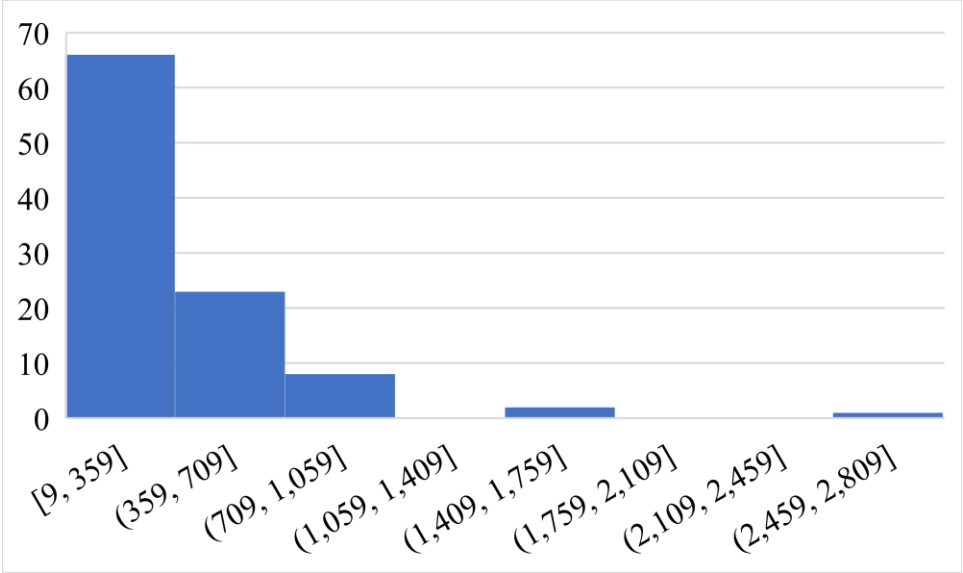


Figure 59: Test Results for Length Lognormal Distribution

The shape of the histograms in Figures 57 and 59 indicate that a normal and lognormal distribution were successfully generated by the simulation. The normal distribution follows a bell curve, while the lognormal distribution follows a skewed and asymmetric curve, both of which correctly correspond with Figure 11.

Numerical Input Handling

In conjunction with whisker distributions, their numerical inputs must be appropriately handled as manipulation of whisker lengths, diameters, and quantity can cause significant errors if boundaries are not set. The most common errors associated with these inputs are whiskers spanning almost infinitely in length (where Unity no longer can process the distance) or being so large that they spawn within each other, continuously colliding with one another until they fully separate into open space. Figure 60 demonstrates a simulation that was run with a normal distribution and created whiskers much larger than they should be represented by.

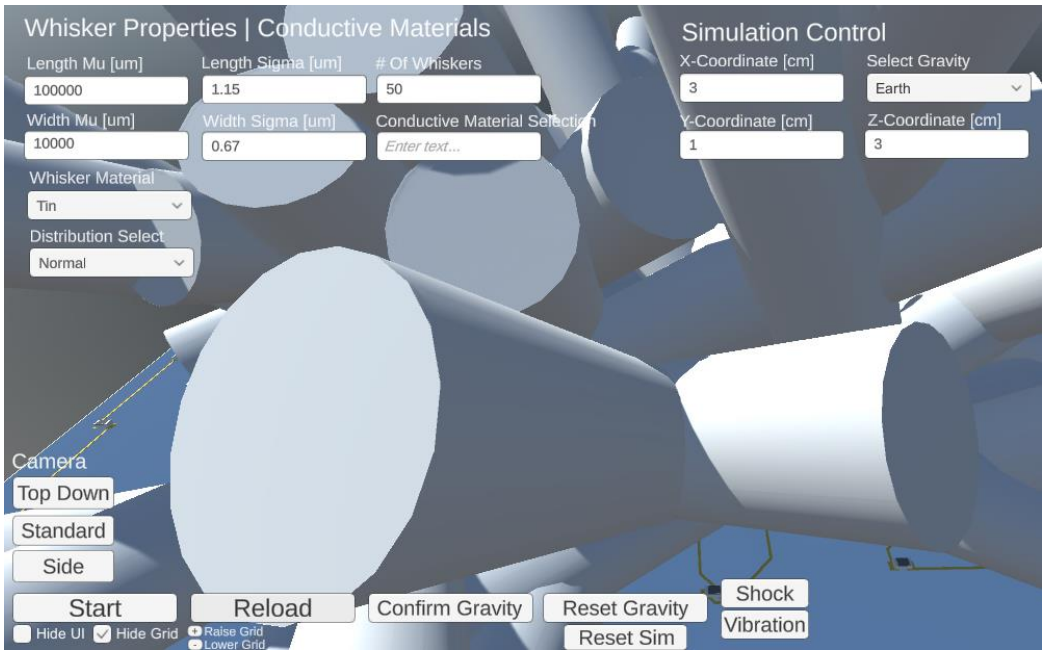


Figure 60: Test Results for Improper Inputs

The whiskers in this figure are so large that they rapidly spawn within each other, thus causing the simulation to freeze because Unity's memory cannot handle the massive objects and repeated collisions.

To prevent similar instances from occurring, if-statements were placed within the respective code to display error messages on the screen and prevent the simulation from starting.

With the option to use lognormal or normal distributions, mu, sigma, and quantity input limits were carefully considered and set so that generated whiskers could not have lengths or diameters that cause collisions with each other when spawned in. Figure 61 demonstrates the error handling for these boundary constraints, where the same inputs from the previous simulation are refrained from creating whiskers, and an error message (displayed in red text) is shown for the input responsible.

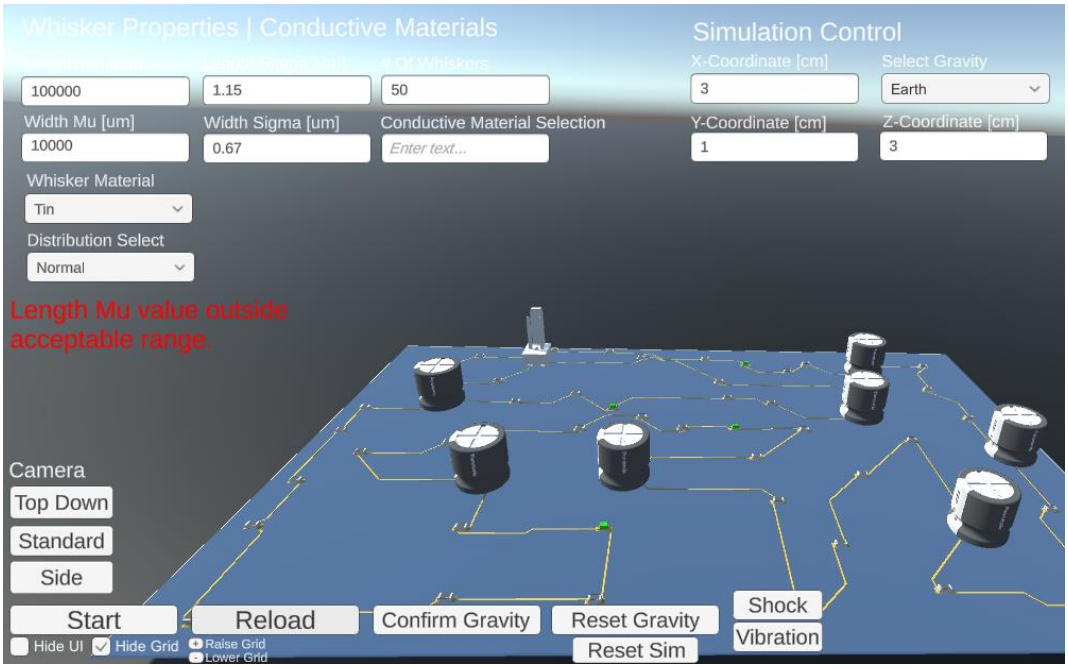


Figure 61: Test Error Handling for Improper Inputs

Other numerical inputs, such as the coordinate location to drop whiskers, external force inputs, and the number of iterations to simulate, do not pose such significant threats to performance. However, for the simulation to properly function under these design elements, their inputs were also restricted. The drop location and external force inputs were limited to positive float values, while the input for number of iterations must be positive integers. Error messages were similarly hard coded to ensure that only acceptable values can be used.

External Force Effects

Forces applied to a circuit board can severely affect the results of bridging whiskers. While changes in gravitational acceleration certainly affect their behavior in falling and settling into a particular spot, vibration and shock play a more significant role in affecting bridging probability. To test the successful integration of these forces in representing real world-physics, vibration and shock were applied to a circuit board to assess the before and after effects of bridging upon their application.

When shock or vibration are applied perpendicular to a circuit board, whiskers are sent airborne, causing them to relocate. This was tested in a single iteration using 200 lognormally distributed whiskers, where, for a single iteration, both shock and vibration were applied perpendicular to a circuit board with exposed copper traces and pads registered as conductive. A single iteration simulation was run to test vibration under inputs of 3 Newtons for amplitude, 5 Hertz for frequency, and 10 seconds for the duration. Figures 62 and 63 respectively reveal the before and after effects of an applied perpendicular vibration.

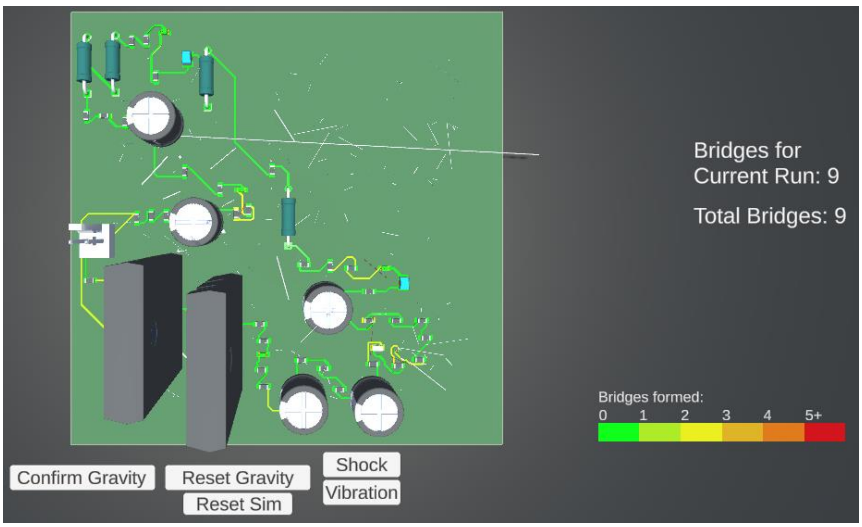


Figure 62: Test Perpendicular Vibration Before Activation

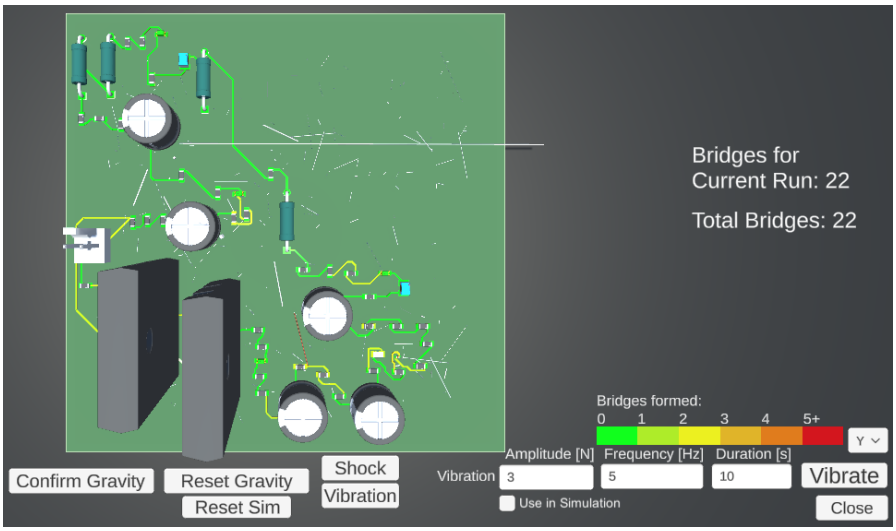


Figure 63: Test Perpendicular Vibration After Activation

A new simulation was run where a perpendicular shock was applied with a 3 Newton amplitude, using the same whisker parameters and drop location as the vibration simulation. Figures 64 and 65 respectively reveal the before and after effects of bridging due to an applied perpendicular shock.

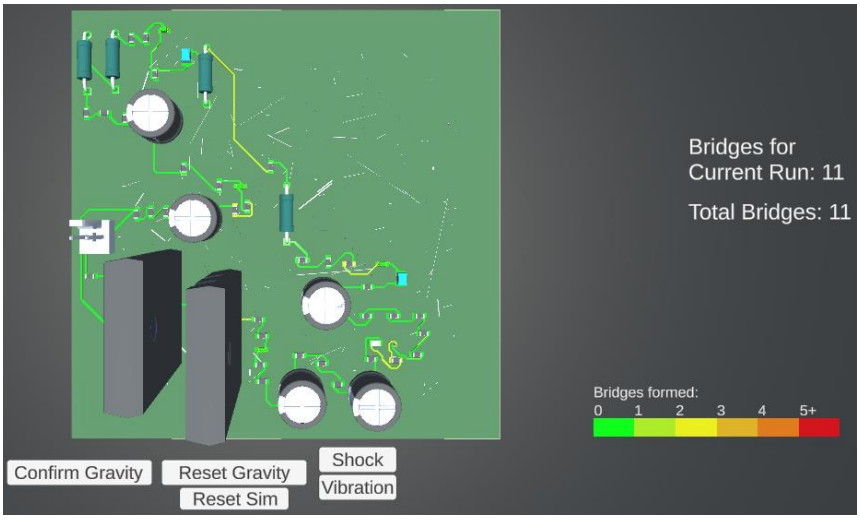


Figure 64: Test Perpendicular Shock Before Activation

Figure 65: Test Perpendicular Shock After Activation

Evidently, Figures 62 through 65 reveal that the number of bridges, shown on the right-hand side in white text, increased after both forces were applied. This is to be expected, as the original placement of whiskers prior to their application were shuffled and rearranged to create another series of bridges. The heatmap also reveals this uptick in the number of bridges. As expected, the constant vibration continuously rearranged the whiskers into various bridging locations, while the shock only rearranged the whiskers a single time to a new location. Vibration experienced far more bridging than the singular shock applied.

However, when both shock and vibration are applied parallel to a circuit board, whiskers are relocated, but in different manners. Under parallel vibration, inertia and friction play a larger role in how much whiskers can be displaced. Similar to the renowned tablecloth trick, where a set of dishes remain in place when the tablecloth underneath is quickly yanked, whiskers will hardly move if the vibration frequency is high enough. This is because while the board oscillates back and forth, the inertia of the whiskers does not change, which follows Newton's First Law of Motion. Along with this, friction between the tablecloth and dishes is minimal due to the cloth's smoother surface, and the short time that friction is accounted for while the cloth is being

removed [42]. The same applies to whiskers, where friction between the board's surface and the whiskers is practically null under a high frequency. To test if the simulation reveals the same truth, a single iteration using 200 lognormally distributed whiskers was ran, where both a high frequency and low frequency parallel vibration was applied. The high frequency vibration was first tested using inputs of 3 Newtons for amplitude, 10 Hertz for frequency, and 10 seconds for duration. Figures 66 and 67 respectively reveal the before and after effects of whisker position and bridging results under high frequency parallel vibration.

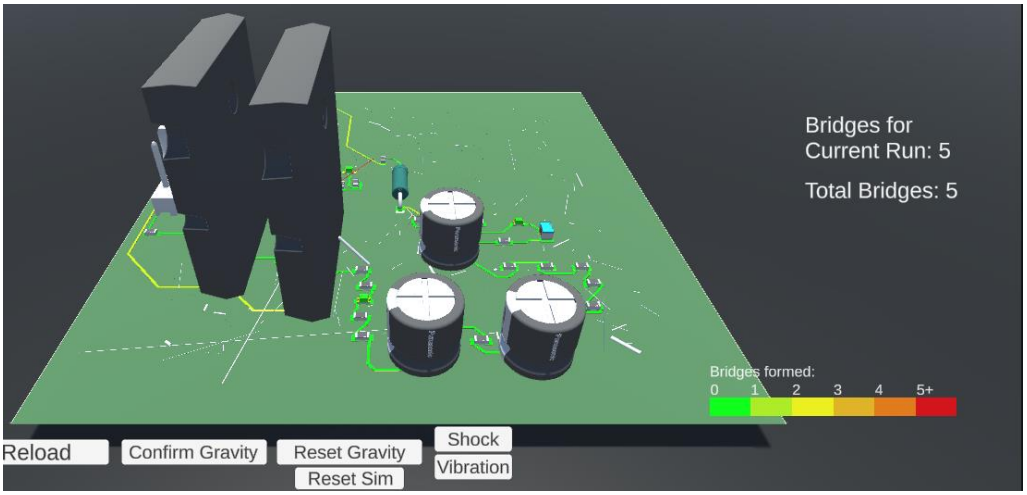


Figure 66: Test High Freq. Parallel Vibration Before Activation

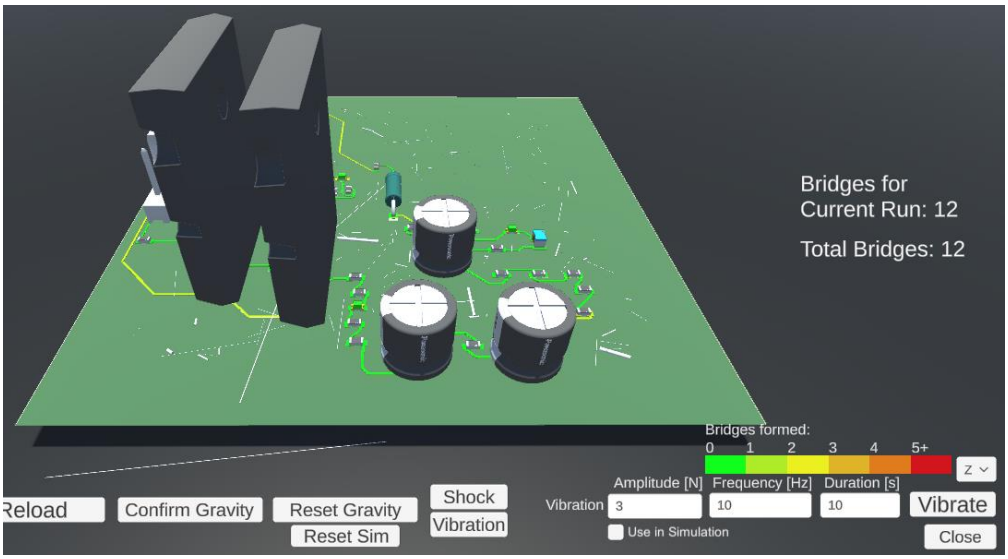


Figure 67: Test High Freq. Parallel Vibration After Activation

The low frequency vibrations used a 1 Hertz frequency with the same amplitude and duration as the previous simulation. Figures 68 and 69 respectively reveal the before and after effects of whisker position and bridging results under low frequency parallel vibration.

