Sam LaPree

FCSM Research: Digital Twins

Cyber-physical systems (CPS) are a field of technology involving blending physical and digital components, in which algorithms are computed to produce an outcome for the physical component. This relationship can also work the other way by having sensors from the physical component feed data and information into the digital component to help modify what the best output the digital side should tell the physical side to do. The recent advances in sensing and communication technologies and Internet 4.0 have had a massive impact on CPS’ ability to automate things, as some devices no longer need to be physically present next to each other to communicate. In the last few years, this has resulted in CPS evolving in many ways by having more and more work, quality of life, and safety processes in various fields be automated through CPS since many devices can be connected, and the digital component has enough information from all these devices to produce a desired output for each device. The aspect on which this independent research focused on is a subsidiary of CPS called Digital Twins (DT).

The world of technology is rapidly evolving through new technologies emerging to solve daily problems. One of these emerging fields of technology is digital twins. Even though the use of digital twins in the present is realistic and applicable to many scenarios, certain fields of everyday life don’t have sufficient solutions to problems they face due to poor management, tedious tasks, or unpredictability, all of which can be minimized with digital twins. Digital Twins (DT) being a sub-category of CPS has all the elements of CPS but has the inclusion of potential possibilities in which DT can use to separate itself from CPS such as simulation, optimization, real-time analyzation, data aggregation, and more. Digital Twin is also different, as it refers to its physical, and digital components as twins as the relationship between the twins is that of simulation and creating a digital copy of the physical component. IBM describes Digital Twins as “a virtual model designed to accurately reflect a physical object. The object being studied—for example, a wind turbine—is outfitted with various sensors related to vital areas of functionality. These sensors produce data about different aspects of the physical object’s performance, such as energy output, temperature, weather conditions and more. This data is then relayed to a processing system and applied to the digital copy. Once informed with such data, the virtual model can be used to run simulations, study performance issues, and generate possible improvements, all with the goal of generating valuable insights—which can then be applied back to the original physical object” (IBM). This explains the relationship between the twins and explains how different data values can be used, and what kind of data you might want to look at for certain examples. Often with DT, we need a bridge to connect the physical twin and the digital twin which is called the middleware. Typically, the middleware used for digital twins is a cloud service, or some sort of hub for IoT that is able to connect the physical component to this cloud service through the use of internet access. The other half of the bridge is the digital twin to the middleware which is typically done with plugins, APIs, interfaces, and data streams which can all be connected through the internet as well.

For my project, I originally intended to create a smart thermostat, or a working prototype of a smart thermostat with the concept of digital twins. My physical twin components consisted of a thermostat, which for the purposes of this research project would be a simulated thermostat and would also include DHT11 sensors which are able to pick up data readings of temperature and humidity in its current environment. The last component of my physical twin is my Raspberry Pi 4B which would be used to store the physical data and send it to the middleware. The middleware I originally was using for this project was an IoT hub on a cloud service in which I connected my Raspberry Pi to it through the use of Device IDs, and other authentication techniques for connecting hardware to IoT. The digital twin side of my project would have consisted of multiple weather APIs for local weather in Towson, in which I planned to aggregate and blend the data in a more consumable, and valid data input to feed into the tools I planned to be using to analyze what course of action my physical twin should take. When I originally would have been ready for my data input I would have sent the data back to my Raspberry Pi, which this time around, the software in the Raspberry Pi would be playing the role of the digital twin as it would look at the data and from software I planned to develop, and would tell the physical twin what course of action needs to be taken by sending this new action back to the IoT hub, and back to the physical twin or in this case the thermostat.

