A Multi-Room Heating System in MATLAB and Simulink

Cameron Baird
Department of Computer Science
Vanderbilt University
Nashville, TN, United States
cameron.j.baird@Vanderbilt.Edu

***Abstract*—While commonly used and well-known, systems for heating buildings are not trivial. There are many technical engineering decisions that go into creating a room heating system in terms of hardware and software that complicate the problem considerably. In this work, we present a model of a room heating system for a small building. While only for a class final project, building this system has given me insight into the difficulty of designing a simple and well-posed system, as well as the opportunity to learn MATLAB and Simulink through a project.**

***Index Terms*—Cyber-Physical Systems, MATLAB, Simulink, Heating System, Embedded Systems**

I. Introduction

Room heating systems have been around for a long time. It is common in many places around the world to use thermostats and heating systems in order to live in cold places without freezing to death. However, it is not a trivial task to design such a system, as there are many hardware and software requirements that could cause potentially life-threatening problems if not properly accounted for. Therefore, it is necessary to rigorously design such systems in order to avoid errors and injuries.

Here, we consider a specific example of a multi-room heating system with a fixed number of rooms and available heaters, as well as fixed requirements for how the system is allowed to operate. The specifications for this system can be found in the released document from Vanderbilt University's Dr. Abhishek Dubey, as well as in Section 3 of this paper.

In this report, I will provide the following contributions:

- A formal specification for the multi-room heating system (Section 3), including assumptions made about the system, input and output interfaces, dynamics, and safety and liveness conditions.
- An implementation of the multi-room heating system using MATLAB and Simulink (Section 4), as well as simulations of the system under different parameters (Section 5).

II. Problem Background

To set the scene for this paper, I will begin with a simple example of a possible approach for heating a multi-room building.

Let's assume that we have N rooms that we would like to keep between two temperatures: a lower bound L and an upper bound U. A simple and logical approach would be to purchase N heaters, and assign a single one to each room. If we assume that there is a way to measure the temperature of each room, a basic algorithm could be constructed as shown in Algorithm 1.

Algorithm 1 An algorithm for heating multiple rooms.

```
for r in Rooms do
    if temp(r) ≤ U then Apply Heat
    elseTurn Off Heat
    end if
end for
```

While this is a trivial algorithm, it provides a few things for this paper. First, it provides some baseline idea of what our heating system needs to do. Under the assumption that we are in a cold area and the outdoor temperature is far lower than the ideal one, this basic logic makes sense. However, it also provokes thought into the complexity of this problem; there are many tweaks, variations, and ideas that are important to consider. How many rooms and heaters are there? If we are trying to heat up an office building and there are hundreds of rooms, it may be expensive to place a heater in every room. Do we want to stop heating the room only when we reach the upper limit, or is there a smaller threshold value where we turn the heater off? It may get too hot in the room if this value is not properly set. What is the optimal number of heaters and the optimal placement of the heaters such that all the rooms are properly heated and we minimize the overall cost and energy consumption? These are difficult questions to answer; one could consider looking at this problem through an optimization lens, and applying different types of mixed integer programming techniques to model this problem. The point of all this is to say, while the idea and goal of the problem are simple the exact manner in which to make assumptions and solve the problem are not.

To solve this problem, we will consider a specific case of a multi-room heating system (discussed in depth in Section 3). Using these assumptions and applying physics, we will design a controller that updates the temperatures of each room and places the heaters in the correct places in order to keep each temperature value within a certain range. With simulations of

the proposed system, I will show that the control logic solves the problem to a reasonable extent.

III. Formal Specification

In this section, we will prepare the specifications for the system.

A. Assumptions

There are assumptions that we will need to make about the system before we begin designing it. Below, I have provided a list of assumptions that help us to define our problem:

- There are four rooms in the house and two heaters, as recommended in the original specification.
- All temperature values are represented in degrees Celsius.
- Heaters can be moved dynamically from one room to another based on the temperatures of the rooms. In real life, this may be difficult, but you can think of this assumption as being able to turn heaters on and off in different rooms based on a set of rules.
- The ideal temperature for each room is between 15 and 20 degrees Celsius.
- The arrangement of rooms in the house is general and defined by a symmetric matrix $A \in \mathbb{R}^{4x4}$. The index $A[i][j]$ represents a measure of distance between rooms i and j; this is important because the temperature in one room will affect the temperature in another room more if the two rooms are close. In other words, the rooms can be arranged as a single row, a square, or any other design, the matrix A just needs to be updated accordingly.
- For this problem, we will only consider the heating situation, although the cooling portion is similar.

