EXTREMUM SEEKING CONTROL OF BATTERY POWERED VAPOR COMPRESSION SYSTEMS FOR VEHICLES

BY

SUNNY SHARMA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Mechanical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Adviser:

Professor Andrew Alleyne

Abstract

This thesis investigates the real-time energy optimization of battery powered vapor compression systems (VCS) for vehicles. Battery powered VCS are critical for maintaining passenger comfort in engine-off situations, and are especially important to long-haul truck drivers who sleep inside their vehicle overnight. However, one drawback of battery powered vehicle VCS is their short lifespan which may not provide cooling through the whole night while the vehicle engine is turned off. One reason for short system lifespan is suboptimal input selection; the combination of inputs to the VCS often yields a power consumption higher than necessary to generate the required vehicle cooling. This thesis proposes the use of extremum seeking control (ESC), a class of real-time model-free optimization algorithms, to determine the optimal combination of system inputs that minimizes the VCS power consumption while meeting given objectives. In order to determine algorithm efficacy, we implemented three different ESC algorithms (perturbation-ESC, least squares-ESC and recursive least squares-ESC) on a simulated and physical integrated VCS (the VCS in conjunction with the battery pack and vehicle cabin). Simulation and experimental results demonstrate significant increases in energy efficiency and battery life through the use of these algorithms, with least squares-ESC and recursive least squares-ESC being the most effective of the three.

To my family, friends, and teachers.

Acknowledgements

First and foremost, I would like to thank my adviser Dr. Andrew Alleyne for his immense support and guidance over the past two years. Rarely do you find an adviser who is entirely committed to his or her students' success above all else. I have learned so much from Dr. Alleyne and I am fortunate to have had an adviser so intelligent, patient, caring and devoted to helping me achieve my life and career goals.

I would like to thank Aaron Sullivan from Bergstrom Inc. for his help in understanding and troubleshooting the experimental system used in this thesis. Aaron was always willing to take time out of his busy schedule to patiently answer my questions, for which I am immensely grateful. This thesis could not have been completed without his help, and I aspire to be a mentor like Aaron when I join the work force.

I would be remiss to not acknowledge my lab mates, past and present, in the Alleyne Research Group. You all have made my graduate school experience special. I will always remember the memories we shared together and I am honored to call you my friends. To the senior lab members, Justin, Herschel, Bryan, and Matt, thank you for the mentorship and guidance you provided me. Thanks to Malia, Oyuna, Ashley, Spencer K., Nate, and Pamela, the labmates one year above me, for making the transition to graduate school as smooth and as fun as possible. Thanks to Spencer I. and Sarah, my fellow members of S^3, for all of the fun memories we made and for pretending like we were still in undergrad. Thanks to Chris and Donald, the future of ARG, for being such great friends and lunch buddies. A special thank you to Sarah for all of her support over the past two years.

Last, but not least, a big thank you to my family. My parents are my biggest inspiration, and I would (literally) not be here without you. You raised me to be hardworking, honest, empathetic and humble, and I hope I have made you proud. To my sister, thank you for always being my biggest supporter. You give me the confidence to believe in myself and always push me to be the best that I can be.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This thesis explores the use of model-free real time optimization strategies to improve the energy efficiency of battery powered vehicle VCS (vapor compression systems). These systems, critical to maintaining passenger comfort, can be highly inefficient and their short battery lives present a significant barrier to mainstream adoption. The combinations of inputs to the system are often energy suboptimal; a different set of input combinations could potentially meet the same performance requirements at a reduced power consumption. One common method used to determine energy optimal inputs is to develop a model of the system to estimate these values. However, VCS have complex dynamics that can be difficult to replicate, and the system behavior may even change over time due to environmental effects. Optimization methods such as extremum seeking control (ESC) can determine optimal inputs without explicit system knowledge. Instead, this approach generates an estimate of the steady-state cost function gradient and uses that to drive the system to its most efficient operating point. However, these techniques are often slow and complicated to implement, which can limit the use of these algorithms to industry experts and academics. This thesis investigates the development and use of intuitive and fast-performing ESC algorithms to improve the energy efficiency of battery powered vehicle VCS.