The reason I chose to use the concept of Digital Twins for this matter is that I planned on simulating a thermostat in this research project instead of using a physical one and be aggregating data to optimize an output which in the scope of the project was, what temperature the thermostat should be set to based off the input that it is given. Another reason I chose to go on the path of digital twins was for the scalability of a project like this as one IoT hub can reach out to many different hardware components, and if I choose to further the research past the current stages I have gone through for it, I will be able to test the research on a larger scale with more data inputs, as well as more thermostats, and simulations being ran. The last reason I chose to go with digital twins, is that my project falls under the category of smart cyber-physical systems. Smart cyber-physical systems can be described as, “Methods for engineering Cyber-physical Systems that are able to respond in real-time to dynamic and complex situations while preserving control, system safety, privacy, reliability, energy efficiency and dependability features, and addressing security and privacy ""by design"" across all levels. This includes CPS that are aware of the physical environment, enabling effective and fast feedback loops between actuation and sensing, possibly with cognitive and learning capabilities” (CORDIS). For my project to be successful, I needed my project to take in real time data and make real time changes to the thermostat based upon current data readings.

From the research I conducted, I hope that I was able to make meaningful contributions to the field of digital twins, as well as home improvement, and energy management. One of the main inspirations of this project is to better managed and limit the use of energy when it comes to heating and cooling homes since there is often a lot of wasted energy output from changing a thermostat from one temperature to another drastically all the time. With this research I hope that I was hoping to be able to achieve a product that will make homes feel completely neutral in regard to temperature, and so that residents will no longer feel the need to touch the thermostat, and that the temperature stays regulated and can make small minor adjustments when necessary to limit the energy and power consumption.

**Literature Review:**

Digital twin paradigm: A systematic literature review:

This article provides a beginner’s analysis of what a Digital Twin (DT) is, along with its rise in the context of Industry 4.0. It outlines the different components that constitute a DT system, including the digital twin, the physical twin, and the middleware, which acts as a bridge between the two. The article further discusses the various stages of the DT lifecycle, comprising design, manufacturing, and service stages. Additionally, it explores the uses of DTs and the methods for implementing or integrating them into projects. Overall, the article offers an introductory overview of Digital Twins by explaining their definition, components, stages, and applications.

Characterising the Digital Twin: A systematic literature review:

This paper discusses the recent growth and popularity of a digital twin approach to many technologies. The paper also goes over the components and configuration of the framework of Digital Twins. There are also many gaps in current understanding and knowledge of the topic of digital twins and the paper goes into detail to fill those common gaps and clear many misunderstandings of the topic. Lastly the paper talks about what innovations and ways digital twins can be implemented in the future.

The Past, Present and Future of Cyber-Physical Systems: A Focus on Models:

The paper overviews the history of Cyber-Physical Systems (CPS) and details the different models of CPS throughout the years, and what is currently used along with the challenges researchers have had with the development of CPS throughout its innovation. The paper specially cites two important projects, that being, PRET (Precision Timed Machines) and Ptides (programming temporally-integrated distributed embedded systems) with PRET focusing on provide precise and fast timing at the software abstraction level, and enhancing the predictability of system behavior while Ptides focuses more on creating deterministic models. Lastly the paper discusses places where CPS can improve and where future work can be done so that CPS is more accurate and reliable.

Cyber Physical Systems: Analyses, challenges and possible solutions:

The paper lists the issues and difficulties of operating a Cyber-Physical System (CPS), while also giving an analysis of those same issues. The paper also notes the importance and necessity of CPS in certain technologies in the current Industry 4.0 stage such as with it being critical in self-driving vehicles, and smart grids. Next the authors of the paper highlight the vulnerabilities and weaknesses of CPS to cyber-attacks. Additionally, the paper highlights the importance of real-time data processing and communications and compatibility between systems when it comes to dealing with CPS.

Human Digital Twin (HDT) Driven Human-Cyber-Physical Systems: Key Technologies and Applications:

The paper analyzes the integration of Human Digital Twins (HDT) in Human-Cyber-Physical Systems (HCPS). The paper explains that with HDT and HCPS, the technologies involved need a much larger focus on human decision making and interaction. The paper also details some current day issues with HCPS such as consistent real-time context perception and control in dynamic environments and that these issues can be fixed through the use of AI, IoT and cloud computing by making the system more adaptable and flexible. This would make the technology work more effectively in these dynamic environments and will speed up the context perception. The paper ends off by detailing the importance of the continuous evolution and interaction between human and machine elements in the realm of HCPS, and how human creativity and decision-making are things that are difficult to replicate with technology and that those qualities are not inferior to technology.