There are more detailed assumptions to make about the movement of heaters between rooms. We will cover this in even more detail later in this section, but from a high level:

- Only one heater can be moved between rooms in a given time step. In other words, when a heater is moved the system must immediately update the temperatures once the heater has moved.
- If a single heater can be moved to two different rooms or if two heaters can be moved to the same room, simply pick the heater with the lowest index $i \in \{1, 2, 3, 4\}$.

B. Input and Output Interfaces

To define the interfaces, we need to consider both the multi-room system and the thermostat controllers. These two components will communicate with each other through time to update the locations of heaters and the room temperatures. Figure 1 shows how exactly this works.

The inputs to the thermostat controller are the room temperatures x. Since there are four rooms, $x \in \mathbb{R}^4$, where each x_i represents the temperature in room i. This controller will then decide whether a heater needs to be moved in that time step based on control logic that will be discussed in the next section. There are two outputs from the thermostat controller. The first output is $p \in \{0, 1\}^4$, where $p_i = 1$ indicates that there is a heater in room i. The second output is $h \in \{0, 1\}^4$,

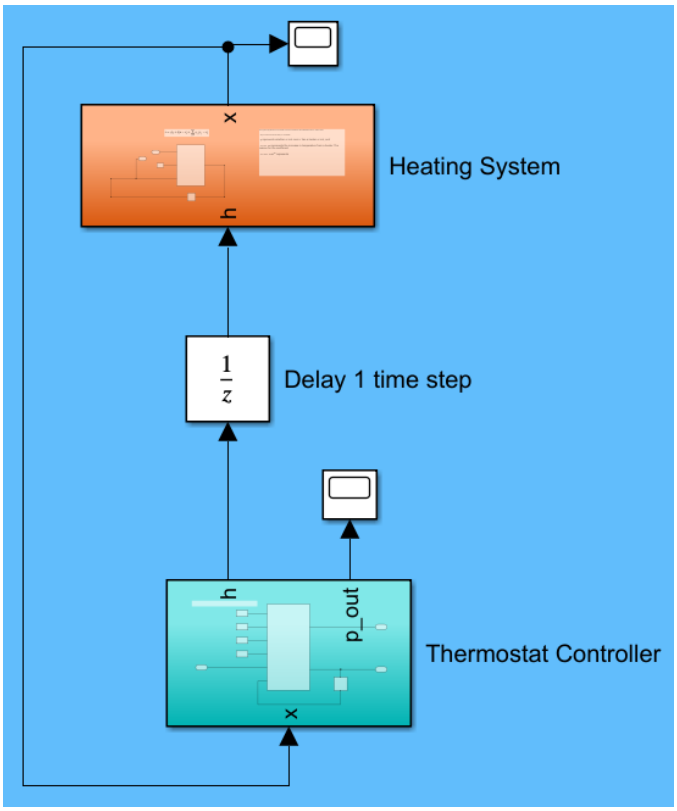

Fig. 1. The input/output interfaces for our system.

where $h_i = 1$ indicates that there is a heater in room i and that the heater is on. The reason for the two outputs is that we need to monitor the placement of the heaters (given by the variable p), but we also need to send h to the heating system since the dynamics depend on it. Rules for determining whether the heater is on or not are discussed in the next section.

For the heating system, the sole input is the vector h. Given this input, the system will use physics to compute and output the updated temperatures in the vector x. It will then pass x back into the thermostat controller for the next step.

Finally, there is a delay block in between the two interfaces. If there was no delay block, the await dependencies between the inputs and outputs of the interfaces would not work; the input of the heating system would require the output of the thermostat controller, and vice versa. The delay block covers this issue. The small white blocks that were not discussed are called scope blocks, and they are used for visualizing the outputs of the different variables (the temperature and placement of heaters).