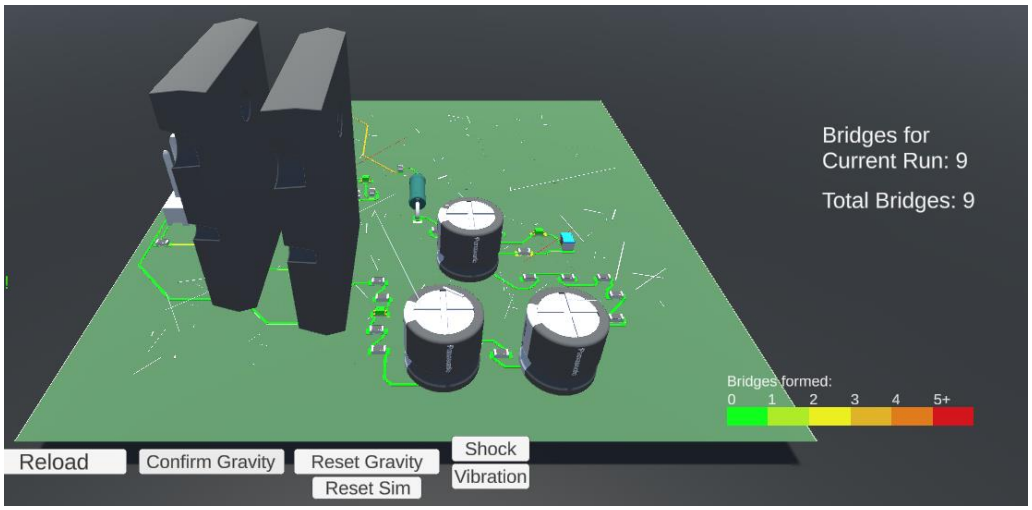


Figure 68: Test Low Freq. Parallel Vibration Before Activation

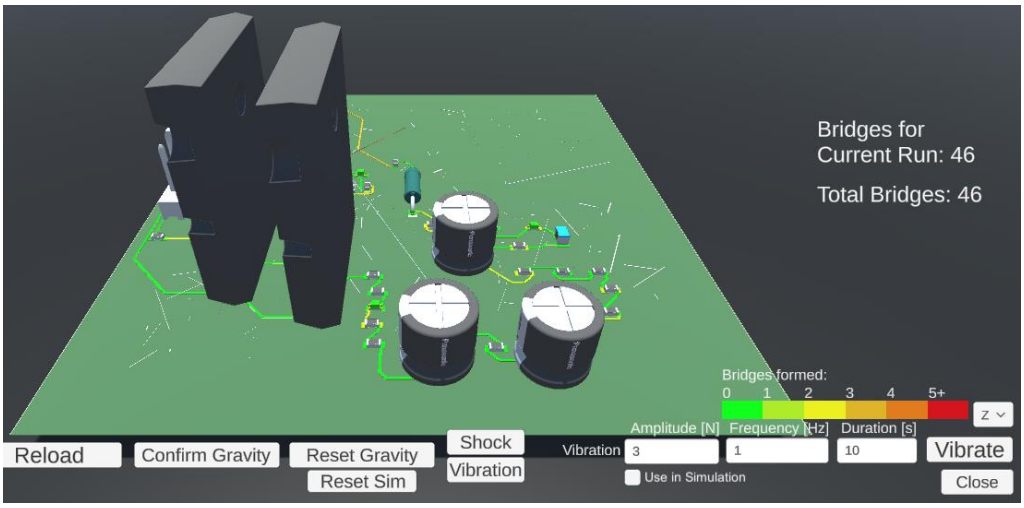


Figure 69: Test Low Freq. Parallel Vibration After Activation

From observation of Figures 66 and 67, the whiskers experience a very minimal change in their position and orientation under high frequency parallel vibration. The number of bridges does not increase much at all, even though 200 whiskers are still present on the board. The high frequency in the vibration kept the inertia of the whiskers constant, while also significantly decreasing any friction between the board and the whiskers. Unless the whiskers were pushed in the direction of the vibration by components extruded from the board's surface, they remained relatively still. Since shock has a constant high frequency value, parallel application of this force experiences the same results as high frequency vibration, where there is little to no movement of whiskers due to its instantaneous impact. Unless the whiskers were pushed in the direction of the vibration by components extruded from the board's surface, they remained relatively still.

In Figures 68 and 69, under low frequency parallel vibration, the whiskers traveled further from their original location since there is more time for friction to act on the board's surface. Due to the repetitive oscillations, and low input frequency, the whiskers had enough time to roll back and forth into bridged and unbridged connections, significantly increasing the number of bridges. This low input frequency changed the inertia of the whiskers and allowed

friction to be applied between the board and whiskers. By the time the vibration had finished, the whiskers returned to their original position.

Bridging Registration

As the simulation is intended to track whether a bridge has occurred, it is vital that whiskers only register as bridged if contact between two conductive surfaces is truly made. Otherwise, results will not truly represent the probability that a circuit board experienced bridging. To test appropriate bridging registration, a set of 500 lognormally distributed whiskers were generated and dropped onto the same circuit board shown in the previous section. Several bridged whiskers were then examined to ensure they are truly contacting exposed conductors. Figure 70 represents a top-down perspective in microscopic view of several bridged whiskers out of a total of 500 lognormally distributed whiskers that contacted exposed copper traces.

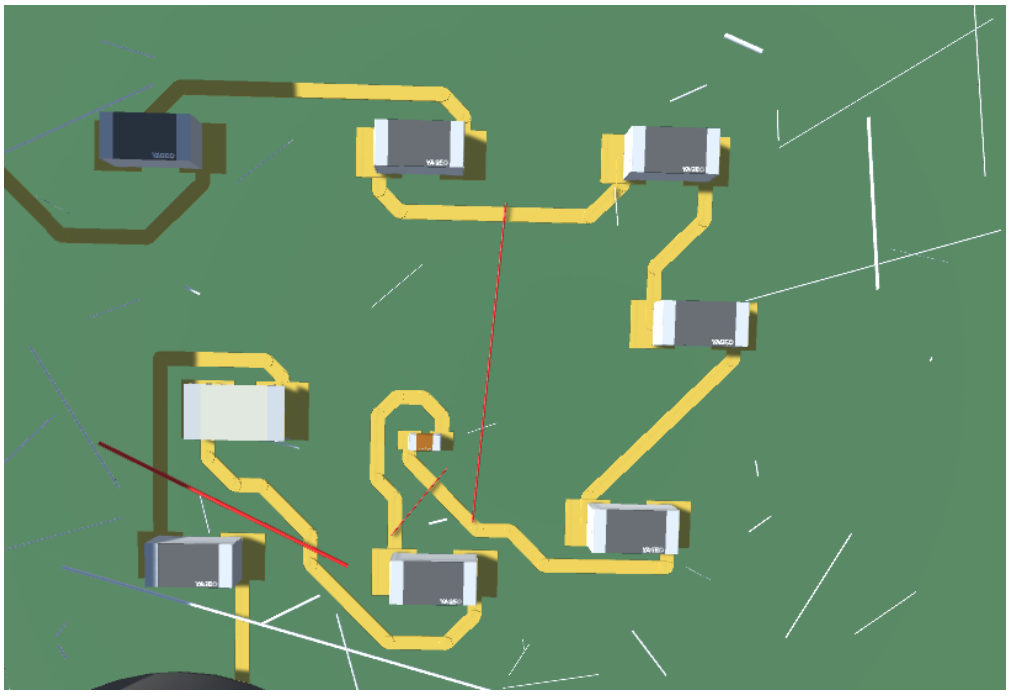
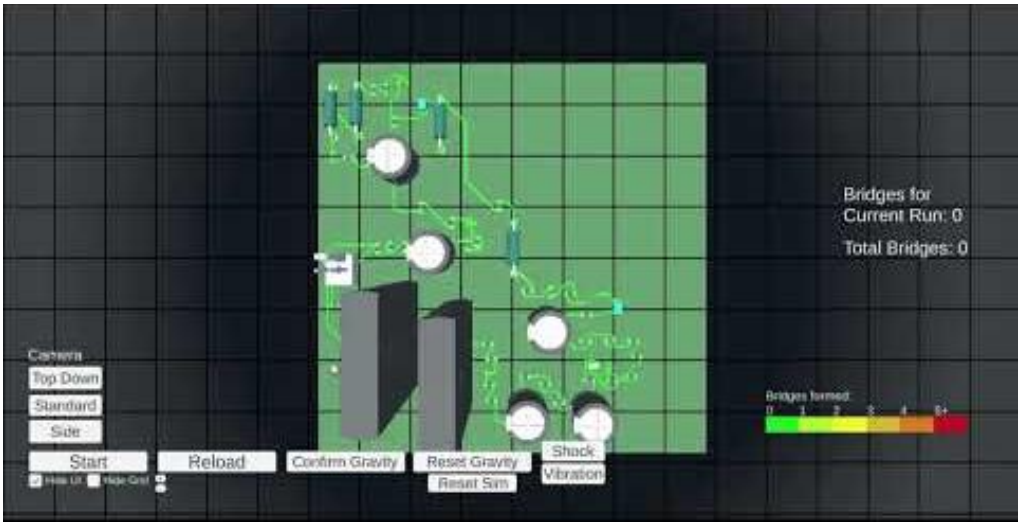


Figure 70: Test Bridging Contact

The figure reveals three bridged whiskers, that the simulation has highlighted in red. The red whiskers are contacting two exposed copper surfaces, forming a bridge. It is also important to note that whiskers outside of this area, with some touching the copper traces, and others not, have not formed a bridge and have remained grey. Therefore, bridging registration is satisfied.

The heat map tool also depends on proper bridging registration. It is known that the heat map is intended to highlight exposed conductors and reveal the frequency of bridging experienced by each conductor. This tool was tested to ensure that it appropriately updates as whiskers continue to form bridges. Object 3 reveals a video where a simulation of 200 lognormally distributed whiskers dropped on top of a circuit board over a course of 10 iterations, with copper used as the conductive material.



Object 3: Test Heatmap Display – Youtube [43]

As each iteration completes every 5 seconds, it can be seen from the video that the heat map continues to get hotter in certain locations as continuous storms of whiskers drop onto the exposed conductors. This is to be expected and means that the heat map correctly updates as whiskers continue to bridge. Users can apply this to their benefit in understanding the likelihood

that the “hot” spots will encounter bridging and reconfigure their circuit board according to the severity of short circuits occurring in that location.

Results Assessment

Providing valuable results is just as important as creating a physical simulation. It is necessary to give users meaningful and justifiable results in understanding the likelihood of their circuit board experiencing whisker bridging. In testing the validity of the results captured by the simulation, two tests were run; a three-iteration simulation with a small quantity of whiskers generated, and another with a large quantity of whiskers generated. This is to serve as a sanity check in verifying that the probability and frequency calculations are true to real-life expectation. The board used in the previous two sections is shown in Figure 71, which will be subject to the two tests.

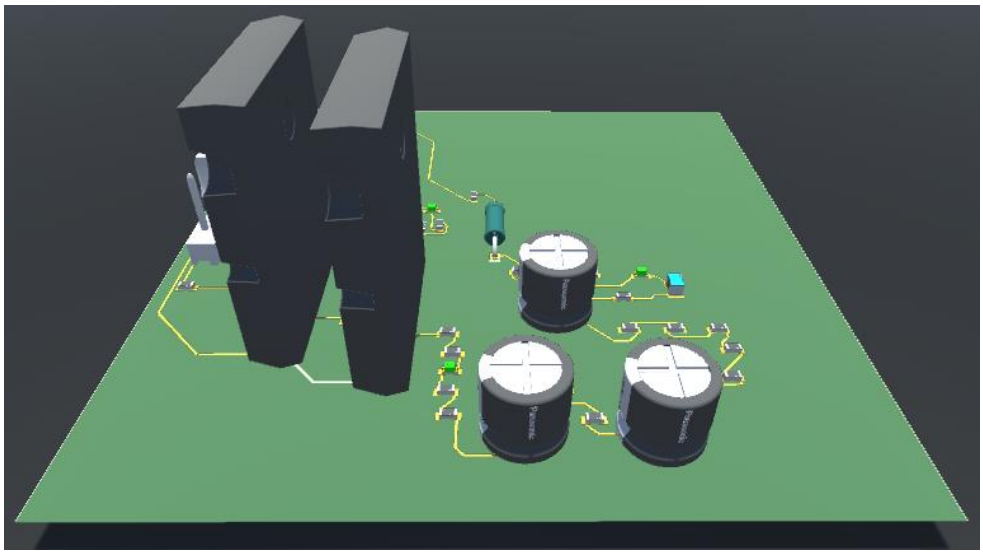


Figure 71: Test CCA for Results Analysis

In the first simulation, a lognormal distribution of 50 whiskers using a length mu of 5 μm and sigma of 1.15 μm and width mu of 1.17 μm and sigma of 0.67 μm were generated. The coordinate location for these whiskers to drop from spanned the full length and width of the

circuit board, which in this instance was 3 inches (7.62 cm) for both parameters. The only conductive circuit board material to be registered was copper, which was appropriately entered into the relevant text box. After completing the simulation, the saved Unity excel file revealed only three whiskers out of the total 150 generated under all three iterations created bridging on the circuit board. Iteration one encountered a single bridge, iteration two having two bridges, and the third resulting in zero bridges. The saved data was brought to the WhiskerResults excel file containing macros to calculate probability and frequency values.

Figures 72 shows the probability calculations for this simulation.

ALT + F8 -> GenerateProb	
Overall Probability	
66.67%	
Individual Probability	
Iter #	Probability
1	2.00%
2	4.00%
3	0.00%

Figure 72: Test Small Qty Bridging Probability

From the figure, the macro has correctly calculated the individual probability. This can be proven by hand-calculation. For instance, since 50 whiskers were generated for each iteration, and iteration one had one bridged whisker, the probability of bridging can be calculated using Equation (10):

$$Individual\ Probability = \frac{\#\ of\ bridged\ whiskers}{\#\ of\ whiskers\ generated} \cdot 100$$

$$Individual\ Probability = \frac{1}{50} \cdot 100$$

$$Individual\ Probability = 2\%$$

Both the hand-calculated and macro-calculated probability for bridging in iteration one check out, being 2%. Additionally, the overall probability was also correctly calculated. Knowing that only two of the three iterations faced at least one instance of bridging, the overall probability can be calculated using Equation (11):

$$Overall\ Probability = \frac{\#\ of\ bridged\ simulations}{\#\ of\ simulations\ ran} \cdot 100$$

$$Overall\ Probability = \frac{2}{3} \cdot 100$$

$$Overall\ Probability = 66.67\%$$

Evidently, the hand-calculated and macro-calculated probability of bridging for the entire simulation also checks out, being 66.67%.

Figure 73 displays the resulting frequency histogram for the frequency of bridged whisker resistances. The remaining length, diameter, and length to diameter ratio frequency histograms were left out for viewing purposes.

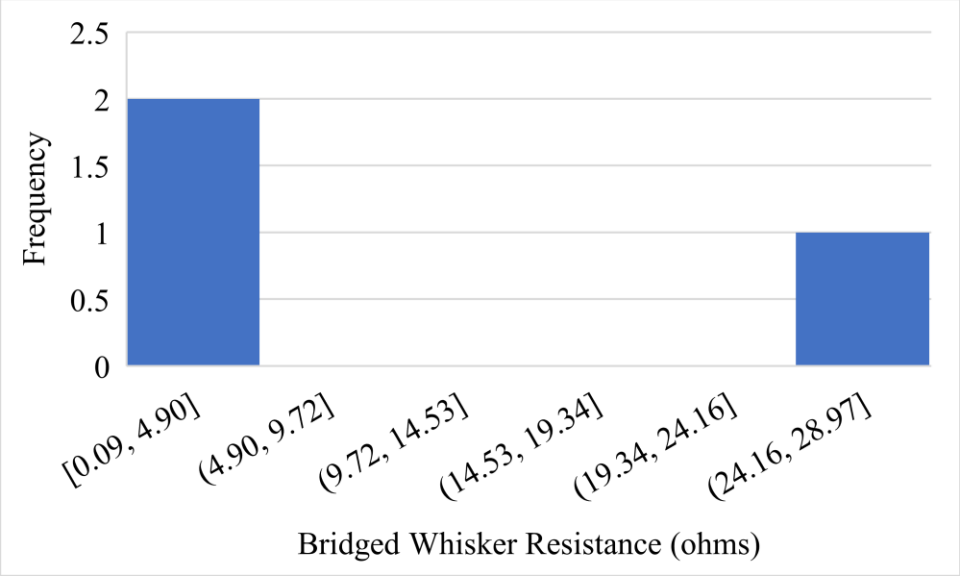


Figure 73: Test Small Qty Bridging Resistance Frequency

Figure 73 accurately represents the frequency values, which indicates that bridging whiskers on a given circuit board will most likely have a resistance of 0.09 to 4.90 ohms under the given input conditions. These results are less reliant upon built-in macros, and more contingent upon the user selecting the corresponding data and inserting their histogram into the appropriate location. For a simulation that was run with a small quantity of whiskers, these results are to be expected.

The same approach was taken in the second simulation, where the only difference in user input was the quantity of whiskers. For this simulation, 500 whiskers were generated per iteration onto the same board, using copper as the exposed conductive material. After completing the simulation, 19 bridges out of the total 1500 whiskers generated formed a bridge on the circuit board. The saved Unity excel file revealed that 9 whiskers bridged in simulation one, 5 bridged in simulation two, and 5 bridged in simulation three.

Bringing the saved data to the "WhiskerResults" Excel sheet, running the macros, and inserting histograms gave the following probability results shown in Figure 74.

ALT + F8 -> GenerateProb	
Overall Probability	
100.00%	
Individual Probability	
Iter #	Probability
1	1.80%
2	1.00%
3	1.00%

Figure 74: Test Large Qty Bridging Probability

As expected, the probability calculations are correct in Figure 74, and backed by similarly using Equations (10) and (11) as shown in the previous sample hand-calculation. With a large quantity of 500 whiskers dropped per iteration, it is expected that the board will encounter at least one bridge per iteration.

Additionally, more bridged whiskers mean a better spread in the frequency results. Figure 75 displays the resulting frequency histogram for the frequency of bridged whisker resistances.