Digital Twins and Cyber–Physical Systems toward Smart Manufacturing and Industry 4.0: Correlation and Comparison:

This paper details the similarities and differences between Cyber-Physical Systems (CPS) and Digital Twins (DT) and their relevance in Industry 4.0. Both concepts involve physical and digital worlds that often are used to optimize processes and make predictions to make systems more efficient. The authors argue that DT and CPS both share the common goal of cyber-physical integration and optimization, however that they differ when it comes to both their application and integration. CPS often focuses more on real time data processing, whereas DT focuses more on virtual models of real-world objects to simulate. The authors argue that CPS lays the groundwork for DT to exist, and that DT is part of the ever-growing evolution of CPS, and that DT will play a role in its continued evolution.

**Problem Statement: (What are you handling? What are you showing in terms of output? System setup)**

The research I am conducting is to create a smart thermostat that is able to better at handling at keeping a steady temperature with outside changing of temperatures surrounding it, as well as creating a better energy management solution for the thermostat. I plan to accomplish this by creating a digital twin of an apartment interior. The way I planned to use this digital twin was to be to perform some level of remote monitoring. Towards this end, I collected the real-world sensor data from a room, and that data was sent to my middleware where it will be blended with other relevant temperature data, and then lastly be sent to the digital twin. Once the data is sent to the digital twin, the digital twin checks to see what course of action should be made based off of its data input. Then based off of what the digital twin says to do, it returns the command back to the physical twin so that it can carry out the proper function in real life.

In terms of output, my current research was originally set out to show a consistent stream of temperature and humidity data that would be blended with other data from online weather APIs. This data blending should give an accurate prediction to what future indoor temperatures will look like based off immediate current temperatures, and what the weather APIs predict so that the smart thermostat built for that specific interior around it is as precise as can be for its unique environment. The other output of my research should be some sort of user interface that would display the relevant temperature data to the user such as current temperature and humidity, along with if the AC, or heating functions will go into effect. A diagram of a digital twin can be seen below displaying the type of set up I was hoping to accomplish for the final product.

A blue and grey room with text

Description automatically generated with medium confidence

The diagram displays a virtual room that mimics the floor plan of an apartment. Each room in the apartment has an independent sensor set up that detects temperature and humidity, and connects with my middleware, which processes each independent sensor data. Each room also has its own heating/cooling equipment that follows the directions of the data sent from the middleware after it has been blended and processed. This way, each room can have it’s own unique temperature set, and have a more energy efficient temperature setting method, rather than all temperature equipment running at once, all the time. This digital twin would then relay the corresponding relevant information to the real-life physical twin so that each room can make the real life changes to each room, which can be in 3 different states. The 3 different states each room can be in is Heating, if the sensor is picking up temperature data that suggests the room is heating up too much past the user threshold that they would like the room temperature at, Cooling if the room is becoming too cold, and Neutral if the room is within the desired threshold, and therefore should not be performing any actions at that time.

In order to accomplish my research, I had to go through a system setup, and configure all of the hardware to be compatible with the other components, and software being used. The first part of my system set up was I had to make sure my Raspberry Pi was fully functioning. I first plugged in all the components into the Raspberry Pi such as the power switch, along with connecting an HDMI, to a monitor. I connected a compatible mouse and keyboard to navigate and operate the Raspberry Pi. When initially turning on the Raspberry Pi I had to select and download an operating system to use for the Raspberry Pi. To download the operating system, I downloaded the Raspberry Pi imager on my computer and transferred the imager file to a SD card where I later inserted it into the Raspberry Pi so that the Raspberry Pi can use the new operating system. The version I decided to use was imager version 1.8.5 since it was the most updated version that was compatible with my model of my Raspberry Pi which is a Raspberry Pi 4B. The next part of my set up was to connect my Raspberry Pi to the internet so that it can work and connect with the middleware in the future parts of the project. Connecting to the internet was intuitive as Raspberry Pi is very user friendly and allows for easy connection internet. The other part of the set up was the sensor bed set up. I connected the DHT11 sensors to the breadboard and then also connected some female-to-male wires from the breadboard to the GPIO pins on the Raspberry Pi. I connected the signal pin on the sensor to GPIO pin 7, which functions as GPIO 4, which allows for the main functionality of the sensor to operate. Then I connected the vcc pin on the sensor to the GPIO 2 pin which is the power pin which turns the sensor on. Lastly, I connected the ground pin on the sensor to GPIO pin 6 which is the ground pin which makes sure the electrical circuits traveling through the wire connections stay grounded.