C. System Dynamics

Physics is used to determine the dynamics of the heating system. The foundation for the physical rules involved in this system originate from Newton's heating law [4]. A simple

form of this equation is written in Equation 1.

$$\frac{dT}{dt} = -k(T - T_O) \tag{1}$$

This equation will serve as the basis for our dynamics. Remember that the goal for the dynamics is to properly update the temperature of each room. In this system, we consider three factors in the changing of the temperatures.

1) If a heater is on in a room, the temperature will obviously increase. The increase in temperature could depend on a few factors, such as the insulation in the room, its size, or its shape. Let's encapsulate those variations per room with a vector $c \in \mathbb{R}^4$, where c_i represents the scale factor for the same amount of heat in each room. Given this vector, Equation 2 represents the increase in temperature due to the presence of a heater.

$$\delta_{1_i} = c_i h_i \tag{2}$$

2) The ambient temperature outside will naturally change the temperatures (this is why we need the heater). Since we are only considering the heating situation, we assume that the outdoor temperature is lower than the indoor temperature. Similarly to the last point, the amount that the outdoor temperature cools a room is dependent on multiple factors, such as the size of the room, the location of the room within the building, and the insulation in the walls. We will again encapsulate those variations in a vector $b \in \mathbb{R}^4$. Remembering (1), we can write the decrease in temperature due to the cooler temperature outside in Equation 3.

$$\delta_{2_i} = b_i(u - x_i) \tag{3}$$

3) The last thing to consider for the dynamics is the temperatures of the other rooms, which will affect the temperatures of each room in a similar way to the ambient temperature outside. Some rooms will affect others more and less, based on factors such as size and distance between each other relative to the house; we will encapsulate these variations in a matrix $A \in \mathbb{R}^{4x4}$, where $a_{i,j}$ represents a measure of proximity between two rooms which will affect the temperature change. Note that this matrix can be thought of as a way to represent the structure of the house or building in our system. Equation 4 mathematically describes this component.

$$\delta_{3_i} = \sum_{j \neq i} a_{i,j}(x_j - x_i) \tag{4}$$

The overall dynamics is the sum of those three terms, written formally in (5).

$$\dot{x_i} = \delta_{1_i} + \delta_{2_i} + \delta_{3_i} \tag{5}$$

This update is performed for every index i. Figure 2 shows a visual representation of these dynamics. As discussed, the inputs to this subsystem include the h vector and u, the outside temperature. The MATLAB function computes the update for the temperatures (Section 4) and outputs the new temperature values for each room in the vector x.

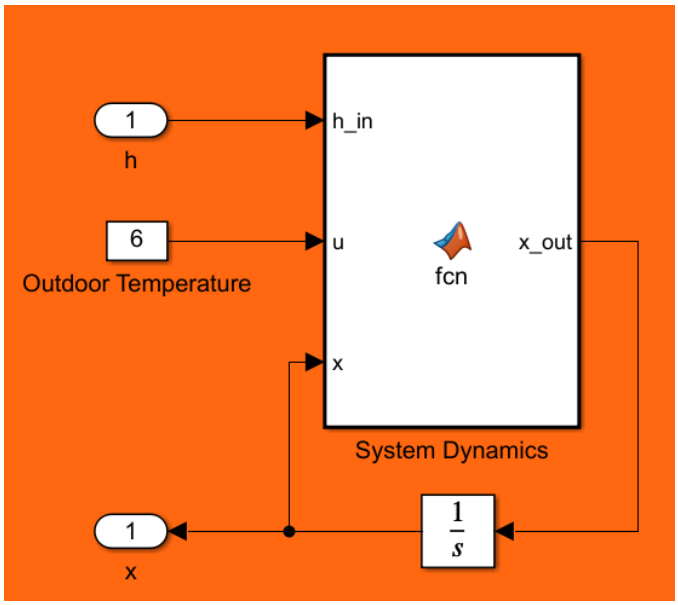

Fig. 2. The heating system dynamics in a visual way.