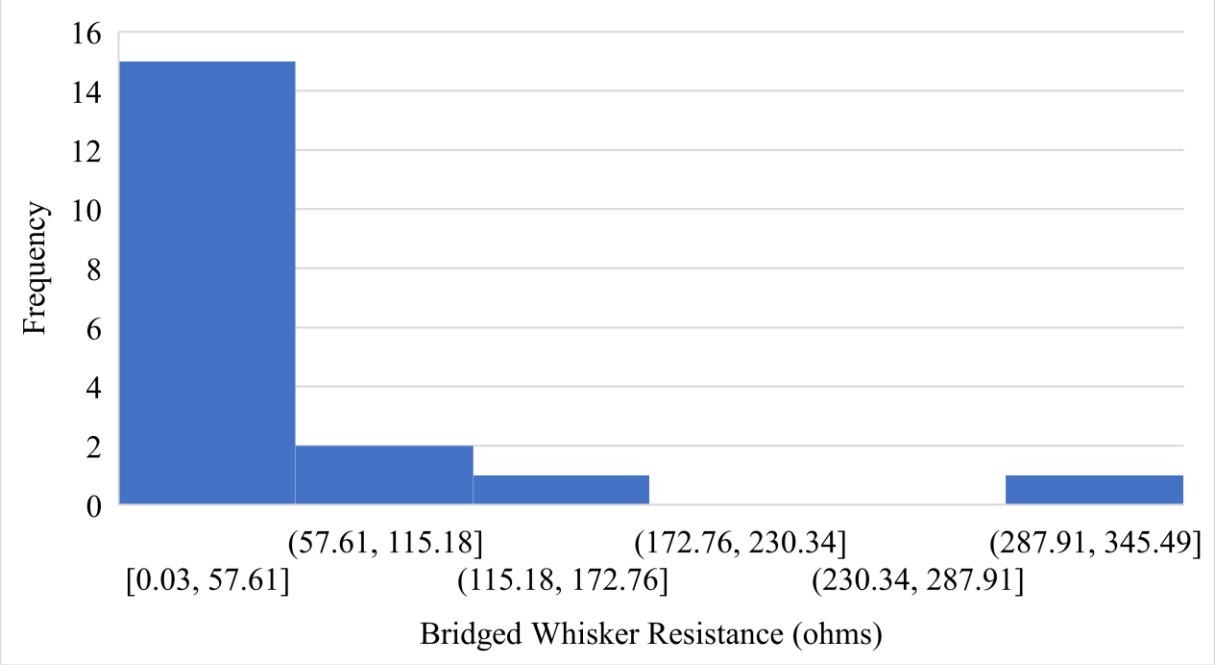


Figure 75: Test Large Qty Bridging Resistance Frequency

It can thus be inferred from Figure 75 that bridging whiskers on the given circuit board tend to have a resistance of 0.03 to 57.61 ohms. This is indicative to the engineer that resistances within this range can be expected from a bridging incident, but a resistance up to 287.91 to 345.49 ohms can also occur.

Sources of Error

While the simulation has gained growth over its development and accounts for several real-world functionalities, it is not fully accurate to what would occur in reality. A design engineer needs to take precaution in understanding the potential sources of error within the simulation as they use this tool for bridging risk-analysis. The following problems and ideas

must be taken into consideration when either running the simulation or using the generated results.

The team's project constraint—modeling the whiskers as rigid cylinders—becomes a source of error itself. The team's sponsor approved of this stipulation in the project, which helps to simplify the development process in Unity. However, if a design engineer were to directly apply the simulation results to real life, they would need to consider the fact that whiskers are not rigid cylinders. In reality, metal whiskers typically have a level of flexibility, allowing them to bend and curl in multiple directions. With the lack of whisker flexibility in the simulation, their spatial interaction is not fully accurate, thus affecting how they would truly behave in creating a bridge. Rigid cylinders do not bend or flex, so the accuracy is affected in regards to if a flexible whisker of the same geometry would create a bridge. Additionally, the rigid cylinder constraint assumes that the cross section of metal whiskers is constant and circular throughout their lengths. This is also not always the case, where in reality their cross-sections can mold into various shapes. This would similarly affect the accuracy in bridging results, as well as the resistance encountered due to bridging.

Error related to whisker resistance is also present. Equation (4), which was used in the simulation, calculates resistance generated by a bridged whisker and relies on the whisker's length. The team's use of this formula assumes that the full whisker length is used to calculate resistance under bridging. However, according to the team's subject matter expert, Jay Brusse, the full whisker length does not always have current flowing through it. In situations where the full length of the whisker is not responsible for bridging, its resistance to current is only the distance(s) formed between the bridged conductors. Figure 76 represents this idea, where a

whisker (seen in red) has bridged, but only a portion of its full-scale length ($L_{whisker}$) spans the bridging length (L_{bridge}) between both conductors.

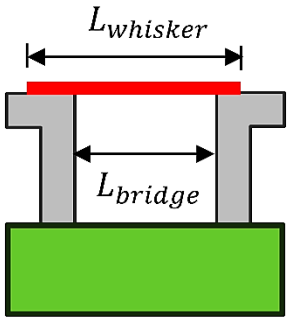


Figure 76: Bridged Whisker Resistance Length

From this figure, it is evident that current is only running through the length of L_{bridge} as the whisker applies resistance to the two exposed surfaces. Therefore, it is not the full-scale length of the whisker that is to be used in this formula, rather the distance between both contact points on the whisker where bridging has been made. For the sake of the team's simulation, the full-scale whisker length was substituted into this part of this equation. This was due to potential complexity that was beyond the original scope of the project. As the length changes, there would be an effect on the calculated resistance amount.

The shock feature that can be applied to a circuit board was not implemented as a half-sine wave, which is typically how shock is understood. During development, the vibration feature was successfully implemented and behaved as expected, however when attempting to incorporate shock as a half-sine wave, the whisker game objects would not react. The team's technical advisor, Dr. George Flowers, recommended the shock feature be modeled as a sine wave, with hard-coded values for frequency and duration. In the real world, there is no guarantee that every shock the board would encounter could be modeled like this.

When importing boards, traces are not always exposed, and are instead covered by a protective coating, or solder mask. By default, if imported boards from Altium are designed with traces, the traces act as if they are exposed on the top of the surface. If this is what the user desires, then there are no issues, and the simulation will perform fine. However, if the user intended them to remain covered and not exposed, then the simulation will render them as conductive surfaces and attach colliders for bridging. These colliders can also pose a problem with certain Altium board designs due to the configuration of traces, which can cause colliders to group together. Though this doesn't always happen, depending on the orientation/placement of traces, their colliders can line up incorrectly. This evidently can affect bridging results if bridges are detected on surfaces that are not conductive. Figure 77 reveals this phenomenon, where the outer trace's geometry is highlighted in orange, but the colliders line up in such a way that hits can be generated in areas that are not conductive in reality.

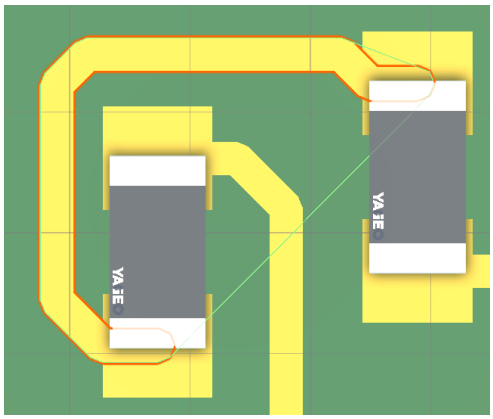


Figure 77: Trace Net Collider Error

The conductive area goes around the orange trace and connects to the green line. This poses a problem, as there is space within this area that is not conductive yet can still cause a bridge in the simulation.

Along the lines of board import settings, the .obj file format has potential problems with certain Altium components. Unity only has compatibility with .obj and .fbx files. The .fbx files

generated a problem that will be discussed in the following section, so the team remained using .obj files for development. While .obj files have some benefits, a drawback is that they can load incorrectly across different platforms [44]. This problem was found during development. Figure 78 shows an example of this with a transistor.

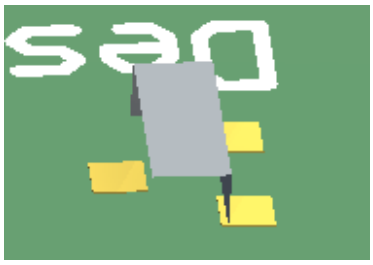


Figure 78: Transistor Construction Error

Additionally, .obj file types creates a “grouping” among all board components, such as resistors or capacitors. This phenomenon occurs when all parts of a component get grouped together. If one material of the component is labeled conductive, the entire component gets labled as that. When the simulation is run, if the whisker makes contact with any part of two or more components, a hit will be registered regardless if the contact points are actually conductive. For example, a chip resistor placed on a board may only have conductive contact points on the pads that it rests on, but the entire resistor object as a whole is labeled conductive.

Altium also has complications when giving proper nomenclature to board materials when exported as .obj and .mtl files for Unity to process. Most materials tied to components on an imported CCA are formatted as “mat_number” (e.g., “mat_19”, “mat_20”). The team was not sure why this was the case, which complicated the conductive material identification process.

Further Development

Though the errors present in the simulation can affect the accuracy in certain circumstances, potential solutions exist within various physical tools available to Unity. This

section will cover potential ideas and solutions to amend a few of the errors discussed in the previous section.

To solve the inaccuracies related to the whisker properties, a different whisker generation process could be considered. For a potential solution, the PolyShape tool could be researched to model different shapes [45]. This tool could add at least a little cross-sectional variation. More research would be needed specifically to see, assuming this tool can be used, what other steps would be needed to incorporate this tool into the simulation. A potential idea is to model one whisker with more unique geometry and duplicate it as necessary. Geometry nodes could also be looked at for modeling a whisker with different faces. More research is required to see if this feature could be applied to a cylinder [46]. Assuming it is possible, more research would have to be done to see what else would be need to be done to work with the rest of the simulation. If this method were chosen, additional variances in the cloned whiskers would be necessary to achieve more accuracy, which could take a while to configure in Unity, if possible at all. Also, different methods would have to be researched to see how and if modeling whisker flexibility could be done.

To improve the inaccuracy in calculating bridged whisker resistance, vectors can be looked at as a potential solution [47]. The length of each whisker can be modeled as a vector, with one end starting from zero, and incrementing by a constant value until reaching the length set by its distribution input parameters. When bridged, the two or more incremental values that have made contact with a conductor can be recorded and used to establish a distance, thus representing the true resistance length of the whisker for calculation.

A simple solution exists when looking to choose whether or not the traces of the circuit board are to be labeled conductive. If the traces are covered by solder mask in real life, simply

remove the covered traces when preparing to export from Altium. Depending on the design this could be complicated, though it would allow for an immediate solution. One option to simplify this process is to use the "Un-Route" tool built into Altium [48]. This removes all of the traces from the board entirely and leaves the components. If this tool doesn't work, the traces can also be manually deleted. For long-term development, a different treatment process on the boards or software would be necessary. Regardless, if the top layer of the board could be rendered properly with its solder mask, this would prevent any of the unnecessary work in removing the traces entirely. Removing these traces also fixes the issue in the overlapping collider phenomenon.

For the shock problem, this solution is not guaranteed, but a different process of shocking the board and/or applying any necessary features to the whiskers could be researched. One potential method involves using a change in time value, with no sinusoidal component. Though in one source it is explained from a 2D stance, a further development area could apply this concept to a 3D stance [49].

The component "grouping" problem also has a few potential suggestions for future development. During development of the simulation, the team looked at .fbx files. A modeling/animation software called Blender allowed circuit board CAD designed in Altium to be exported from Altium and converted into a .fbx file. This allowed for each individual part of a component to register as its own object in Unity. However, complications arose when the necessary treatment was applied to the board to get the collisions to register properly. Unity applied oddly shaped colliders to the board when it was imported into the simulation. For simplification, the team ended up going with .obj files, which led to the previously mentioned "grouping". More research and work are suggested to try the. fbx method instead to see if it is possible to use this simulation with this file type.

In terms of material selection, the .obj export feature doesn't have an explicit solution. This software, suggested to us by our sponsor and team experts, does not give much obvious detail on the materials present on the board. Different PCB design software, processes, export options, or file types would have to be looked at to see if and how the conductor registration process could be simplified. If names were given by the files when imported into Unity, the material input feature would be simplified. Further research on PCB design software or other methods of handling these file types is necessary to make the best use of the material input feature.

For the incorrect component display problem, a short-term solution could be tested. This solution requires swapping out the troublesome component for another with similar makeup that is constructed correctly. For long-term solutions, a suggested research area might include the action of flipping the normals [50]. These normals, which are the directions "perpendicular to a mesh surface", must face the same way, "otherwise 'holes' might appear" in certain game objects [51].

CHAPTER 7
CONCLUSION

The completion of the 3D simulation in Unity successfully addressed the desired design constraints. Whiskers are generated as detached and airborne, modeled as rigid cylinders using either a normal or lognormal distribution, and can be dropped onto a user-imported CCA. The circuit board can encounter varying external forces such as gravitational acceleration, shock, and vibration. Additionally, Monte Carlo techniques were employed to simulate multiple iterations based on user-customized inputs to assess conductor bridging frequency via a heatmap, as well as bridging probabilities and bridged whisker frequencies. Results such as these allow the user to assess the bridging risks present in their CCA and make any necessary modifications. These design constraints were met using C# as the only scripting language. Upon integrating these features, each formulated design element encountered various boundary tests to ensure the simulation performed as expected. While all tests managed to pass, it is important to note that the simulation makes many assumptions to achieve its results. Although the team's simulation attempts to account for various real-world environmental factors, not all could be accurately represented. Users should be aware that the simulation does not accurately represent true whisker physics or external forces and may misidentify conductors if CCAs are incorrectly imported.

The team accomplished the simulation's construction within the timeline outlined in the Gantt Chart, located in Appendix C. Due to the limited timeframe, the errors encountered in the final version could not be resolved. For future development, the team recommends researching methods of applying flexibility and cross-sectional variance to whiskers, as well as improving shock to be modeled as a half-sine pulse. It would also be beneficial to explore other CCA design software and file types for board importing to resolve issues in net colliders, component grouping, and conductive material identification, in the case that these cannot be fixed in Altium.

APPENDIX A

Programmatic Questions

1. Are there any professional and ethical responsibilities that came up in your project (if any), and what your team did to address them and why.

   For the project, one major professional responsibility was ensuring that the team delivered a product that was accurate, easy to use, and applicable to current issues. The simulation has many practical applications in a society that heavily relies upon electronics. It is important that the user is able to accurately gauge the risk of a metal whisker causing a bridge, as a bridge could have potentially disastrous effects. One major example of this would be the Galaxy VII satellite, as bridged metal whiskers caused both of the satellite's onboard processors to fail. Because of events like this, it was important that the team delivered a simulation that could accurately warn the user of the possibility of a bridge occurring.

2. What are the economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability constraints related to your design, and how your team chose to address them.

   As this project was entirely based on software and code development, there were no major constraints related to some of the factors listed. This project was strictly software-based, so many constraints were minimal or not specified. The project was developed at no cost by using free, open-source tools and software. It was also developed with the idea in mind that this program would be used to make electronic systems safer by helping to reduce failure due to whisker bridging. Overall, the simulation provides the users with a useful tool that was built with a focus on sustainability, reliability, and providing valuable insights to all.

3. Lifelong learning is important. What are the new areas and topics that your team had to learn in order to complete your project? What methods did you use to become educated on the things you did not know before?

   The primary area that the team had to learn was C# code. This is because the simulation required C# code in Microsoft Visual Studio to be able to handle various inputs and generate calculated outputs to account for any user variation to the simulation environment. Additionally, the Unity Game Engine was new for the team, and required extensive background knowledge to be able to traverse the editor and implement various design elements. To overcome these obstacles and become educated with these unfamiliar tools, the team's technical advisor, Dr. George Flowers, introduced the team to a graduate student, Jake Botello, who had a relevant skillset unique to the purposes of this project. Jake supported the team in both Unity modification and its integration with C#. Additionally, the team used online resources for tutorials and troubleshooting advice for successful integration of design elements.

4. What are the contemporary issues related to your project and design?

   Overall, a version of the team's design, with improvements, could have a benefit to a circuit card designer. As discussed in the early stage of the report, whiskers in general can and have posed a problem for different design applications. Having a simulation in general allows for the user to get at least a semblance of what their design might behave like in real life. Depending on the quality of the results, the user could use the results to make the necessary adjustments before committing the time and financials to make their board.

APPENDIX B

3D Simulation User Manual

AUBURN UNIVERSITY

Samuel Ginn College of Engineering

3D Modeling of Detached Metal Whiskers User Operations Manual

Version 1.16

Mech 4250 – Group 8

7/16/2024

Table of Contents

Table of Figures

Introduction

This manual contains information regarding the 3D Modeling of Detached Metal Whiskers Simulation, created by a team of Senior Mechanical Engineering students at Auburn University. This program is designed to utilize the Unity Game and Physics Engine to run simulations that analyze the odds of whisker bridging across circuit boards in various environments and settings. It is intended for use in reliability testing and predictive analysis in the field of electronics, offering a user-friendly interface to assess and mitigate the risks associated with whiskers on and around circuit boards.

The program allows users to customize their entire simulation, providing control over whisker dimensions and properties, spawning locations, external forces, and circuit board analysis, among other features. It aims to give users a realistic simulation of their own boards, which can be easily imported into the program. With highly customizable parameters, users can tailor any simulation to specific, real-world scenarios.

This user manual is designed to be a comprehensive guide to effectively utilizing the 3D Modeling of Detached Metal Whiskers Simulation. By following the instructions and insights provided, you will be able to maximize the program's potential, making informed decisions to enhance the reliability of your electronic designs. If this is the first time accessing and learning how to utilize this program, continue on to: **Getting Started.** If the program is already functioning and a faster, more condensed version of the instructions are needed, continue on to **Quick Overview.**

Getting Started

Section 1: GitHub and Repository Access.

To gain access to the simulation, you will need to obtain it through GitHub. GitHub is a powerful version control and software management tool that enables developers to collaborate efficiently and manage their code effectively. By using GitHub, you can easily access the latest version of the simulation, ensuring that you are always up to date with the most recent changes and improvements. GitHub can also facilitate easy updates, allowing users to synchronize their local copies with the latest changes made by developers. This process eliminates the need for extensive redownloading. Additionally, GitHub's version control system tracks all modifications made to the code, allowing developers to provide a comprehensive history of changes to the users as updates come out. This makes it easy to revert to previous versions if needed and understand the evolution of the project over time. The utilization of GitHub for this simulation was vital to allow users to easily access and manage the program files, along with staying updated as future teams develop the program further. The following sections explain how to gain access to this program.

There are two main methods that can be used to download and gain access to this simulation. Both methods require a GitHub account. If you do not have an account already, it is vital to create one using the following link and instructions listed on the website:

https://github.com/

Once an account has been made, you may follow one of the two methods to download or gain access to the repository.

Method 1: Using GitHub Desktop (Recommended)

a. If you do not have GitHub Desktop already, refer to the following link to download:

https://github.com/apps/desktop

b. Once downloaded, open the app and log in.

c. From there, look to the top left of the app for the File tab. Click this, and find the option: Clone Repository

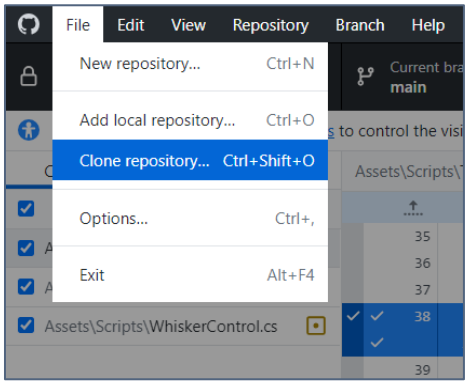


Figure 1: Cloning Repository

d. This will open a new screen. Click on the tab titled URL, and enter in the following URL to the input field:

https://github.com/connorm0088/3DStuff.git

e. Define a directory for the repository to be cloned into. Then, click Clone.