**Experiments:**

For my research I needed a couple of components and hardware to conduct and create my project. A table list of the project components and their purposes can be found below.

|  |  |
| --- | --- |
| Component | Component Purpose |
| Raspberry Pi 4B | Physical Twin that also connects to the sensor to collect data and send data to middleware. |
| Raspberry Pi Power switch | Turns on and powers the Raspberry Pi. |
| Micro HDMI to HDMI cable | Allows for display of Raspberry Pi on monitor. |
| Keyboard and Mouse | Allows for navigation and operation of the Raspberry Pi. |
| SD Card | Allows for downloaded Raspberry Pi operating system to be loaded onto the device. |
| Breadboard | Allows sensor to be connected to Raspberry Pi and holds sensors in place. |
| Wires | When connected to the breadboard properly, allows for sensor data to be transported to the device. |
| DHT11 Sensor | Allows to pick up temperature and sensor data from the environment around it. |

**Configuration:**

The main component of this project, being the Raspberry Pi 4B, needed some configuration to use the features and functions I needed to accomplish my goal of this research. First, I needed to configure the initial connection between my Raspberry Pi which is my physical twin to the middleware which I decided to use Azure Dev Ops. Establishing this connection would allow me to send future data to an IoT hub on Azure which could hold and store the temperature data that would then be blended with Weather APIs and then could be sent back to my Raspberry Pi so that the user interface can be updated.

For establishing the connection between my Raspberry Pi and my IoT hub, I fist had to create the IoT hub in the first place. When making my IoT hub, there were other things that needed to be created first to be used for the IoT hub, such as a resource group which serves to manage the resources for my IoT hub such as incoming data, and access control. I also needed to create a device on the IoT hub that has a unique ID so that when my Raspberry Pi sends messages to the hub, it can do it under this registered device ID. After I made my IoT hub, I also needed to make a container or place to store my future data, and where I can check that my messages and properly sent to the IoT hub. I decided to go with a blob container as it was the simplest option to set up, and all I aimed to do was make sure my messages went through, but in the future another container type may be considered if I find one that handles numerical data better.