D. Control Logic

The other main component from our interface is the thermostat controller. This component is responsible for updating the locations of the different heaters. To describe this component, we will begin with the visual specification, which can be found in Figure 3.

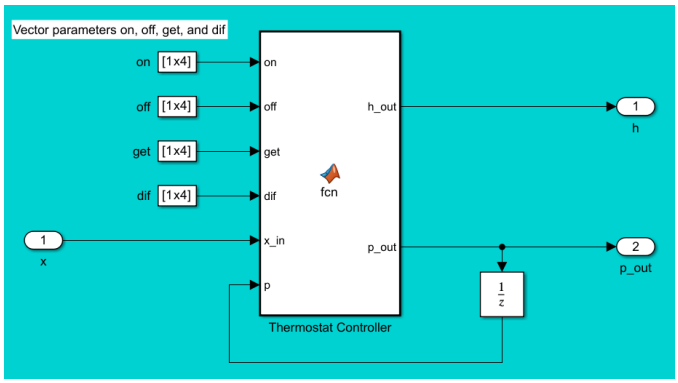

Fig. 3. Control logic for the thermostats.

This controller is parameterized by four vectors: on, off, get, dif $\in \mathbb{R}^4$. The reason for this is the rules for moving the heaters around that were defined in the specification. Namely, a heater can be moved from room j to room i if the following conditions hold:

- Room i does not have a heater

- Room j does have a heater
- $i \neq j$. In other words, the heater cannot be "moved" to the same room that it is already in.
- $x_i \leq \text{get}_i$. In other words, the temperature of the room that receives the heater must be low enough to warrant receiving it.
- $x_j - x_i \geq \text{dif}_i$. In other words, room j has to have a high enough temperature (or, room i has to have a low enough temperature) to warrant moving the heater between them. If the temperature in room j is low, then it should not give up its heater.

Given these parameters, the MATLAB function in the thermostat controller logic will compute the new locations for the heaters and store them in the vector p. Then, the function will compute the vector h, which is required for the system dynamics. The difference between the vectors p and h is that the vector p simply indicates the presence of a heater, while vector h indicates if the heater is on. The heater is on (e.g., $h_i = 1$) for a room i if the following conditions hold:

1) $p_i = 1$. In other words, room i contains a heater.
2) $x_i \leq \text{on}_i$. In other words, the temperature is low enough such that the room needs to be heated.

The final thing to discuss in this section is how we choose the heater that is moved. In some cases, multiple options exist for moving the heater between two rooms. In these cases, we will just pick the room i with the lowest index to receive the heater. In other words, I checked if there was an available move given the rules described above until one was found, and when I found an available move, I moved the heater to that room and moved on to the next iteration. Remember that only one heater can be moved at a time.

E. Safety Conditions

There are some safety concerns with this system. First, we are making lots of assumptions (such as the thermostat not exploding or avoiding random bit flips from nearby magnetic fields). In terms of the system that we have control over, however, there is one main safety concern: that the rooms will be either too hot or too cold. In other words, we want to have this as an invariant of our system. We can write this property in Linear Temporal Logic (LTL) as shown in Equation 6.

$$\textbf{HeaterSafety} : \Box(L \leq x_i \leq U : i = 1, 2, 3, 4) \tag{6}$$

In later sections, we will discuss if this property actually holds true.

F. Liveness Conditions

In class, we discussed that liveness refers to something good eventually happening with our system. I could write down a liveness requirement for the overall system, where eventually the temperatures are all in the specified range. However, this is taken care of in the safety conditions section. Besides, the more interesting liveness requirement includes the context of fairness. A situation we do not want to happen is for a room to be deprived of a heater. If the temperature in a room is too low, we need to provide it with a heater before the inhabitants freeze.