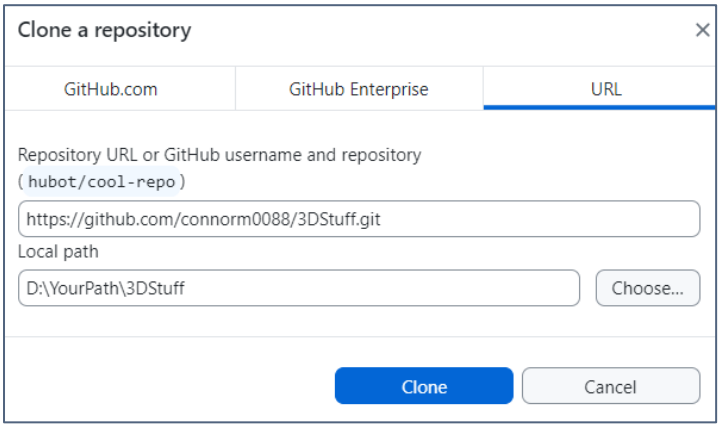


Figure 2: GitHub URL

The repository containing the simulation is now saved to your device, and you may skip to Unity Startup

Method 2: GitHub Website:

This method is more prone to issues, as it revolves around downloading the files instead of cloning them directly onto your device, which takes up more space and is much slower. This method is not recommended but can be used if GitHub Desktop is not accessible.

a. Assuming a GitHub account has been made, navigate to the following link to access the online version of the simulation repository:

https://github.com/connorm0088/3DStuff

b. Click the green Code button and click Download Zip. This will begin the installation of the program files.

c. Once fully downloaded, extract the files to a directory of your choice on your device.

The repository containing the simulation is now saved to your device, and you may continue on to Unity Startup.

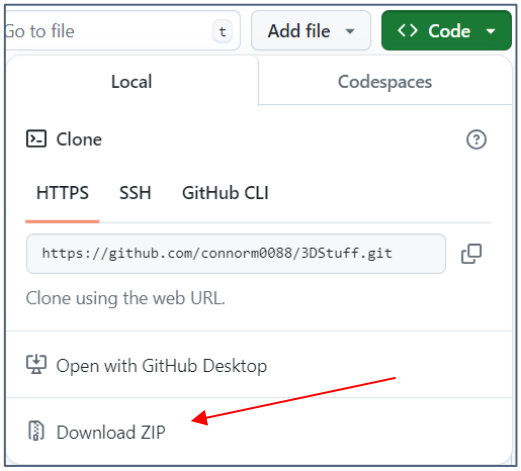


Figure 3: GitHub Repository Webpage

Unity Startup

Section 1: Downloading Unity.

As stated previously, this program runs off of the Unity Game Engine, so it is vital to have the Unity Editor and Unity Hub installed for this program to work as intended. The both the Editor and the Hub can be downloaded by following the official tutorial found on the link below. This program was developed in version 2022.3 of the Editor. It is highly recommended to get this same version for running the simulation as other versions might have different features and layouts, making it more complicated to follow the instructions.

Unity Hub and Editor Download

Section 2: Unity Hub

Once completed, you will have access to the Unity Editor and the Unity Hub. The Hub is a Unity program used to control and manage all of the projects users may have saved. This will be the main way to access and run the simulation.

To access the simulation, the repository saved in **Getting Started, Section 1** must be added to your Unity Hub. Refer to the following instructions to complete this process:

a. Open Unity Hub.

b. In the Projects tab, click Add. This will bring up your file explorer. *The image for your reference already has projects listed, however, your Unity Hub will most likely be empty.

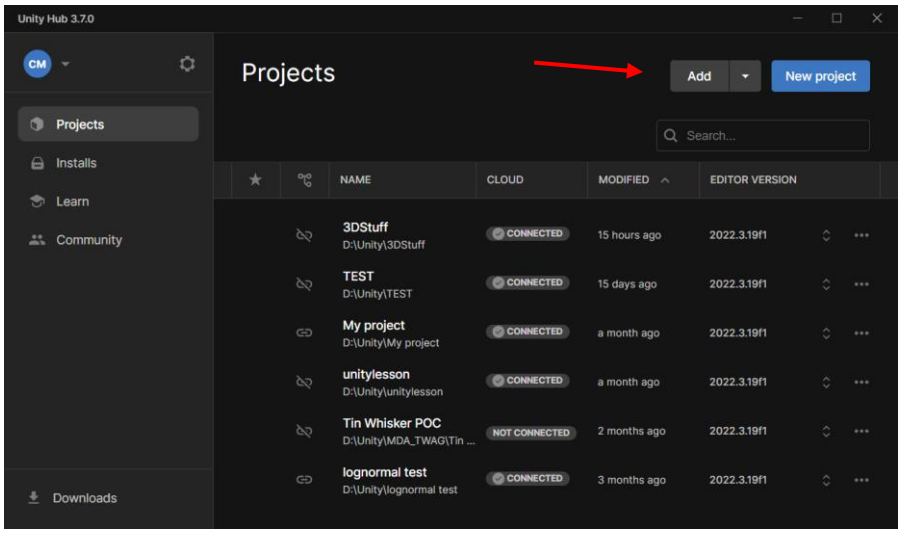


Figure 4: Unity Hub Interface

c. Navigate to the directory where the repository was saved. Click the repository and click Open.

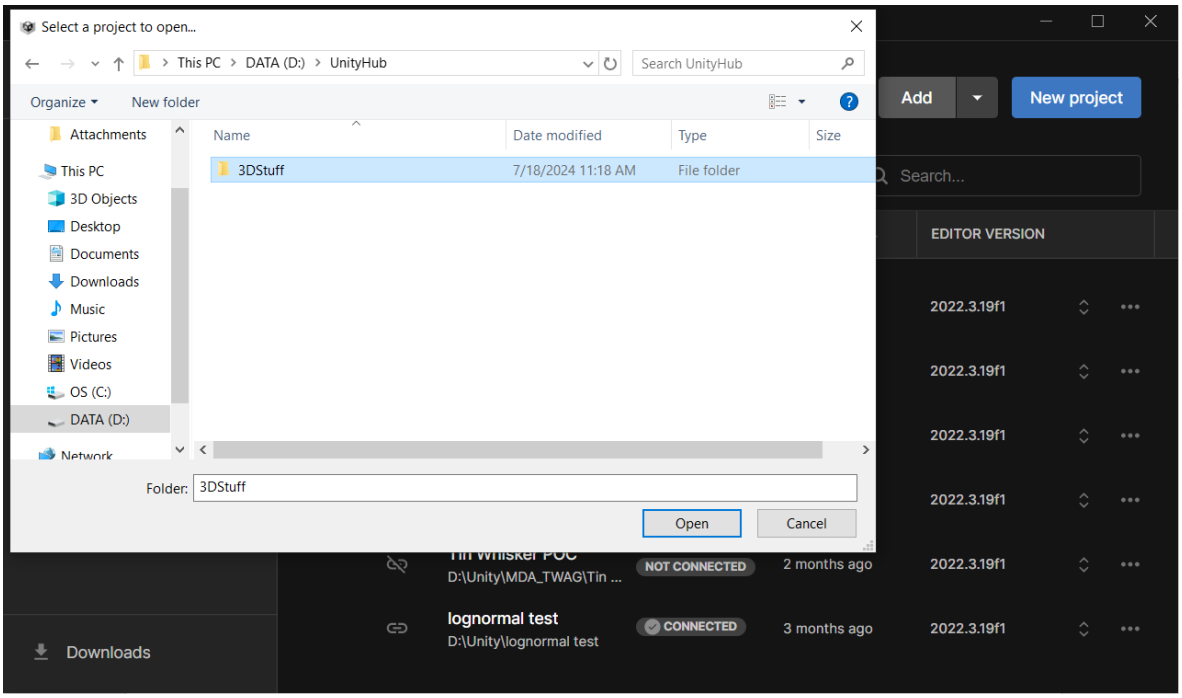


Figure 5: Selecting the Repository

The project should now be available in the Unity Hub. You may move to Section 3: Unity Editor. *Note, repository name may be different from the one shown in Figure 5.

Section 3: Unity Editor

Now that the simulation has been cloned from GitHub and added to your Unity Hub, you may now open the simulation. Clicking on the project will open it in the Editor. Please allow a few moments for this to complete. When the project is opened, you should see Unity's interface in the Editor. There are four main sections of the Editor: Hierarchy, Scene/Game window, Inspector, and the Project/Console Tabs. If this is the first time the Editor is being opened, it should be blank with nothing in the main sections. As a brief overview, the following table describes what each section handles when using the Editor:

Editor Layout

Hierarchy	On the left side of the Editor, the hierarchy lists all the Game Objects in the current scene in a hierarchical order.
Scene/Game Window	Scene: Interactive view into the simulation where objects can be manipulated. Game: Preview of sim. before being ran.
Inspector	On the Right side of the Editor, it is filled whenever a Game Object is selected, and provides important information about the object.
Project/Console Tabs	At the bottom left side of the Editor, these tabs contain project folders and any error messages being printed from the console

It is not necessary to be fully familiar with the Editor and its features to run this program. However, if more information is desired, the official Unity website has countless tutorials and help pages to answer any questions. Figure 6 shows a fresh Editor when opened:

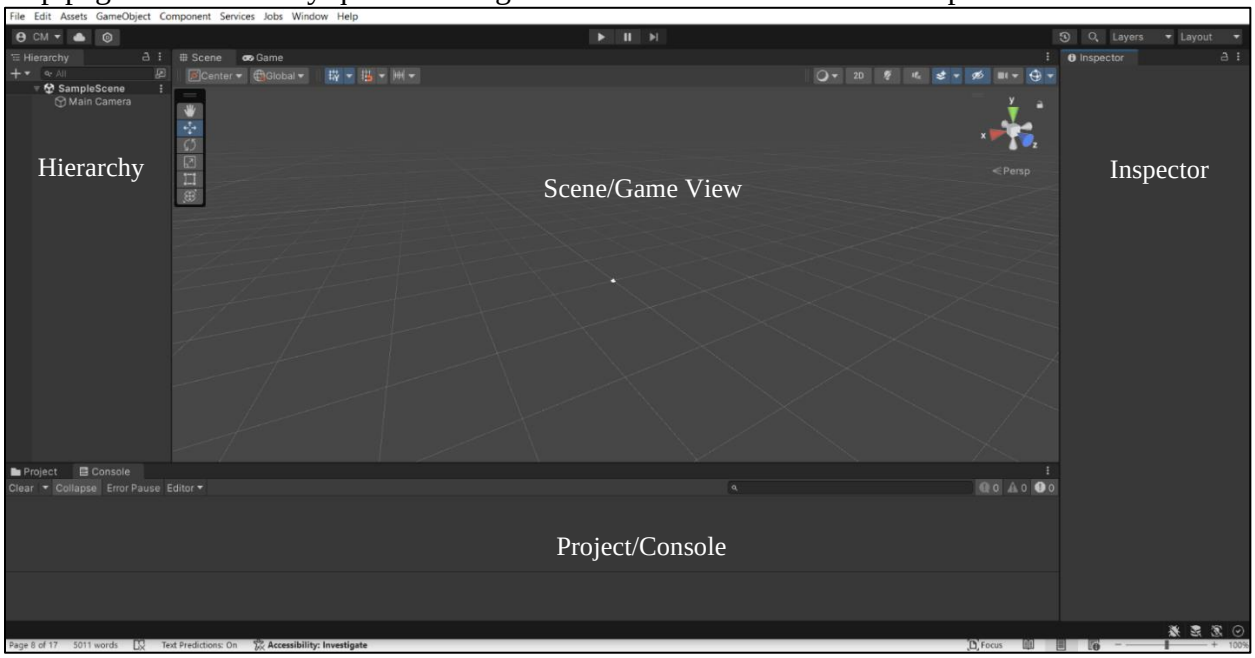


Figure 6: Empty Unity Editor

Section 4: Importing the Simulation

Once in the Editor, you may now complete the process of accessing the simulation. Refer to the following instructions to finalize this process:

a. To fully import the simulation into the fresh Unity Editor, click on Project in the Project/Console Tabs. On the left side of the tab, there should be a folder icon labeled Assets. Click the dropdown next to this folder.

b. Once opened, there will be a file labeled Scenes. Click into this folder.

c. Once inside, there will be one object: a Unity logo with the name SampleScene. Double click this object to fully import the Simulation into your Editor.

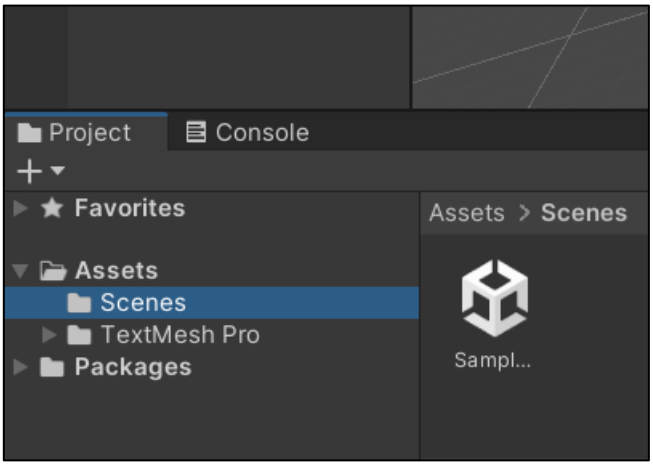


Figure 7: Sample Scene

Once these steps have been completed, the simulation is now being hosted by the Editor. To verify this was done correctly, refer to Figure 8 to compare your Editor. These steps will not have to be repeated as the simulation is now saved to your Unity Hub and can be reopened at any time.

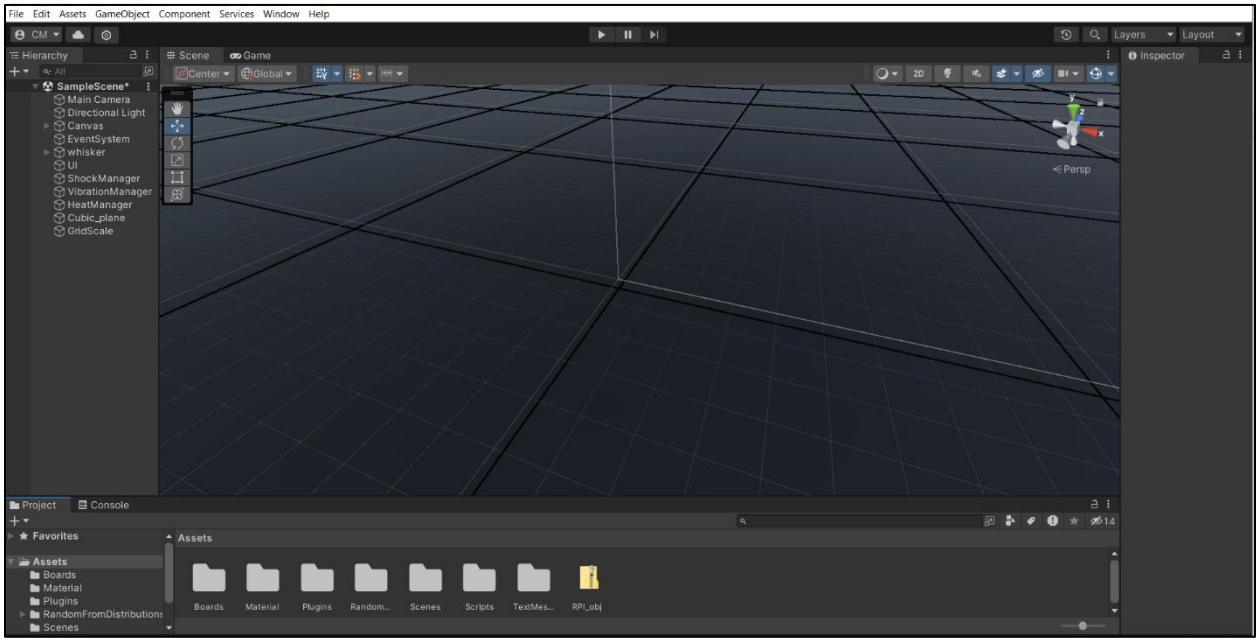


Figure 8: Simulation Editor Screen

You may now proceed to **Setting up your Simulation.**

Setting up your Simulation:

Section 1: Loading in your Desired Board

To get the most accurate results for your desired board, it will need to be imported into Unity as a .obj file. Other file types can be imported into Unity successfully, but some manipulation of the components may be required to get them to function properly. Many programs such as Altium, Blendr, Autodesk Eagle, and others have the capabilities to produce .obj files while maintaining the important contours and components of your circuit board.

a. Once you have your desired PCB in a compatible file type, open the folder containing the 3D object. Next you will need to access the Boards folder that came with the simulation. This can be done through the Unity Editor, under the Project tab. Click to open the Assets folder, and find a new folder titled Boards, then click into the folder. Once here, drag and drop your board into the folder. The board will now be present in the Project tab. ** Disregard preexisting boards in Figure 10. These will not be present in your project.