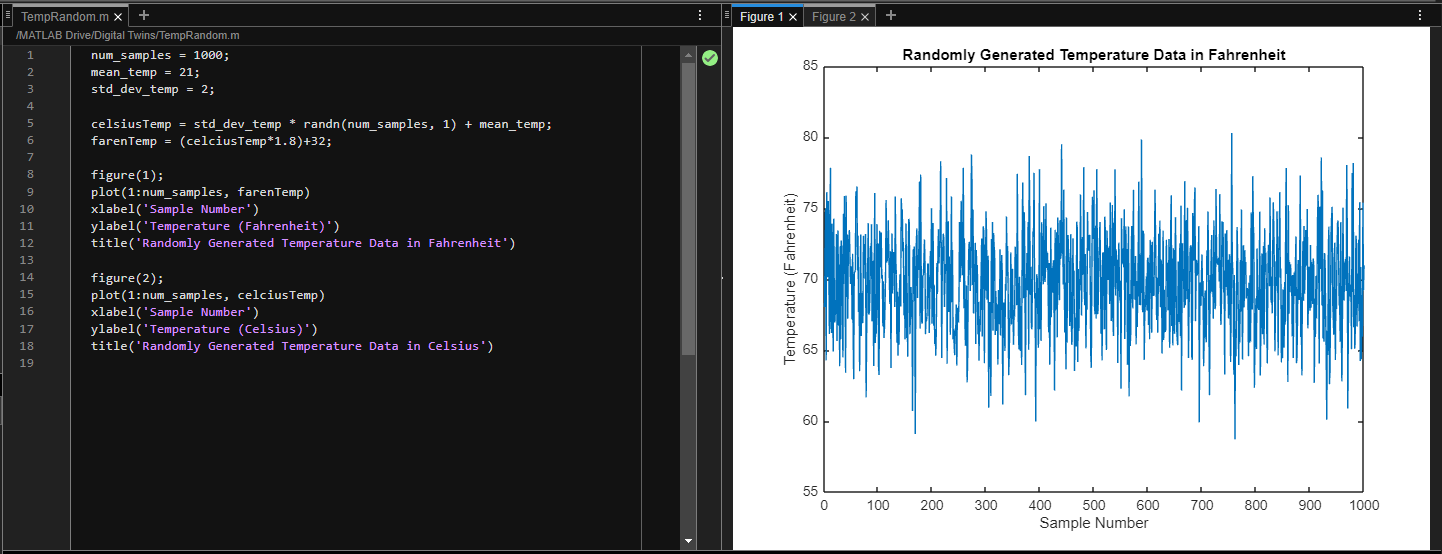
After the Azure IoT hub was set up, I needed to prepare my Raspberry Pi to handle sending messages to my IoT hub by ensuring all the proper Azure packages were installed onto it, and by developing a test program to send out a test message to check if the connection has been made. When installing packages, the main package that I needed to install was Azure SDK which allowed for the compatibility between Azure Dev Ops, and my Raspberry Pi. There were some issues with it along the way as I needed to update all my programming languages on the Pi to the newest versions, as well as set up a virtual environment that has pip installed. There was one particular issue I ran into with setting up and configuring the build for Azure SDK where there were many text files missing from multiple folders that needed certain text files so my terminal could read them but I managed to find an old GitHub forum that provided terminal code that fixed this issue, and once I passed that issue, the only thing I needed to worry about was writing a program to actually send the messages/data.

When writing my program, I decided to use python as it was already installed to the newest version on my Raspberry Pi and would meet the requirements I needed for this project. I found a simple template online for sending messages to an Azure IoT hub, and filled in the relevant variables I needed to insert such as my IoT hub connection string which allows for the connection to the hub, the device id which serves to authenticate the device to send messages, as well as the container connection string, which allows the message/data to be routed to the desired container. Eventually I was able to see encrypted messages sent from my Raspberry Pi meaning the initial connection had been made.

Unfortunately, later on in the project process, I found that I needed to switch middleware as my free student subscription to Azure’s services ran out and could not be funded by the university. I then moved on to moving MongoDB as my middleware since it is a free online database that does not require any subscriptions or payments. My DHT11 sensor also broke down and was unable to pick up signals, and collect data so I temporarily moved onto creating random data and sending it to my middleware in the meantime to compensate for the lack of data collection. A picture of the Raspberry Pi operating with a light sensor can be seen below when testing the compatibility of the sensor.

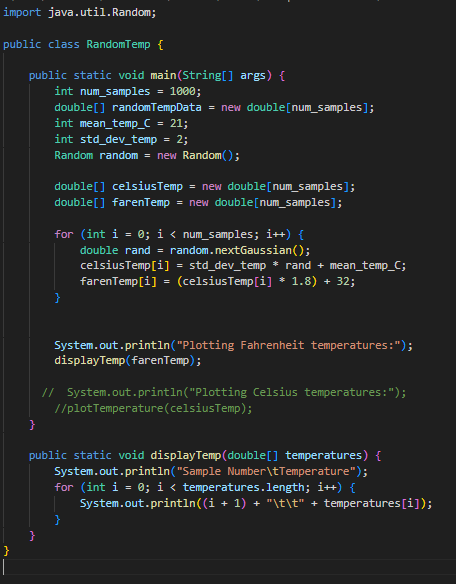
A circuit board with wires and wires

Description automatically generated For generating random sensor data, I originally was using MATLAB since it is capable of producing random mathematical data efficiently. I found a code template online for generating random temperatures in Celsius. From there I modified the template to change the number of samples being produced and kept the standard deviation in their random number function the same since I am not as familiar when it comes to producing random data. I also converted the temperatures to Fahrenheit since that is the temperature unit of measure, I am most familiar with.