To analyze the fairness requirement, I will explore weak and strong fairness for a task Q. In this case, let Q be the task of moving a heater to any given room i. This implies that $wf(Q)$ represents that if the task Q is enabled at a current step, then in the future, it must either be taken or disabled [1]. We don't want it to be disabled, because then room i could be in a situation where it does not receive a heater. Therefore, the liveness condition that I want to emphasize is the strong fairness condition for task Q. Remember, $sf(Q)$ represents that if the task is repeatedly enabled, then it must be repeatedly executed. For us, this is important because the temperature in a given room will keep going down if there is no heater provided. This requirement can be written formally in Equation 7.

$$sf(Q) : \Box\Diamond Guard(Q) \rightarrow \Box\Diamond(taken = Q) \tag{7}$$

In plain English, every room will always be eventually provided with a heater. The properties of our system that we want to convey are represented by the strong fairness assumption.

G. Verification

There are automated tools for verification. However, I did not use any of these in this project. This could be a piece of future work, discussed in Section 7; however, as we will see in Section 5, the simulation results provide a decent idea of whether or not the LTL properties mentioned in this section were satisfied by the implementation.

IV. Implementation

The source code for this project can be found on my GitHub page[1].

The multi-room heating system was implemented using MATLAB and Simulink [3], [7]. The room heating system dynamics as well as the thermostat controller were designed as subsystem blocks in Simulink. The control logic throughout the system was implemented with a combination of graphical tools from Simulink as well as MATLAB code. Simulations were run and visualized using the simulation window in Simulink. All implementations and experiments were done on the Windows 11 Pro operating system, using MATLAB version r2022b (the current latest version shown on the MATLAB website).

V. Simulation Results

In this section, we will discuss the simulations. First, we will look at the temperature values of each room under different conditions. Next, we will analyze the placement of the heaters as well as whether they are on or off in different times. Finally, we will change some parameters of the controller and discuss how that affects each section.

[1] https://github.com/bameroncaird/Room-Heating-System

A. Temperature Values Over Time

I started by experimenting with the default parameters that were provided in the specification. First, we will take a look at the temperature values over time for a single room. These results can be found in Figure 4.

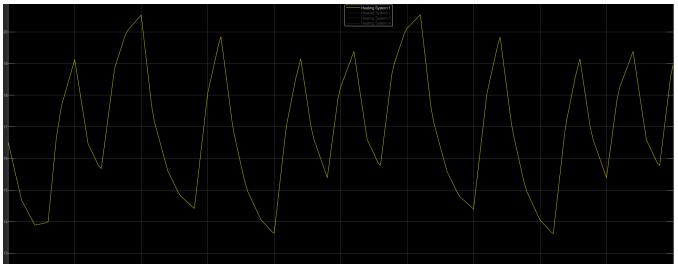

Fig. 4. Temperature values over time for a single room in the building.

This figure is reasonable; there are moments where the room is heating and moments where it loses its heater, resulting in a gradual decrease in temperature. However, let's look at the safety requirement in (6). In the specifications, we set $L = 15$ and $U = 20$. It is hard to see the axes in this single-column paper format, but the minimum temperature value in this result is slightly below 14 and the maximum value is slightly above 20. We can see that the safety requirement was not satisfied; therefore, we can also know that the requirement is not an invariant, since we found a counter-example.

Now, let's look at all of the rooms together to see if it looks similar. Figure 5 shows this result.

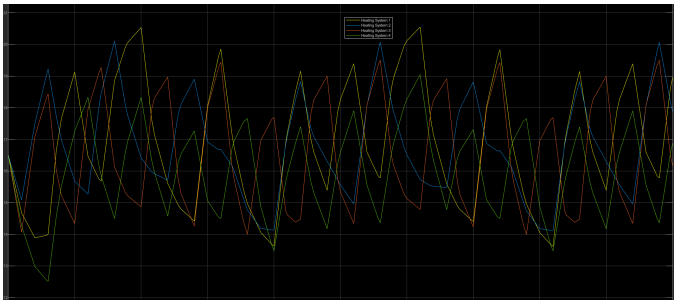

Fig. 5. Temperature values over time for all rooms in the building.

The general output, again, seems to make sense. The values are still not in the correct range that we specified in the safety requirements. However, the fairness requirement seems to be satisfied on visual inspection, as every color has moments of temperature increase where the heater is active and in that room.