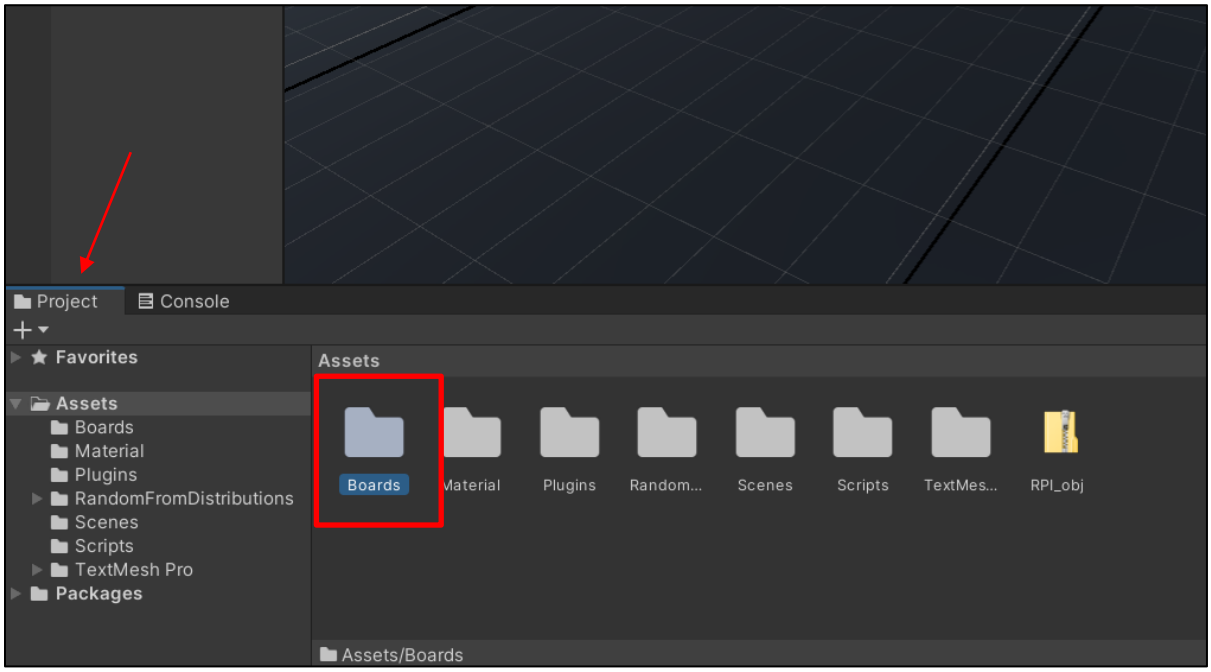


Figure 9: Accessing Board Folder

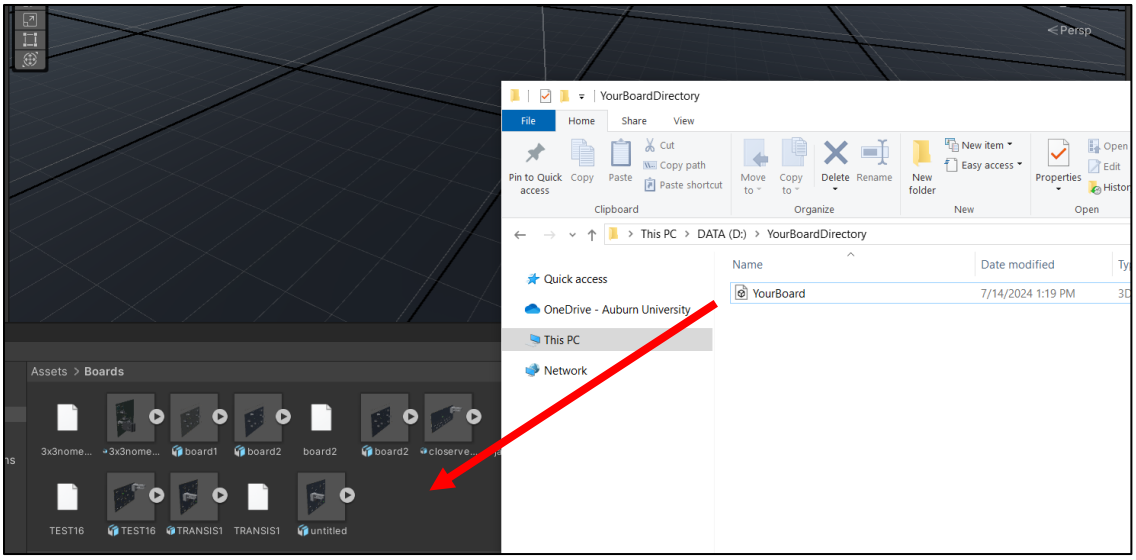


Figure 10: Importing your PCB

b. Click on your board and drag it into the Scene View. It does not matter where it is placed in the Scene, as through the following steps and the background code, it will be placed for you. Figure 11 demonstrates this task. *Disregard excess boards shown in figure.

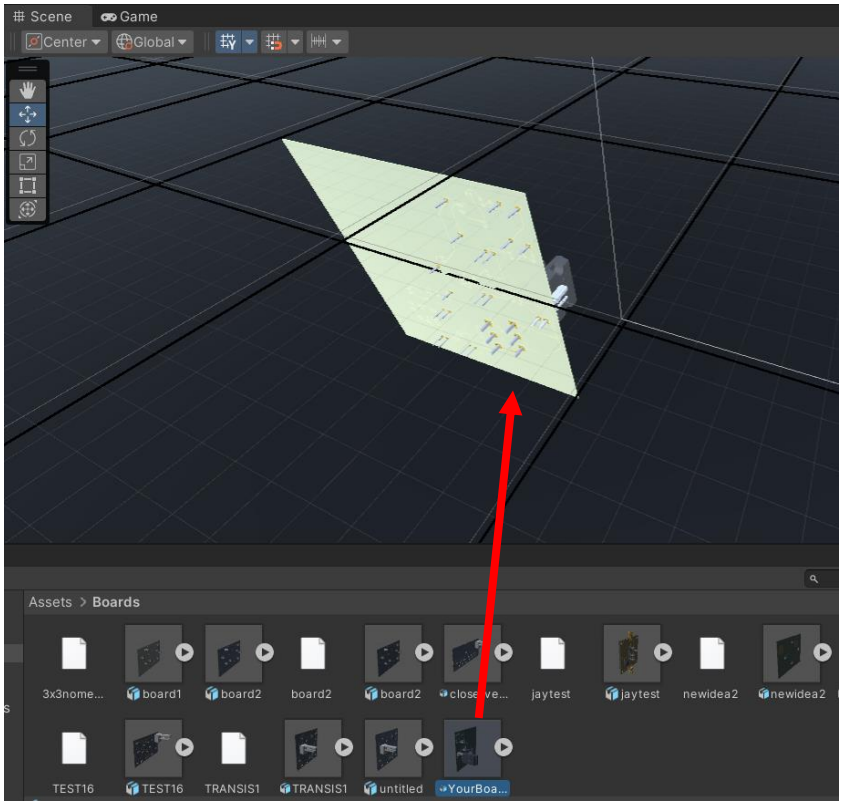


Figure 11: Dragging in the PCB

c. Correct the Orientation of your board in the case it does not import correctly. This can be done by selecting the board, and then looking in the Inspector tab. There should be a subsection

called Transform, Rotation, where you can edit the rotation of the board to orient it properly. Example: typing in 90 under the X axis rotation will rotate the board 90 degrees in the positive X direction. Once this is done, the board should now be ready for the setup.

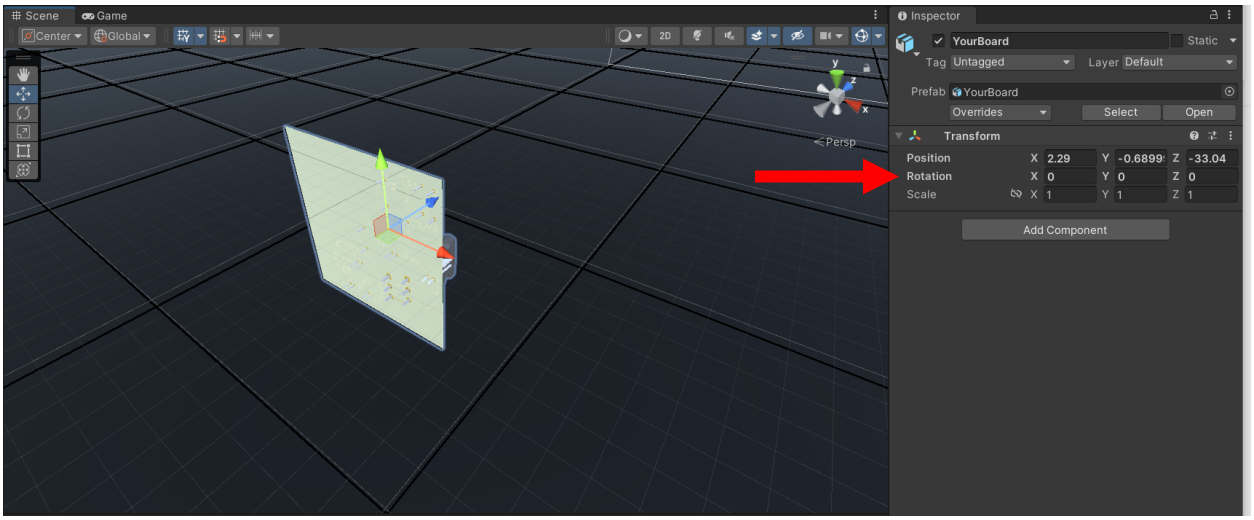


Figure 12: Orienting your Board

Section 2: Preparing your board for Simulation

A few easy steps are required in order for the simulation to run as needed. These involve using the unity editor to attach certain scripts/objects to the board in order to effectively run the simulation.

a. If the previous section was completed properly, the board should now be in the Scene. Select your board and look to the Inspector. There will be a button called Add Component. Clicking this will bring up a search bar. Type in the bar: TriggerControl, and an icon that looks like a paper with a green pound symbol with the matching name should show up. Click this item. The script will be added as a component, and a new field will be added to the Inspector for your board.

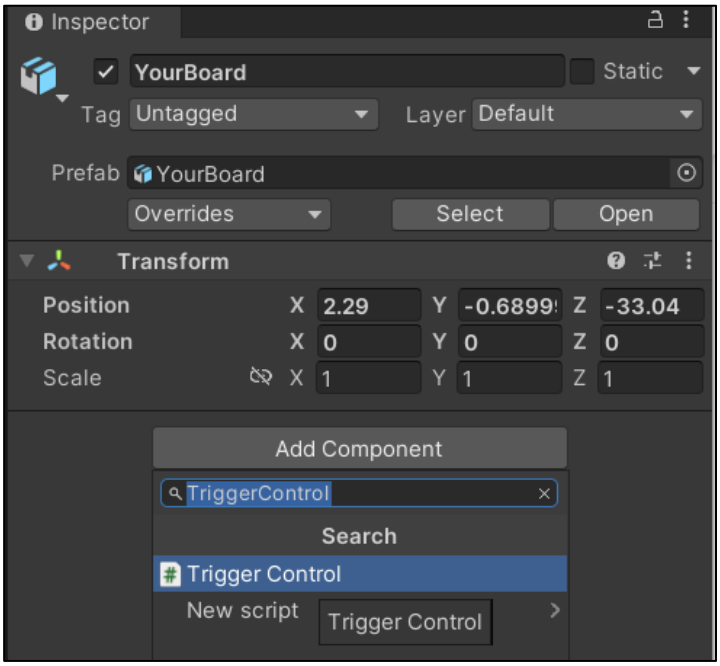


Figure 13: Adding Components

b. Now, you should see a few new fields in the Inspector under the section Trigger Control. There is an empty field titled Trigger Material. Click the small circle to the right of this field, which will bring up a text input field. Type “TriggerMaterial” and click the item that comes up.

c. Finally, directly under the Trigger Material field should be one titled Material Input. Similar to the last step, click the small circle to the right of the field, but instead type type “MaterialInput”, and click the item that comes up. Leave all other fields in this section untouched. Your board is now ready to be run through the simulation. Figure 14 shows what the fields should look like once completed properly.

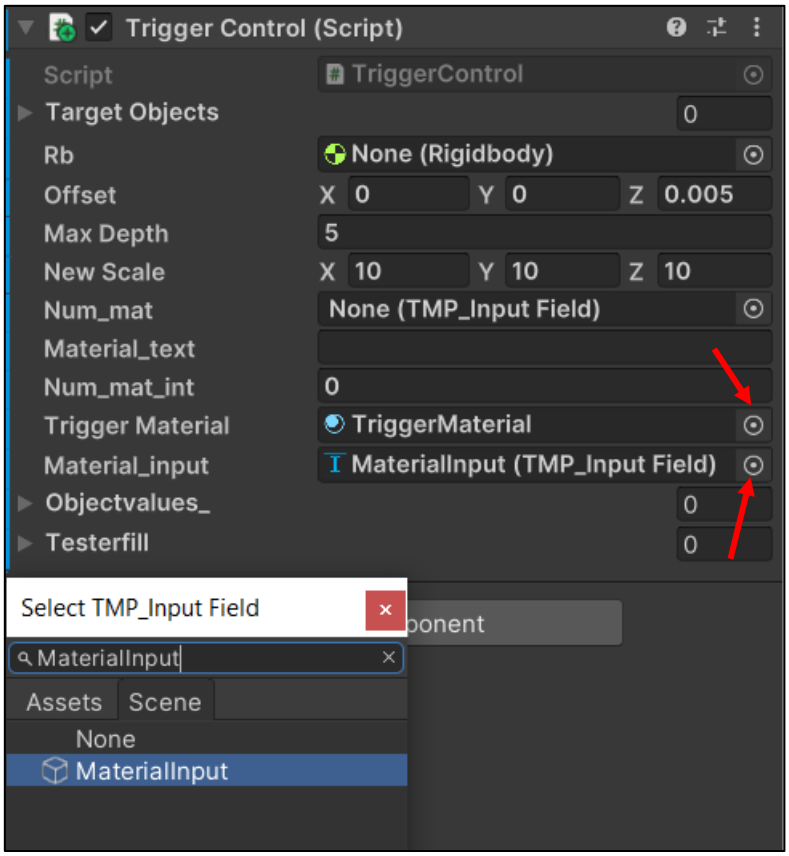


Figure ~~38:~~14: Finalizing the Board

The PCB of choice can now be correctly analyzed in the simulation. Keep in mind that these steps will need to be repeated as more boards are imported into the simulation but will not need to be repeated for ones already set up. If there are multiple boards in the Scene and only one is required for testing, you can turn off the others by selecting them and looking at the top left of the Inspector window. There, you will see a small check mark indicating that the board is 'on.' Clicking this check mark will turn the board off, allowing Unity to retain the board's setup but remove it from the Scene until it is needed again.

Running the Simulation (Walkthrough):

Section 1: Camera Controls

Now that the boards are set up, the simulation can be run. To do this, look to the top of your Unity Editor. Click the Play Button to start the program.

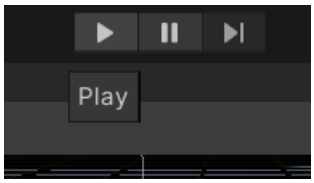


Figure 15: Play Button

a. When in Play Mode, users can navigate the camera around the simulation to get the best views of their PCB/whisker bridges. To rotate the camera, hold the left mouse button and move the mouse in any direction. Additionally, use the W, A, S, and D keys to move the camera forward, left, right, and backward from the direction the camera is facing. The E, and Q, keys can be used to move the camera vertically up or down, and clicking the right mouse button will reset the camera orientation to level with the board. On the bottom left of the Interface, there

is a section titled ‘Camera’, which has three buttons that when pressed, set the camera to preset conditions. Standard is a front facing ariel view, Top Down is a birds eye view from directly over the board, and Side is a side facing view from the left side of the PCB.

Figure 16 shows what the simulation would look like with a PCB imported in. There are four main areas of the interface that will be covered in the following sections in detail. As a brief overview: each section handles a different part of the simulation and allows users to either give inputs to fully customize the simulation and save any data necessary or run and manage the simulation. Each simulation comes with a scale grid, which are the black lines laid across the board and the entire Scene. These grids are all 1x1 cm squares.

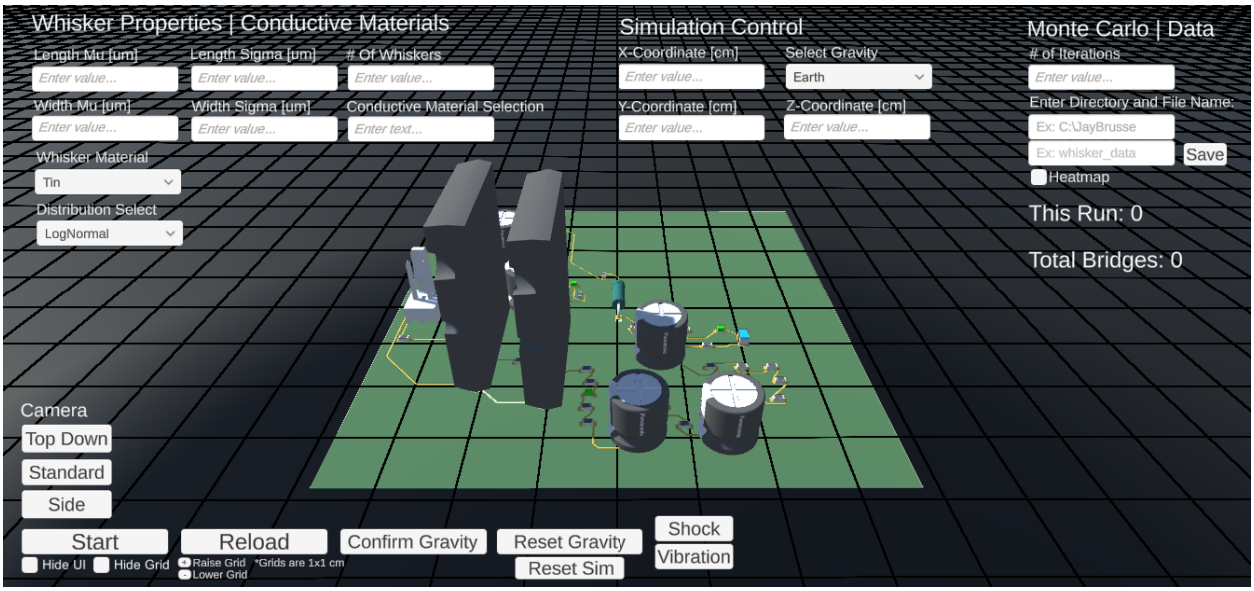


Figure 16: Simulation Interface

Section 2: Whisker Properties | Conductive Materials

This is the section of the Interface that handles all of the user inputs required to generate whiskers into the Scene, along with the input field that allows users to tell the program what materials on their board are to be seen as conductive. Below is a table giving a description of what each part of the Interface does:

Whisker Properties Inputs	
Length Mu [µm]	Takes in the value and uses it as the location parameter or the mean value for the Lengths of the whiskers depending on if the distribution is Lognormal or Normal.
Length Sigma [µm]	Takes in the value and uses it as the scale parameter or the standard deviation for the Lengths of the whiskers depending on if the distribution is Lognormal or Normal.
Width Mu [µm]	Takes in the value and uses it as the location parameter or the mean value for the diameters of the whiskers depending on if the distribution is Lognormal or Normal.
Width Sigma [µm]	Takes in the value and uses it as the scale parameter or the standard deviation for the diameters of the whiskers depending on if the distribution is Lognormal or Normal.
# of Whiskers	Sets how many whiskers are to be generated.
Whisker Material	A dropdown list that allows users to select from Tin, Zinc, or Cadmium, changing the base properties of the whiskers.
Distribution Selection	A dropdown list that sets what distribution type is to be used for the generation of the whiskers. Defaults to Lognormal.
Conductive Material Selection*	A text field that when material names are entered in, the program searches the components of the board for the specific material listed, and if it finds a match, it labels that object as Conductive.

a. You can now begin to enter in values for your Mu and Sigma inputs in micrometers and choose your number of whiskers, along with selecting any desired options from the dropdown lists. Verify that these parameters are in line with the distribution being, as the whiskers may not spawn in or be visible depending on the size of the inputs and the distribution being selected. The whisker material may now be selected from the dropdown as well.

Figure 17: Example Inputs

b. Finally, the conductive material can be set. This will be done by entering the name of the material into the input field, and then clearing the field before entering in more conductive materials if there are multiple. This can handle any number of conductive materials, but once the conductive materials are set, they cannot be undone except by exiting Play mode, and refreshing the simulation.

*NOTE: Many outside programs used for creating models of circuit boards do not handle the materials of the components very well, regardless of which program is being used. Often times, instead of a material being labeled correctly (e.g. copper), it will be labeled as mat_12, or a variation of that format, which will then be how it appears in Unity. It is important to scan through your board after importing and see what material names have been assigned to properly use this program. These could change depending on the program the board gets exported from. To view these materials, open the Boards folder in the Project tab, under Assets. Click the small arrow next to your board. This will open up a list of all the components on the board and any materials that were imported alongside.