As soon as I had a consistent random output from MATLAB, I soon found out that many of the extensions required for me to connect MATLAB to MongoDB were locked behind a premium version which I did not have access to since I was only using the student version. I then decided to switch the language I was using to generate my random temperature data to Java since Java is the language I am the most familiar with. I then translated my code that I developed in MATLAB to Java with the assistance of some online research to determine the best course of action to generate the best random data since Java is known for not creating the best random data. From there I was able to generate data that closely imitates what random data I was able to produce from MATLAB.

The next step of my research this week was to create the MongoDB database and establish the connection between my computer to MongoDB. When creating my MongoDB database I was mostly concerned with establishing the connection first, so I have yet to design the database, but I plan to add that to this week’s research work. When establishing the connection, I had to create a username and a user password for authentication when connecting to the database, as well as creating a connection string for my program to make the connection. I also needed to create a Maven project and move my Java program into the Maven project so I can use the MongoDB extensions in my Java program by adding the extension in the Maven project’s pom.xml file. For transferring the code to the Maven project, all I had to do was copy my code and paste in into the Maven project’s main class to run it. After running it a first I had some initial complications making the initial connection to the database, but I soon figured out the issues with the connection and authentication and was able to populate the data with the random data produced by the Java program which a screenshot of the code can be seen below along with part of the Maven project directory. The MongoDB database with its populated data and a corresponding chart can be seen after.



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated A green bar graph with numbers

Description automatically generated

**Final Experiment and Results:**

Based off the previous work I did with MATLAB and MongoDB, I decided to move forward with the direction of simulating the data and representing it on my computer as there were too many issues with the configuration of the Raspberry Pi and the sensor hardware. I decided to adjust the method in which I simulate the temperature data to Java. Even though MATLAB had better random data generation, the Java language offers more options for customizing how the data is created, Java allows for easier integration of a timer to continually, and dynamically generate data to populate the MongoDB middleware as well as delete old irrelevant data.

In order to connect my temperature generation program to MongoDB I also had to follow 3 steps to run my project. First, I had to install MongoDB on Visual Studio Code. The next 2 steps I had to go through every time I ran my project. The 2nd step, I had to open the command palette in Visual Studio Code which can be done by clicking the “View” tab and clicking “Command Palette”. Then I had to search "MongoDB: Connect" on the Command Palette and click on "Connect with Connection String." My last step was to connect to my MongoDB deployment by pasting in my connection string which for the purposes of this experiment was, “mongodb+srv://lapreesam:1492@cluster0.qycivic.mongodb.net/”

The main snippet of relevant code can be seen with comments in the screenshot below. The main code here generates 10 random temperatures every one minute, while deleting the previous temperatures after new ones are generated. The codes are also displayed in the terminal so that the user can ensure the temperatures on the MongoDB are the same ones generated by the Java code.   
  
A screen shot of a computer program

Description automatically generated

For the middleware being MongoDB, I decided to keep things the same as MongoDB is very intuitive, and very adaptable to adjust its purpose for multiple languages, and I used a similar way to store the data using a cluster within a database I created on MongoDB.

For the physical twin, I decided to create a user interface that would interact with the digital twin that creates simulated temperatures. For the user interface, I decided to write the user interface with Java Swing since I was familiar with that user interface programming, and it ensured compatibility with my already existing Java project from the digital twin side of the project. The user interface had to be connected to the MongoDB and pull the temperature data. The following screenshot shows the code snippet that connects the user interface with the MongoDB.

A screen shot of a computer code

Description automatically generated

Considering the rate at which I was populating the cluster with new data, which was 10 new temperatures every minute, I decided to let the user interface interact with a new temperature every 6 seconds. In the below screenshot you can view an example of what the user interface thermostat looks like.