B. Heater Movement

Next, we want to analyze the movement of the heaters across time. In a similar manner to last section, we will first look at one and then all. Figure 6 shows the initial result with the heater movement for p_1, e.g., whether or not room 1 has a heater in it as time goes along.

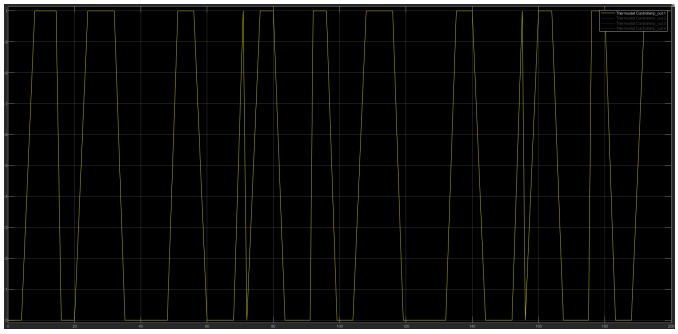

Fig. 6. Heater placement for the first room in the building.

This figure somewhat makes sense. We see the sharp vertical lines due to the heater being moved at an infinitely fast rate between rooms on every time step. However, some of the lines do not make sense. At some time steps, the value of $p_1 \notin \{0, 1\}$. I was not able to figure out why this was the case. Upon printing out the variables in the MATLAB code, I was able to see that they were in fact 0 or 1 at every time step, but the display did not reflect that. My idea for why this happened is the internals of the "scope" block in Simulink, not the values of p being non-binary. However, the important thing to take from this figure is that the heater is in fact being moved between rooms, satisfying the fairness requirement from (7).

Let's scale up and see the heater placement for all four rooms in Figure 7.

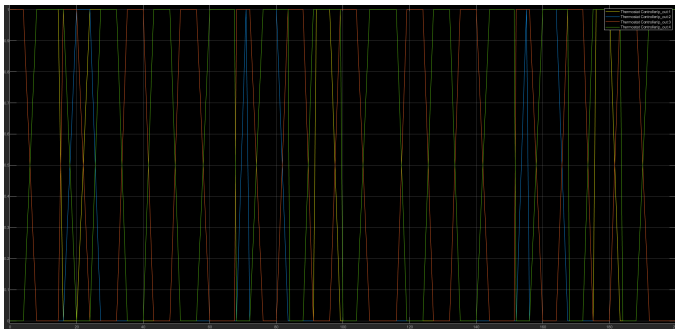

Fig. 7. Heater placement for all rooms over time.

Every room bounces between having a heater and not, satisfying the fairness requirement in terms of this simulation. However, the slopes are still not infinite during the switch; as I mentioned previously, I was not able to discern exactly why this was the case, but I think it has something to do with the scope block.

Finally, let's look at the h vectors over time. The subtle difference between these vectors and the p vectors is that the h vectors indicate if the heaters are actually on. Figure 8 shows

this result. For this part, I will only plot h_2. The other vectors appear in an analogous form.

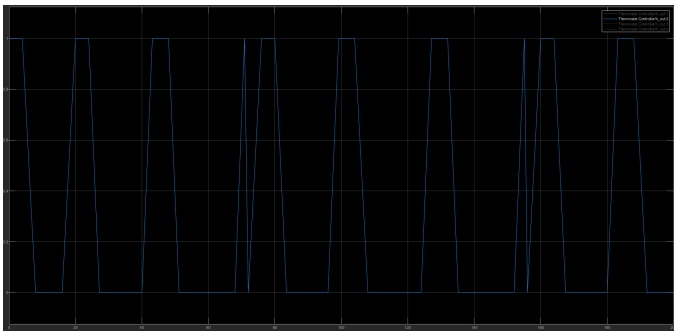

Fig. 8. Heater on/off visualization for room 2.

As we can see, there are more valleys in this figure than the previous 2. The reason for this makes perfect sense: because $h_i = 1$ also requires the heater to be on, this value will be 0 more often than the previous $p_i = 1$ figures, although they do look similar. Note that this figure still has the same slope issue as the last ones did, which again, I did not figure out why this happened.