Figure 18: Materials Imported from Altium

Section 3: Simulation Control

This section of the Interface handles where the whiskers will spawn in from, along with what gravity scale will be applied upon confirming the gravity later in the walkthrough. Below is a table showing what each input field handles.

Simulation Control Inputs

X – Coordinate [cm]	Set the bounds in the X direction where the whiskers can spawn in from.
Y – Coordinate [cm]	Set the bounds in the Y direction where the whiskers can spawn in from.
Z – Coordinate [cm]	Set the bounds in the Z direction where the whiskers can spawn in from.
Select Gravity	A dropdown list that allows users to select what gravity scale is being applied with a selection of Earth, Moon, or Mars.

a. The spawning locations can now be set using the coordinate system. This system works by effectively creating a cube that the whiskers can spawn into. When a value is entered into the coordinate field, it sets the boundaries for spawning objects in both directions. For example, if you type 3 into the coordinate field for the x direction, the objects will be able to spawn anywhere from -3 to +3 on the x-axis, providing a total span of 6 centimeters. Similarly, the same applies to the y and z directions if values are entered for those fields. This allows PCBs of any size to be effectively covered by whiskers. Figure 19 demonstrates this:

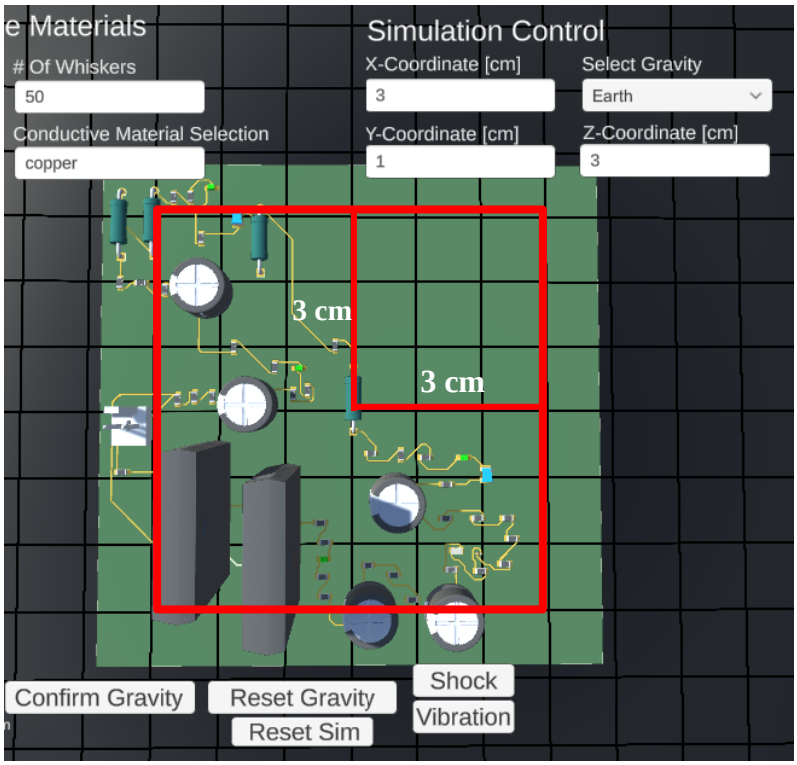


Figure 19: Whisker Spawning Location

It can be seen from the red box where the whiskers would be able to spawn in from given an input of 3 cm for the X and Z axis, as when the simulation is run, the board is instantly placed at (0, 0, 0) in the Scene.

b. Next, the gravity scale can be selected. Clicking the dropdown will show the list of options: Earth, Moon, or Mars. Clicking one of the options will set the gravitational acceleration to the respective value.

Section 4: Monte Carlo | Data Storage

Once the inputs covered in sections 2-3 have been filled in, the final inputs can be set. This section allows the user to run the simulation through multiple iterations, automatically, along with providing a directory and file name for the data to be stored in order to save information from the simulation. This section also has the toggle button used to turn on the Heatmap, which will be discussed later on in the walkthrough. The table listed below describes what this section of the interface provides:

Monte Carlo | Data Storage Inputs

# of Iterations	Tells the simulation how many iterations to run before coming to a stop.
Directory	Input field for users to place their specific/desired location for their data on their device.
File Name	Input field for users to set the file name of their .csv sheet, which will contain the simulation data.
Save Button	Saves the Directory and File Name inputs into the code.
Heatmap Toggle	Toggles the Heatmap on or off.
Bridges for Current Run	Displays number of bridges detected for the current iteration.
Total Bridges	Displays the total number of bridges detected across all iterations.

a. The number of iterations for the simulation to run through can be set. It is recommended to enter in “1” for this field until the whiskers have their desired dimensions and spawning area.

b. Enter in a directory on your personal device and a file name for the csv file to give the simulation a location to store the data into. Press Save once ready. If an incorrect directory or file name is entered, reenter the correct name and press Save again.

c. Optionally, click the Heatmap Toggle to turn on the Heatmap. The components containing the conductive materials you have given in **Section 2-b** will be highlighted in green, and a scale of the gradient will appear on the screen, ranging from 0-5+ whiskers, moving from light green to dark red. More details about this feature can be found **in Running the Simulation: Section 6.**

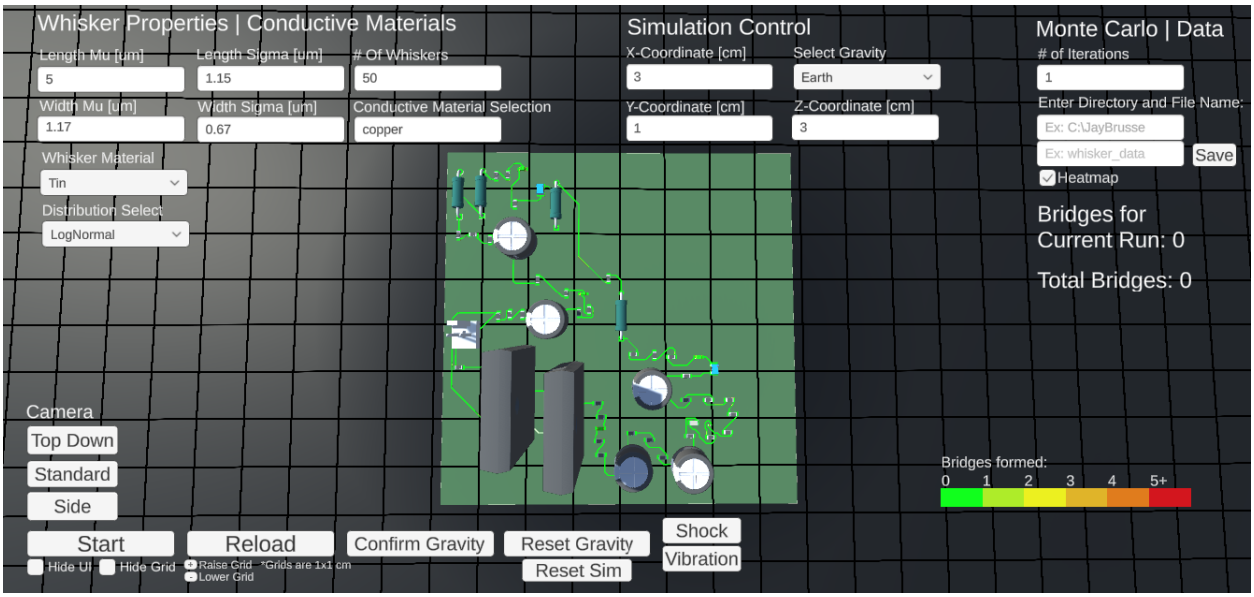


Figure 20: Heatmap Toggled

Section 5: Simulation Management

This section of the Interface allows users to spawn in whiskers, reload them to better fit to the PCB, begin and stop the simulation, along with adding external forces onto the board. The external forces are covered in detail in **Section 7: External Forces**. Below is the final table describing what each Interface object handles in this area:

Simulation Management Controls	
Start	Initially spawns in whiskers based off all of the previously described inputs.
Reload	Allows users to load fresh whiskers into the Scene after making a change to either their dimensions, number in the Scene, or coordinate location over the board.
Confirm Gravity	Confirms the gravity and officially starts the simulation. Pressing this will run the simulation through all the iterations desired.
Reset Gravity	Turns off the gravity, and the iteration counter, which allows the whiskers or data storage to be changed to different parameters.
Reset Sim	Clears and stops the iteration counter, and clears bridges formed - total and per run.
Shock	Opens the inputs for the external force: Mechanical shock or impact.
Vibration	Opens the inputs for the external force: Physical vibration.
Hide UI	Toggle that hides the user inputs.
Hide Grid/Raise/Lower Grid	Series of a toggle and buttons that allow the user to turn off or on the scale grid, along with raise or lower it to better fit their PCB.

a. Once the previous sections and steps are completed, all inputs should be filled in and any selections from the dropdown lists should have been made. The Start button should now be pressed to begin the simulation. This will spawn in the whiskers based off the parameters chosen.

b. Next, you should verify that the whiskers are spawning in with the desired dimensions over the appropriate areas. If any parameter must be changed, pressing the Reload button will spawn new whiskers into the scene based off of the new parameters. Repeat this step until the whiskers are generating over the entire board and are the appropriate size for your simulation needs.

c. If the whiskers are correctly sized and generating over your PCB, confirm that the directory and file path are as desired, along with the number of iterations for the simulation to run through. Figure 21 demonstrates these inputs being filled.

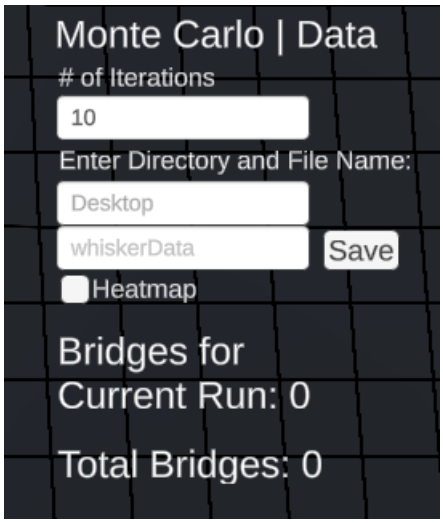


Figure 21: Data Storage Input

d. Finally, press the Confirm Gravity button to confirm all settings, and fully begin the simulation. The whiskers should now be dropping towards the board, and every 5 seconds, they will be cleared along with generating a new set with the same parameters you have entered in. The whiskers creating bridges will be highlighted red and will be tracked, while the number of bridges being made will be printed to the screen on the right. The data from the simulation will be stored into the file name you have provided. Figure 22 provides a look at what the simulation looks like while running, while Figure 23 shows a bridged whisker being highlighted and compared to a non-bridged whisker. *NOTE the whiskers used in Figure 22 were generated using a Normal Distribution with extremely exaggerated parameters for demonstration purposes.

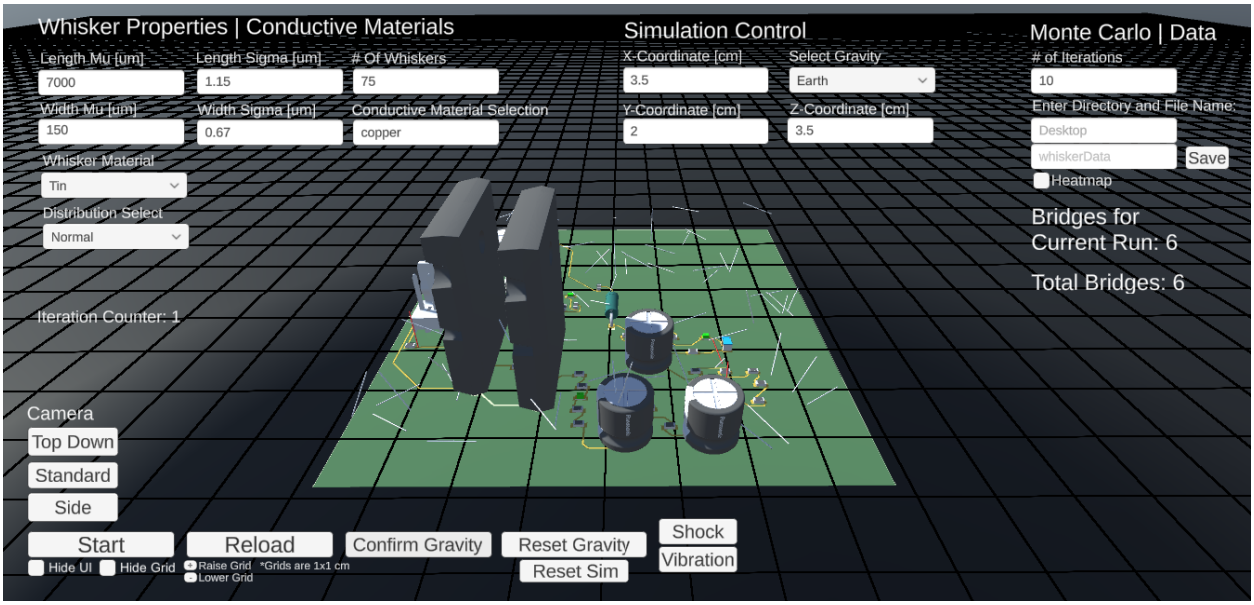


Figure 22: Simulation Running

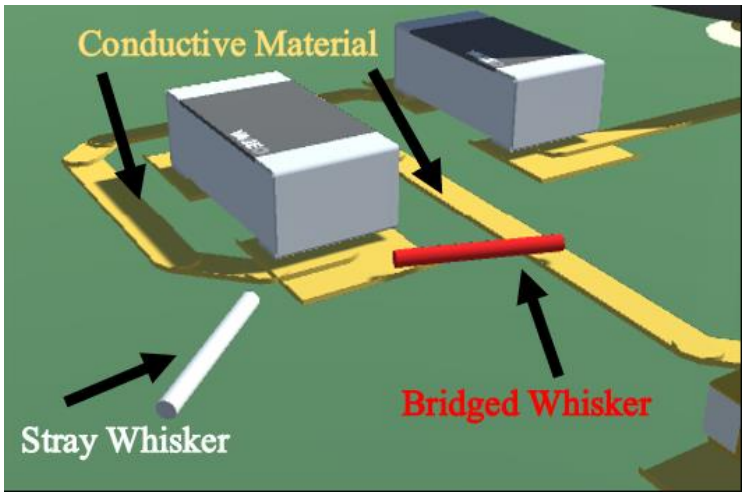


Figure 23: Bridged vs Stray Whiskers

Section 6: Heatmap

Once your simulation has been completed. A message will appear on the left side of the screen in green: "Simulation Complete!". The Heatmap toggle located in the **Monte Carlo | Data Storage** section of the Interface can be turned on if it has not been already. Post-simulation, assuming there were some bridges formed, some components will have changed colors representing how many bridges each component contributed to making. This allows a visual of what components are most likely to fail due to whisker bridging.

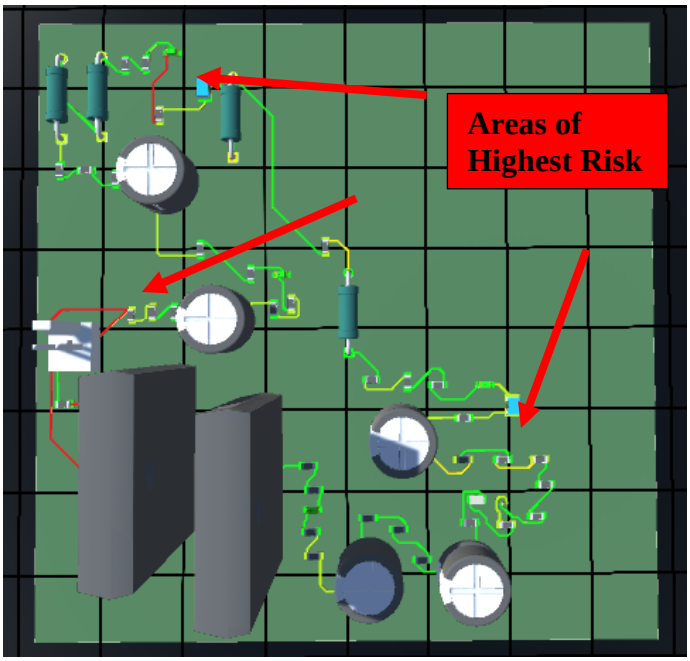


Figure 24: Heatmap – Post Simulation

Section 7: External Forces

As mentioned in Section 5, there are buttons in the user interface that open the external force input fields. To use them properly, please refer to the following steps:

Shock | Impact

a. To apply the mechanical shock or impact force to the board, press the Shock button. This will open the Shock Force Interface. There will be an input field for the value of force, a dropdown list that allows the force to be applied to any axis, a toggle, and a new Shock button underneath the dropdown.

b. **Enter in a value for the Amplitude (N) text field. This will tell the program how many Newtons of force to apply to the board. Decimal and negative values are accepted. Entering in a negative value will push the board in the opposite direction.

c. Select an axis to apply the force to through the dropdown. The options are X, Y, and Z.

d. Finally, press the Shock button on the right of the screen to apply the shock. This will apply the force to the board based off the input and direction given.

e. The toggle can then be checked. Clicking this will run the shock continuously, using the inputs provided by the user, while running multiple iterations of the simulation.

** NOTE, any shock value over 10 N could cause whiskers to be pushed dramatically in the direction of choice. It is highly recommended to operate in the 0-10 N range.

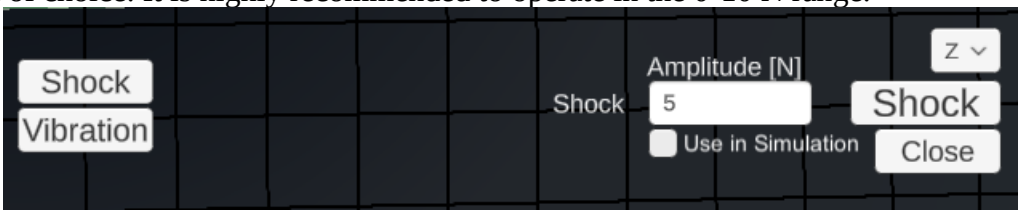


Figure 25: Sample Shock Input

Vibration

a. To apply a vibration to the board, press the Vibration button. This will open the Vibration Interface. Similarly to the shock, there will be an input field for the Amplitude of the vibration, a dropdown list that can select the axis to apply the vibration, a toggle, a new vibration button, and inputs for the Frequency and Duration of the vibration.

b. Enter in values for the Amplitude, Frequency, and Duration. A smaller amplitude (e.g. 0.2) and a larger frequency (e.g. 5-10) will provide a proper vibration, whereas a larger amplitude with a smaller frequency will provide a much smoother and wider vibration. The amplitude can be a negative value, while the Frequency and Duration cannot be.

c. Select an axis to apply the vibration on from the dropdown list.

d. Press the vibration button to apply the vibration.

e. Check the toggle to on to apply the vibration across multiple iterations when running the simulation fully.

**Set the vibration duration to 5 seconds or less before pressing Confirm Gravity. Failure to do so will result in the vibrations overlapping and moving the board out of place.