A screenshot of a red screen

Description automatically generated

The user interface displays the current temperature at the top as well as a timestamp from when it was received/being displayed. It also displays the temperature at which the user sets the room for, and a new temperature can be set with the text filed and button under the current temperature information. By default, when you load the user interface, the temperature is set to stabilize around 70. Under the information of the status of the temperature to be set at, is a message telling what the thermostat would do if it was a real thermostat. In the case of the above example, the above temperature was a little above the set temperature, so it needs to cool down a little bit. If the user interface receives a temperature that is 3 degrees outside of its range, then it displays a message such as “Need to warm up a lot” or "Need to cool down a lot” since the temperature is relatively extreme to what should be set, and the thermostat should work a little harder to stabilize the room temperature. The background also changes based off what the current temperature is, for example, if the temperature is too hot for the set temperature, then the background will be displayed as red, if the temperature is too cold, then the background will be displayed as light blue, and if the temperature is perfect, then the background will be displayed as gray.

**Project Demonstration:**

The first part of my project demonstration, I will show how the terminal display new temperatures, and how those temperatures will be reflected in my MongoDB cluster.

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The next part of my experiment will show how the user interface interacts with the MongoDB cluster with new temperature data.

A screenshot of a computer

Description automatically generated

A video can be found beneath this paragraph for a live demonstration of how this works as well.  
Video download: 

YouTube video if download doesn’t work: <https://youtu.be/imoLynGoorM>

**Conclusion:**

Even though the project did not turn out the way I originally envisioned with the hardware and Raspberry Pi components, I believe I was able to demonstrate a use/purpose for Digital Twins and how they work and how the different components interact with one another. Given more time on this project, I believe I could implement a more realistic simulation of temperature data simulation that more closely follows how a room would interact with the thermostat and produce temperatures that correlate with the actions of the thermostat.

Works Cited

CORDIS. (n.d.). *Smart cyber-physical systems: Programme: H2020: Cordis: European Commission*. <https://cordis.europa.eu/programme/id/H2020_ICT-01-2014#:~:text=Smart%2C%20cooperative%20and%20open%20CPS,%22%22by%20design%22%22%20across>

Lee, E. A. (2015). The past, present, and future of cyber-physical systems: A focus on models. Sensors, 15(3), 4837-4869. <https://doi.org/10.3390/s150304837>

IBM. (2022b, June). What is a digital twin? <https://www.ibm.com/topics/what-is-a-digitaltwin>

Semeraro, C., Lezoche, M., Panetto, H., & Dassisti, M. (2021). Digital twin paradigm: A systematic literature review. Computers in Industry, 130, 103469. <https://doi.org/10.1016/j.compind.2021.103469>

Sethuraman, R., Gautam, R., Adhikari, R., & Sharma, S. (2021). Cyber physical systems: Analyses, challenges and possible solutions. Internet of Things and Cyber-Physical Systems, 1, 22-33. <https://doi.org/10.1016/j.iotcps.2021.12.002>

Tao, F., Qi, Q., Wang, L., & Nee, A. Y. C. (2019). Digital twins and cyber–physical systems toward smart manufacturing and Industry 4.0: Correlation and comparison. Engineering, 5(4), 653-661. <https://doi.org/10.1016/j.eng.2019.01.014>

Tao, F., Sui, F., Liu, A., Qi, Q., Zhang, M., Song, B., Guo, Z., & Lu, S. C. Y. (2020). Characterising the Digital Twin: A systematic literature review. CIRP Journal of Manufacturing Science and Technology, 29, 36-52. <https://doi.org/10.1016/j.cirpj.2020.02.002>

Wang, B., Zhou, H., Yang, G., Li, X., & Yang, H. (2022). Human digital twin (HDT) driven human-cyber-physical systems: Key technologies and applications. Chinese Journal of Mechanical Engineering, 35(1), 11. <https://doi.org/10.1186/s10033-022-00680-w>