C. Tweaking Parameters

I will not provide figures for this section because they look similar to the last one. However, changing these parameters changes the output in reasonable ways. For example, reducing the "on" vector in the thermostat controller makes the heaters turn on at lower temperatures, so the controller gets closer to not overshooting the upper temperature bound.

I was not able to keep the temperature of each room between 15 and 20 degrees with any combination of the parameters on, get, and diff while leaving everything else untouched from the specification. I believe that this could be accomplished with a smarter way of updating the heaters; see the Future Work section for more details.

VI. Related Work

There is not much of a related work section here. This is because I mainly used Dr. Abhishek Dubey's specification for this problem. This file can be found in the project GitHub repository[2], the same one mentioned in Section 4.

However, there is still some related work that should be mentioned. The MATLAB Onramp[3] and the Simulink Onramp[4] were useful tutorials for the implementation. Additionally, I gained basic ideas and intuition from more complicated literature on multi-room heating systems [2], [6], a related presentation on simulating HVAC systems in MATLAB and Simulink [8], and a tangentially related paper about simulations of heat pump demands [5].

[2]https://github.com/bameroncaird/Room-Heating-System

[3]https://www.mathworks.com/learn/tutorials/matlab-onramp.html

[4]https://www.mathworks.com/learn/tutorials/simulink-onramp.html

VII. Conclusion and Future Work

In conclusion, this report presented a multi-room heating system with a formal specification and implementation using MATLAB and Simulink. First, I provided some basic intuition about the problem, after which I formally defined the system. Finally, I presented, discussed, and analyzed the results of the software simulations from the Simulink model.

There are many improvements that could be made on this work. First, different types of controllers with different logic could be implemented. For example, I could try using a PID controller for the thermometer rather than the specification-defined system control. Another thing to try with the thermometer is a smarter way of selecting the room to move the heater to, such as the one with the lowest temperature rather than the somewhat arbitrary lowest index method that I used in this implementation. Artificial intelligence based controllers would also be interesting to look at for the thermometer control. Additionally, we could various optimization methods for finding the best locations of heaters; mixed-integer programming (MIP) comes to mind for this scenario. Finally, we could try to make this system more general, with more than four rooms and two heaters (maybe N rooms and M heaters). Overall, there are many tweaks that could be attempted to improve this system if more time was available.

Acknowledgment

I would like to acknowledge Dr. Abhishek Dubey of Vanderbilt University, who provided a basic template and specification for this report as a part of the "Foundations of Hybrid and Embedded Systems" course.

References

[1] Rajeev Alur. *Principles of Cyber-Physical Systems*. The MIT Press, 2015.

[2] Ali Behravan, Roman Obermaisser, and Amirbahador Nasari. Thermal dynamic modeling and simulation of a heating system for a multi-zone office building equipped with demand controlled ventilation using matlab/simulink. In *2017 International Conference on Circuits, System and Simulation (ICCSS)*, pages 103–108, 2017.

[3] Simulink Documentation. Simulation and model-based design, 2020.

[4] Mark Gockenbach and Kristin Schmidtke. Newton's law of heating and the heat equation. *Involve, A Journal of Mathematics*, 2:419–437, 10 2009.

[5] R.C. Johnson, M. Royapoor, and M. Mayfield. A multi-zone, fast solving, rapidly reconfigurable building and electrified heating system model for generation of control dependent heat pump power demand profiles. *Applied Energy*, 304:117663, 2021.

[6] Ciprian Lapusan, Radu Balan, Olimpiu Hancu, and Alin Plesa. Development of a multi-room building thermodynamic model using simscape library. *Energy Procedia*, 85:320–328, 2016. EENVIRO-YRC 2015 - Bucharest.

[7] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[8] Fabian Ochs, Dietmar Siegele, Georgios Dermentzis, Eleonora Leonardi, Toni Calabrese, and Felix Bartagnoli. Building and hvac simulation in matlab/simulink. URL: https://www.matlabexpo.com/content/dam/mathworks/mathworks-dot-com/images/events/matlabexpo/de/2017/gebaude-und-anlagensimulation-mit-matlab-und-simulink-am-beispiel-des-ffg-projekts-saluh.pdf, 6 2017.