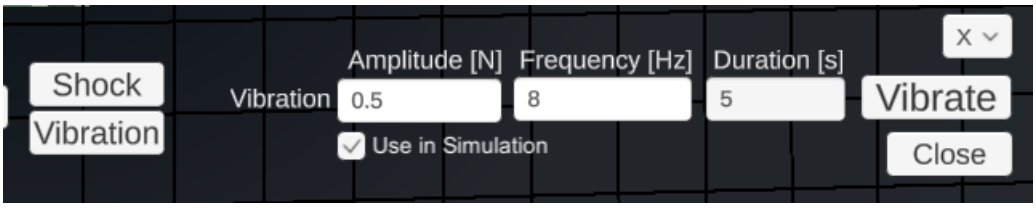


Figure 26: Sample Vibration Input

Section 8: Resetting the Simulation | Saving New Data

To reset your simulation, start a new set of iterations, or save a new set of data, you will have to reset the simulation. There are a few ways to do this. The first method involves using the Reset Sim button located near the bottom of the screen when viewing the Interface.

a. Click Reset Gravity. This will set the gravity back to 0 and prevent the iterations from running until any new changes to the inputs are made and ready, along with preventing any excess or unwanted data from being stored.

b. (Optional) Clear the Directory and File Name input fields and press Save.

c. Set the Number of Whiskers field to 0 and Press Reload to remove the whiskers from the surface of the board, preventing premature bridging detection.

d. Click the Reset Sim button. This will clear the bridges detected and the iteration counters so a new run can be completed.

e. Enter new parameters into the input fields as needed.

f. Fill in the Directory and input a new File Name to store the new data set into. Press Save. (e.g. if the first .csv file was titled ‘whisker_data’, name the next one something new like ‘vibrationOnData’ or ‘whisker2_data’)

g. Press Reload with the new or original inputs until whisker are desired size and locations.

h. The simulation should now be ready to be run again by pressing Confirm Gravity.

Alternatively, you can clear all inputs and values and fully reset the simulation by simply pressing the Play button at the top of your Unity Editor to fully stop the program, and then pressing it again to reopen it.

Data Processing

Section 1: Using WhiskerResults

After the data has been saved into your file of choice, you may navigate to the directory you have provided, and open the file. You will see that the data has been saved and formatted. Additionally, in the program files that were saved to your device previously from GitHub, there was an Excel file titled WhiskerResults. This .xlsm file is equipped with Excel macros and instructions which allows more in-depth results from the simulation to be generated and viewed.

Figure 27 shows an example of what the data might look like when stored into the .csv file you have generated. Notice that the file keeps track of every whisker that is generated, along with its dimensions, resistance, and what iteration it was generated in. The same applies to the whiskers that created a bridge. **Data values in the example are highly exaggerated.

All Whiskers				Bridged Whiskers			
Length (um)	Width (um)	Resistance (ohm)	Iteration	Length (um)	Width (um)	Resistance (ohm)	Iteration
5500.664	151.3164	0.03334108	1	5499.993	148.587	0.03457302	1
5500.777	150.6189	0.03365131	1	5500.922	150.2488	0.03381817	1
5499.958	149.7035	0.03405903	1	5501.795	150.1787	0.03385513	1
5500.992	149.6109	0.03410761	1	5498.949	150.8678	0.03352919	1
5499.831	150.6828	0.03361695	1	5499.732	149.6539	0.0340802	1
5500.442	148.7595	0.03449571	1	5502.174	149.8021	0.03402789	1
5499.725	149.9621	0.03394021	1	5499.625	149.8744	0.03397935	1
5500.098	149.9509	0.03394758	1	5499.877	151.0531	0.03345262	1
5500.473	149.5945	0.03411185	1	5499.977	149.3731	0.03420996	1
5497.817	150.4017	0.03373037	1	5499.055	150.1084	0.03386997	1
5501.496	150.4387	0.03373636	1	5500.166	149.5694	0.03412142	1
5500.687	149.5997	0.03411079	1	5499.003	150.3931	0.03374154	2
5497.783	149.8971	0.03395764	1	5498.267	150.1608	0.03384146	2
5499.492	150.496	0.0336984	1	5501.301	150.5085	0.0337039	2
5500.704	150.5976	0.03366037	1	5501.881	150.9467	0.033512	2
5499.759	150.7462	0.03358824	1	5498.669	150.372	0.03374894	2
5501.197	150.593	0.03366545	1	5500.64	149.2015	0.03429285	2
5500.395	149.9039	0.03397069	1	5502.168	150.5374	0.03369628	2
5502.018	149.9339	0.03396711	1	5500.912	149.6023	0.034111	2
5500.601	149.7112	0.0340595	1	5500.173	150.3027	0.03378933	2
5499.558	150.3568	0.03376124	1	5499.691	150.6411	0.03363475	2
5498.945	148.4898	0.03461167	1	5498.708	150.2798	0.03379063	2
5501.795	150.1787	0.03385513	1	5498.628	151.3692	0.0333055	2
5499.492	149.6872	0.03406354	1	5497.918	150.3175	0.03376881	3
5499.663	150.234	0.03381709	1	5499.193	151.0179	0.03346404	3
5500.308	149.9406	0.03395355	1	5498.81	149.7552	0.03402841	3

Figure 27: CSV File Data

To take this data and get the results properly, refer to the following instructions for guidance:

a. Open WhiskerResults.xlsm.

b. Open the file containing your simulation data.

c. Copy and paste the values from your data set to the designated location in WhiskerResults.

d. Once all of the values are set into the proper location, there will be instructions listed inside of the file that give details on how to properly analyze the data.

Copy and paste new Unity data below.							
298.625	2.33234	4.12385	1	797.613	6.43327	1.44774	1
234.075	2.40527	3.03942	1	577.434	5.68211	1.34353	1
150.199	4.48344	0.56131	1	214.628	1.82064	4.86408	1
34.504	2.90437	0.30728	1	5437.34	3.31164	37.2445	2
45.5618	2.78521	0.44121	1	409.633	3.55443	2.43566	2
160.746	6.94118	0.25063	1	332.493	2.71252	3.39468	2
19.0187	1.23628	0.93478	1	81.5415	1.63816	2.2826	2
258.642	4.47146	0.97177	1	56.5992	2.54944	0.65416	2
189.303	1.28257	8.64483	1	270.461	1.65912	7.38095	2
1531.53	1.42842	56.3865	1	402.077	2.92299	3.53522	3
62.7893	2.7378	0.62928	1	133.358	2.11385	2.24198	3
71.608	8.19879	0.08002	1	172.296	3.61279	0.99163	3
725.45	2.91612	6.40852	1	301.436	6.50395	0.53531	3
46.8027	3.40158	0.30386	1	204.567	1.97747	3.92988	3
72.6412	7.02549	0.11056	1	54.8071	1.14971	3.11476	4
1393.06	2.4075	18.0551	1	19.2166	9.14976	0.01724	4
200.653	3.41422	1.29308	1	173.018	4.32173	0.69589	4
1070.39	4.31052	4.32758	1	8.10114	3.31514	0.05537	4
652.171	4.27984	2.67466	1	161.056	1.20088	8.38957	4
577.766	2.84501	5.36224	1	41.2818	1.08942	2.61295	4
46.8904	3.46618	0.29319	1	165.95	1.68048	4.4144	4
30.3718	1.44439	1.09361	1	461.17	6.2458	0.88807	4
142.188	6.68722	0.23885	1	30.1957	1.96854	0.58536	4
129.773	5.46161	0.32682	1	153.531	2.74972	1.52539	4
74.2305	2.90219	0.66205	1	288.011	4.14752	1.25775	5
19.8655	3.833	0.10157	1	476.62	1.58571	14.2393	5
66.8773	3.76375	0.35465	1	716.574	5.38013	1.85967	5
287.827	3.75607	1.53259	1	325.767	3.15344	2.46094	5

ALT + F8 -> GenerateProb	
Overall Probability	
Individual Probability	
Iter #	Probability

Figure 28: Sample Data in WhiskerResults

e. Pressing the Alt key and F8 at the same time will open the macro list used to generate the data.

f. Selecting the GenerateAll macro will analyze the data and produce something similar to what is seen in Figure 29 and Figure 30.

ALT + F8 -> GenerateProb	
Overall Probability	
100.00%	
Individual Probability	
Iter #	Probability
1	0.60%
2	1.20%
3	1.00%
4	2.00%
5	1.80%

ALT + F8 -> GenerateFreq			
Length	Diameter	L:D Ratio	Resistance
797.613	6.433271	123.982497	1.44774
577.4344	5.682107	101.623289	1.343526
214.6283	1.82064	117.886183	4.864083
5437.337	3.31164	1641.8865	37.24451
409.6334	3.554434	115.245747	2.43566
332.4931	2.712521	122.577152	3.394678
81.54153	1.638159	49.7763221	2.282595
56.59921	2.549437	22.2006702	0.6541598
270.4611	1.659118	163.014987	7.380952
402.0768	2.922991	137.556633	3.535218
133.3579	2.113854	63.0875642	2.241976
172.2962	3.612794	47.6905686	0.991634
301.436	6.503949	46.3466119	0.535307
204.567	1.977467	103.449008	3.929879
54.80706	1.149706	47.6705001	3.114762
19.21662	9.149758	2.1002326	0.01724328
173.0178	4.321728	40.0344029	0.6958857
8.101142	3.315135	2.44368389	0.05537399
161.0557	1.200879	134.114844	8.389572
41.28175	1.089418	37.8933981	2.61295
165.95	1.680482	98.7514296	4.4144
461.1695	6.2458	73.8367383	0.8880688
30.1957	1.968538	15.3391502	0.5853553
153.5307	2.74972	55.8350305	1.525388
288.0109	4.147521	69.4416978	1.257748
476.62	1.585709	300.572173	14.23925
716.574	5.380134	133.188876	1.859674
325.7667	3.153436	103.305315	2.460938
120.336	5.481293	21.9539441	0.3008789
692.2314	4.037853	171.435513	3.189425
1992.172	5.468721	364.284812	5.004

Figure 29: Probability and Frequency Data

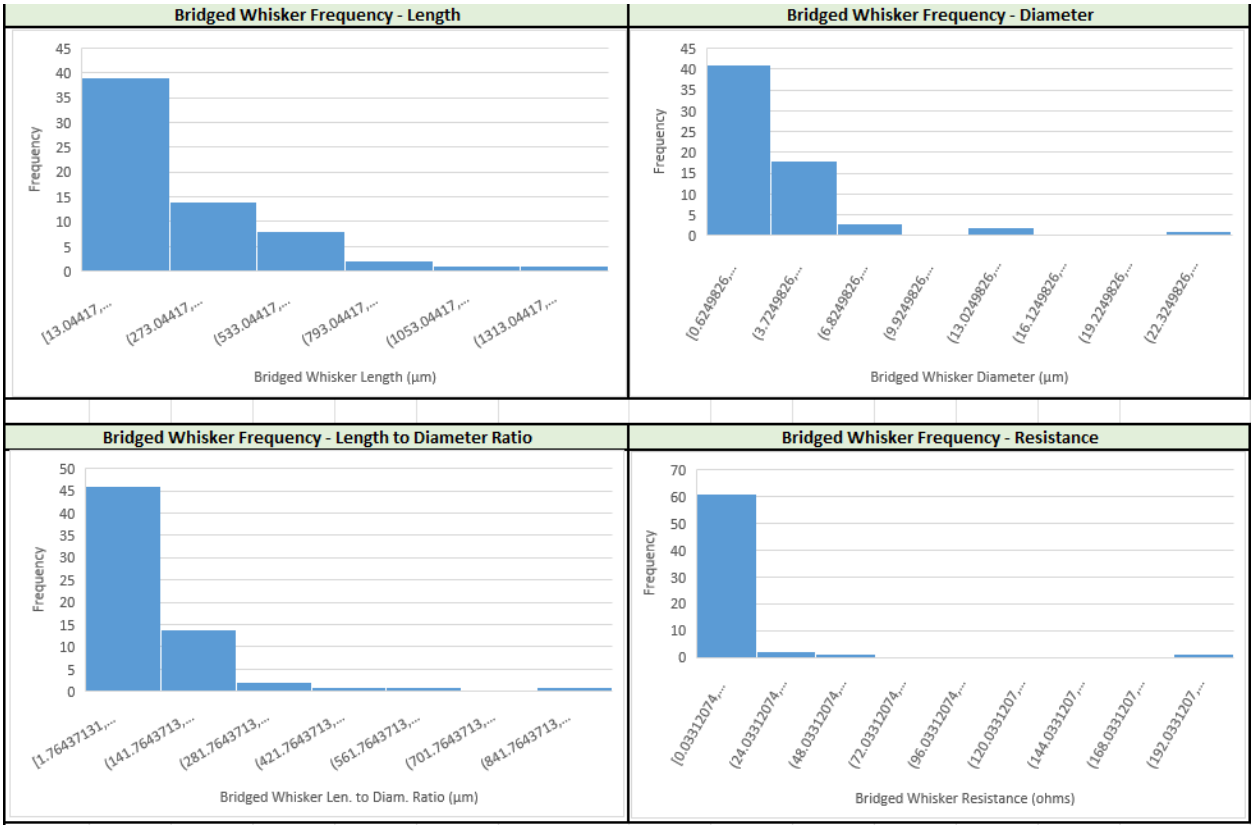


Figure 30: Frequency Plots

Extra Features

Section 1: Filtering Objects Based on Material

This is a built-in feature that the Unity Editor offers. What this allows users to do is when in the Scene view (Simulation is not running, users are examining their board or fixing orientation), they can filter objects based on the materials the components are made of. This can be useful when figuring out which components need to be labeled as conductive, as is done in **Running the Simulation, Section 2.**

a. To do this, navigate to your Project tab, into the Assets folder, and then to the Boards folder where your Board should be stored.

b. Press the small arrow to the right of your PCB icon to open up all components of the board.

c. Find any material you wish to see highlighted, and right click to bring up a new menu.

d. Click on Find References in Scene. This will highlight all objects containing the material you selected.

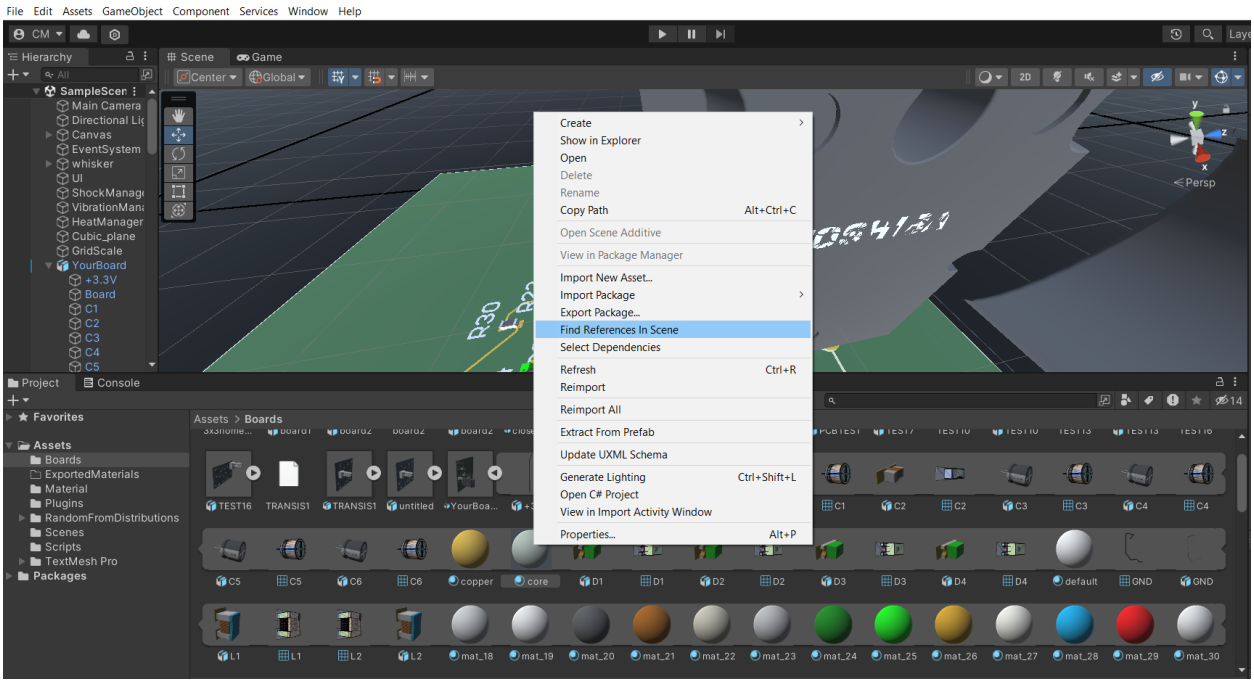


Figure 31: Material Filtering

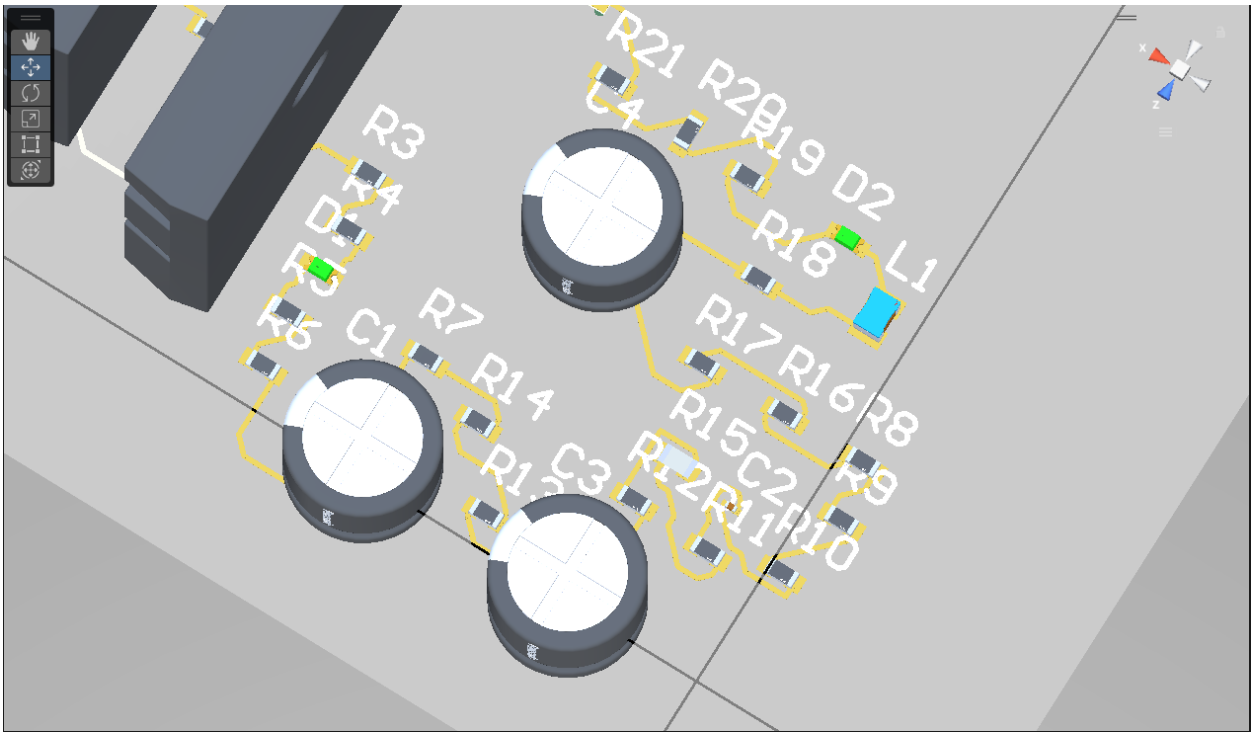


Figure 32: Filtering for Copper

Section 2: Changing Iteration Time Limit

Users can change the time limit between iterations simply through the use of the Unity Editor. The original time limit is 5 seconds but can be increased or decreased depending on the situation at hand. To do this, follow the below instructions:

a. In the Editor, under the Hierarchy, click on the UI object.

b. In the Inspector will be a large section containing many fields and objects. Scroll until a field titled Sim Time Thresh is present.

c. Change this from 5 to any number desired.

The sim time will now have a different time step between iterations, increasing or decreasing the speed of the simulation.

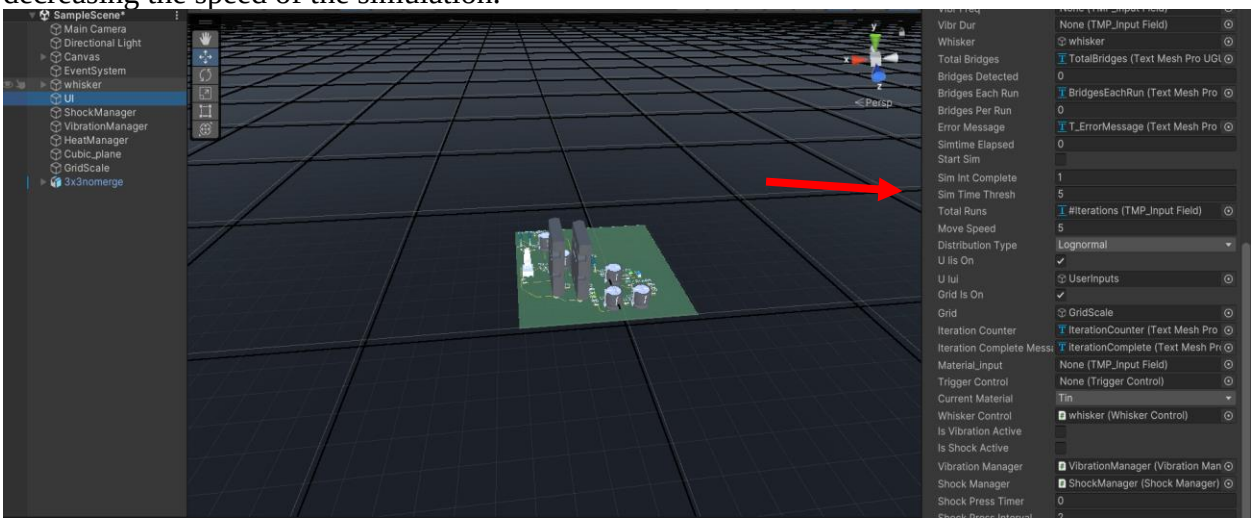


Figure 33: Changing Time Steps

Quick Overview

This section of the manual was designed to give users who are familiar with the simulation and already have it saved into their Unity Editor a brief overview of the instructions as a reference or reminder.

a. Import your board into the scene: Drag and drop your board from the file explorer on your personal device into the Boards folder in the Project tab, under Assets.

b. Orient the board if needed using the rotation parameters in the Inspector.

c. Add the Trigger Control script to the imported board and assign the Trigger Material and Material Input objects in their respective fields.

d. Press the Play button to run the simulation.

e. Fill in your Whisker Properties and search the board for any particular materials using the Conductive Material Selection input field.

f. Fill in your Simulation Control Parameters.

g. Set your desired number of iterations.

h. Press Start

i. Change parameters and inputs if needed, and press Reload to generate new whiskers.

j. If you are ready to save data, input a directory and file name into the inputs, and press Save.

k. If you desire to have one of the external forces to be applied, open the desired interface, fill in the inputs, and click the toggle to tell the simulation the forces should be applied.

l. Press Confirm Gravity to begin the simulation

m. Once the simulation is complete, the data will be stored into the csv file with the name you have given.

n. Copy and paste the data from the csv file into the WhiskerResults file included in the repository.

o. To reset the simulation, press Reset Gravity > Clear the Directory and File Name and press Save > Enter in new parameters and press Reload > Press Reset Sim

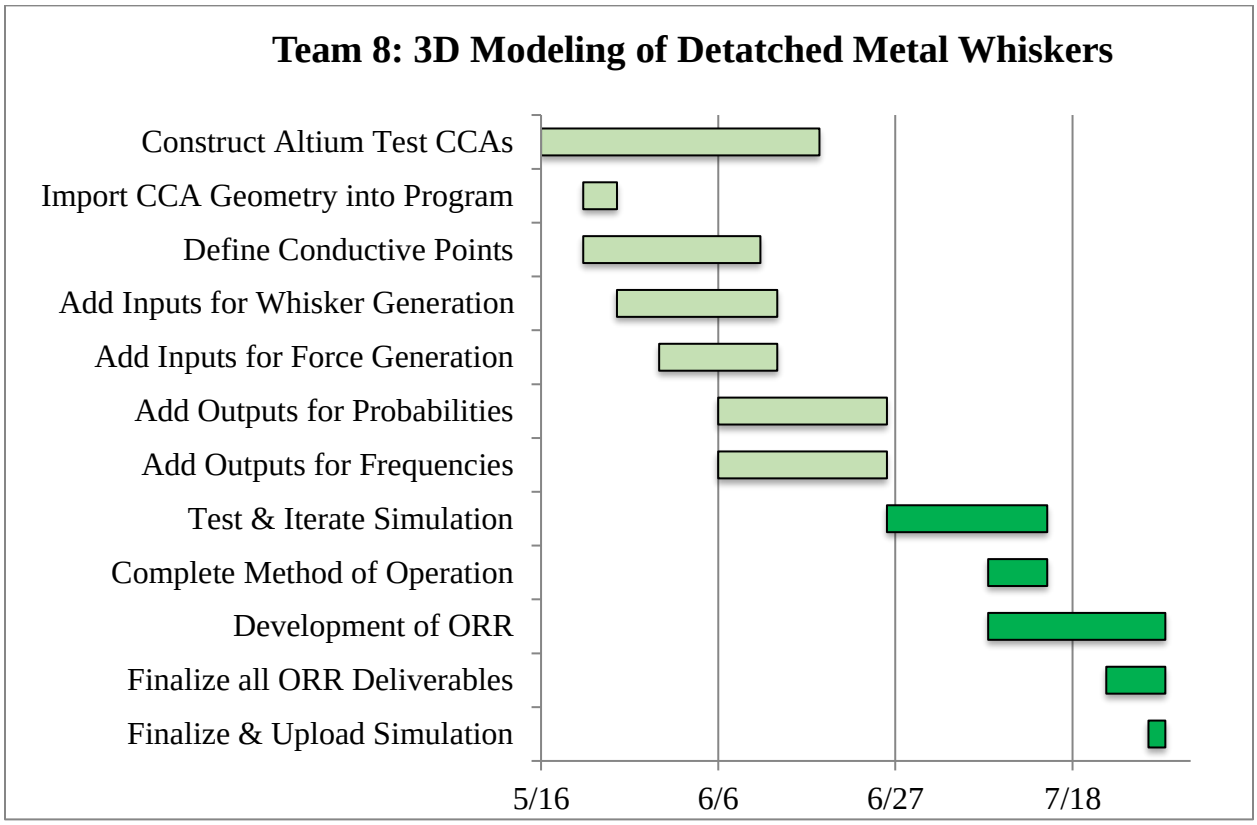
p. Alternatively, press the Play button again to fully stop the simulation, and press again to reopen.

Closing

This program has been designed with reliability and ease of use in mind. The goal of this user manual has been to provide a useful guide on how to use this program to its fullest potential. By following the instructions and tips provided, you'll be equipped to navigate every aspect of the program, regardless of experience with programs such as Unity or GitHub. It is highly encouraged to experiment with this tool and get the very most out of what it has to offer. Watch for updates being pushed out by Auburn's secondary team of engineers as they are made available, as they have been tasked with improving this simulation design further.

APPENDIX C

Gantt Chart



LIST OF REFERENCES

[1] Panaschenko, L. (2012) *The Art of Metal Whisker Appreciation: A Practical Guide for Electronics Professionals, NASA.* https://nepp.nasa.gov/whisker/reference/tech_papers/2012-Panaschenko-IPC-Art-of-Metal-Whisker-Appreciation.pdf (Accessed Feb. 1 2024).

[2] M. Sampson and H. Leidecker, "Basic info on tin whiskers," NASA, https://nepp.nasa.gov/whisker/background/index.htm (accessed Feb. 24, 2024).

[3] J. Nitin, "Tin whisker," YouTube, https://www.youtube.com/watch?v=MJjIclFh7hY (accessed Feb. 12, 2024).

[4] "Mechanism of Tin Whisker Growth in Electronics," Mechanism of tin whisker growth in Electronics, https://cave.auburn.edu/rsrch-thrusts/mechanism-of-tin-whisker-growth-in-electronics.html (accessed Feb. 12, 2024).

[5] "What is PCB tinning?" PCBA Store, https://www.pcbastore.com/blogs/pcb-tinning.html (accessed Feb. 14, 2024).

[6] G. Gaylon, A History of Tin Whisker Theory: 1946 to 2004, https://hlinstruments.com/RoHS_articles/A%20History%20of%20Tin%20Whisker%20Theory%201946%20to%202004.pdf (accessed Feb. 14, 2024).

[7] "Spaceflight now: Breaking news: U.S. galaxy 7 television satellite lost in Space," Spaceflight Now | Breaking News | U.S. Galaxy 7 television satellite lost in space, https://spaceflightnow.com/news/n0011/25galaxy7/ (accessed Feb. 14, 2024).

[8] P. Daddona, "Dominion learns Big Lesson from a tiny tin whisker," Reactor Shutdown: Dominion Learns Big Lesson From A Tiny "tin whisker,"

https://nepp.nasa.gov/whisker/reference/tech_papers/2005-dadonna-nuclear-reactor-shutdown.pdf (accessed Feb. 17, 2024).

[9] A. Shrivastava, “Dielectric strength,” Dielectric Strength – an overview | ScienceDirect Topics, https://www.sciencedirect.com/topics/engineering/dielectric-strength (accessed Feb. 25, 2024).

[10] “CALCE Tin Whisker Induced Metal Vapor Arc 70 torr,” YouTube, https://www.youtube.com/watch?v=MJjIclFh7hY (accessed Feb. 18, 2024).

[11] “Tin Whisker Mitigation Strategies: Cleaning or Coating?,” Chemtronics, https;//www.chemtronics.com/tin-whisker-mitigatin-strategies-cleaning-or-coating (accessed Feb. 20, 2024).

[12] T. A. Woodrow and E. A. Ledbury, Evaluation of Conformal Coatings as a Tin Whisker Mitigation Strategy, https://nepp.nasa.gov/whisker/reference/tech_papers/2005-Woodrow-tin-whisker-conformal-coat-presentation.pdf (accessed Feb. 20, 2024).

[13] Anoplate NEWS, “The trouble with tin: Get the lead out!,” Anoplate, https://www.anoplate.com/news-and-events/tin/whiskers/ (accessed Feb. 20, 2024).

[14] Y. Telles, Unity Pros and Cons, https://ventionteams.com/blog/unity-pros-and-cons (accessed Feb. 20, 2024).

[15] iXie, “Top 5 coding languages compatible with the Unity Game Development Engine ,” iXie Gaming, https://www.ixiegaming.com/blog/top-coding-languages-unity-game-development/ (accessed Feb. 20, 2024).

[16] W. Kenton, “Monte Carlo Simulation: History, how it works, and 4 key steps,” Investopedia, http://www.investopedia.com/terms/m/montecarlosimulation.asp (accessed Feb. 20, 2024).

[17] L. Panashchenko, "Evaluation of Environmental Tests for Tin Whisker Assessment." Order No. 1474312, University of Maryland, College Park, United States -- Maryland, 2009.

[18] U. Technologies, “GameObject,” Unity Documentation, https://docs.unity3d.com/Manual/class-GameObject.html (accessed Apr. 5, 2024).

[19] U. Technologies, “Materials introduction,” Unity Documentation, https://docs.unity3d.com/Manual/materials-introduction.html (accessed Apr. 9, 2024).

[20] “Creating and Using Materials,” Unity Documentation, https://docs.unity3d.com/2019.3/Documentation/Manual/Materials.html (accessed Apr. 15, 2024).

[21] “Raspberry pi 3D model + download,” RaspberryConnect, https://www.raspberryconnect.com/projects/32-multimedia/134-raspberry-pi-3d-model-download.

[22] “Compare Altium Designer® vs. Solidworks® PCB,” Altium, https://www.altium-designer/compare/solidworks-pcb (accessed Apr. 12, 2024).

[23] “Altium designer tutorials – how to create your first PCB,” YouTube, https://www.youtube.com/watch?v=-o5TMYgURdM (accessed Apr. 15, 2024).

[24] “What is a PCB connector?,” Harwin, https://www.harwin.com/blog/what-is-a-pcb-connector/ (accessed Apr. 18, 2024).

[25] “How to use 3D view mode in Altium designer | PCB design for Beginners,” YouTube, https://www.youtube.com/watch?v=g_LO-i9GcaE (accessed Apr. 14, 2024).

[26] “Altium designer [export .OBJ file to Keyshot],” YouTube, https://www.youtube.com/watch?v=qA-Awv2vicI (accessed Apr. 14, 2024).

[27] J. Frost, “Normal distribution in statistics,” Statistics By Jim, https://statisticsbyjim.com/basics/normal-distribution/ (accessed Apr. 18, 2024).

[28] M. Pavlovic, “Log-normal distribution - a simple explanation,” Medium, https://towardsdatascience.com/log-normal-distribution-a-simple-explanation-7605864fb67c (accessed Apr. 18, 2024).

[29] A. Katz et al., “Log-normal distribution: Brilliant Math & Science Wiki,” Brilliant, https://brilliant.org/wiki/log-normal-distribution/#:~:text=The%20median%20of%20a%20distribution,the%20highest%20probability%20of%20occurring (accessed Apr. 18, 2024).

[30] “Gravity of Mars,” Wikipedia, https://en.wikipedia.org/wiki/Gravity_of_Mars (accessed Apr. 17, 2024).

[31] J. Zhao, “Basics of structural vibration testing and analysis,” Crystal Instruments, https://www.crystalinstruments.com/basics-of-structural-vibration-testing-and-analysis#:~:text=Sinusoidal%20vibration%20is%20not%20very,in%20Figure%201%20(top). (accessed Apr. 18, 2024).

[32] one half-sine signal with the pulse width of 6 ms. (b)... | download scientific diagram, https://www.researchgate.net/figure/a-One-half-sine-signal-with-the-pulse-width-of-6ms-b-Amplitude-frequency_fig5_329050626 (accessed Apr. 18, 2024).

[33] “Types of shock testing machines - understanding vibration testing,” VRU, https://vru.vibrationresearch.com/lesson/types-shock-testing-machines/ (accessed Apr. 17, 2024).

[34] Hutch and Samuel, “Array of an unknown length in C#,” Stack Overflow, https://stackoverflow.com/questions/599369/array-of-an-unknown-length-in-c-

sharp#:~:text=Use%20List%20to%20build%20up,array%2C%20and%20not%20a%20List%20.&text=For%20%me%2C%20the%20List%20solved,the%20solved,the%20unknown%20size%2C%20and%20then%20 (accessed Jul. 8, 2024).

[35] U. Technologies, “Inputfield.OnValueChanged,” Unity, https://docs.unity3d.com/2018.1/Documenttaion/ScriptReference/UI.InputField-onValueChanged.html (accessed Jun. 28, 2024).

[36] Glurth, “Check if Inputfield is empty – questions & answers – unity discussions,” Unity, https://discussions.unity.com/t/check-if-inputfield-is-empty/160502 (accessed Jul. 8, 2024).

[37] “Tutorial Altium designer how to set PCB board sizes perfectly,” YouTube, https://www.youtube.com/watch?v=3VoyTh_pZRA&t=174s (accessed Jun. 12, 2024).

[38] “Linear Dimension,” Altium Documentation, https://www.altium.com/documentation/altium-designer/pcb-linear-dimension?version=21 (accessed Jun. 4, 2024).

[39] “Switching units from MM to mils and other PCB design measurement preferences,” Altium, https://resources.altium.com/p/switching-default-uint-and-other-pcb-design-measurement-preferences (accessed Jun. 4, 2024).

[40] “RF Power Amplifier Module PCB design,” Altium, https://resources.altium.com/p/rf-power-amplifier-module-pcb-design (accessed Jul. 12, 2024).

[41] “Import Obj and MTL file – unity,” YouTube, https://www.youtube.com/watch?v=0SzuPgFQO-8 (accessed May 18, 2024).

[42] “Tablecloth whip-off Trick,” Steve Spangler, https://stevespangler.com/experiments/tablecloth-science-

trick/#:~:text=Plain%20and%20simple%2C%20Tablecloth,the%20object%20to%20move%20it. (accessed Jul. 23, 2024).

[43] "U3DS - Heatmap test," YouTube, https://www.youtube.com/watch?v=fRn-Bmi3w8U (accessed Jul. 23, 2024).

[44] OBJ files: Uses & 3D applications – adobe substance 3D, https://www.adobe.com/products/substance3d/discover/what-are-obj-files.html (accessed Jul. 18, 2024).

[45] P.M. Games, "Probulder Unity | drawing 3D shapes with polyshape tool," Youtube, https://www.youtube.com/watch?v=lr9cU0ejY8U (accessed Jul. 23, 2024).

[46] "Sci-fi style geometry nodes modeling in Blender 3.0 & Eevee," YouTube, https://www.youtube.com/watch?v=app=desktop&v=LgQSBcKwUCM (accessed Jul. 28, 2024).

[47] Wintersbane, Finding vector3 position of GameObject – Questions & Answers – Unity Discussions, https://discussions.unity.com/t/finding-vector3-position-of-gameobject/148031 (accessed Jul. 24, 2024).

[48] "Modifying existing routes," Altium Documentation, https:// www.altium.com/documentation/altium-designer/modifying-existing-routes-pcb#routing-aware-move-component (accessed Jul. 22, 2024).

[49] Ekta-Mehta-D, "Electric shock wave effect 2D – Questions & Answers – Unity Discussions," Unity Discussions, https://discussions.unity.com/t/ electric-shock-wave-effect-2d/71778 (accessed Jul. 24, 2024).

[50] "Why do my object files look weird when imported to unity?," Stack Overflow, https://stackoverflow.com/questions/72842393/why-do-my-object-files-look-weird-when-imported-to-unity (accessed Jul. 23, 2024).

[51] "Flip normal," Keyshot Manual, https://manual.keyshot.com/manual/tool-3/flip-normals/ (accessed Jul. 28, 2024).