1.1 Motivation

Vehicle VCS are a ubiquitous staple of modern life, providing a comfortable driving experience regardless of the conditions outside. These systems are especially important to

long-haul truck drivers, who not only work long hours within the confines of their vehicle, but also sleep and spend much of their free time inside the vehicle cabin. Truck drivers have traditionally resorted to idling their vehicles in order to provide cooling or heating while not driving. However, vehicle idling is extremely energy inefficient, expensive and is a significant source of greenhouse gas emissions. One organization estimates the annual fuel cost of idling to be $3 billion dollars – the equivalent of burning 1,800 gallons of diesel [1]. Not surprisingly, around half of the states in the U.S. have some sort of law against idling – with more states soon to join this trend.

Figure 1.1 States with laws on no idling [1].

A more efficient and cost effective alternative is to use a no-idle VCS, which is a battery powered air conditioning system that is charged while the vehicle drives and is commonly used to provide a comfortable sleeping environment while the truck is stopped for the night. However, one significant issue limiting the potential of no-idle systems is their relatively short battery life. In some cases, these systems only operate up to 6 hours before needing a recharge. This is an unacceptable to industry practitioners, who need a minimum of 8 hours of comfort while sleeping. One solution to this issue is to develop larger battery banks to meet these performance requirements. However, this does not address the underlying energy inefficiency. To do so, we need to take a closer look at system behavior and performance.

Modern VCS have a number of adjustable inputs, including the compressor speed, the evaporator and condenser fan speeds, and the expansion valve opening. A combination of these inputs generates corresponding outputs, the most important being the cooling capacity – or the amount of heat absorbed by the system, and the power consumption, which is the sum of the power consumed by the compressor and evaporator/condenser fans. Air conditioning literature has shown that a number of input combinations can yield the same cooling capacity; however, only one unique set of inputs will yield the same cooling capacity while consuming the least amount of power [2]. The question is – how can we determine what the optimal set of inputs are?

One method of determining these inputs is by developing a dynamic model of the system. System models can give significant insight into the system behavior and can help estimate the optimal input values. However, precise modeling of VCS is extremely difficult: its dynamics are often highly complex, and the system behavior may even change over time due to equipment aging and environmental effects such as corrosion and fouling. Physics based modeling approaches – using dynamic equations to describe system behavior often require significant assumptions and simplifications in order to produce tractable solutions. Furthermore, these models often require users to tune parameter adjustment factors to match system behavior precisely. Black box system-identification methods, on the other hand, estimate system dynamics based on a range of given input and output data. However, this approach may only yield accurate results over a small range, and furthermore does not take into account changing system behavior over time. Thus, it is highly desirable to develop model-free optimization schemes that can select the optimal system inputs in real time without prior knowledge of system behavior.

1.2 Introduction to Extremum Seeking Control (ESC)

As stated earlier, ESC is one of the most popular model-free optimization approaches used to improve system performance. This optimization method works by deriving an estimate of the system's steady-state performance function gradient from the system's input and output signals and uses it to drive the system inputs to values that minimize the system's steady state

performance function (note that for proper minimization, performance function convexity with respect to inputs is a necessary condition). ESC can thus determine the system's optimal inputs without having explicit knowledge of the system itself. However, this lack of knowledge comes at a cost: convergence to these optimal inputs is an inherently slow process. ESC traces its development to a paper written by LeBlanc in 1922 [3], but it was only until the early 21st century that ESC became a major field of research when Krstic and Wang published the first formal stability proof of ESC in 2000 [4]. ESC has since been utilized in a wide range of applications, from maximizing photovoltaic power point tracking [5], to increasing biomass production in reactions [6] to improving the energy capture of wind turbine systems [7]. In particular, there is a great deal of literature demonstrating ESC effectiveness in optimizing VCS performance due to the convex relationship between inputs and power consumption. A wide variety of ESC controllers, from perturbation based ESC [2] to time varying ESC [9], to least-squares based ESC [10] have demonstrated increases in energy efficiency while meeting given objectives. However, ESC has not yet been utilized to demonstrate battery life extension in battery-operated vehicle VCS, which is the main focus of this thesis.

1.3 Organization of Thesis

This thesis is organized as follows. Chapter 2 introduces the vapor-compression cycle, VCS design and the modeling of VCS components in Simulink. The thermal modeling and validation of a vehicle cabin model is also discussed, as well as the integration of the cabin model with the VCS model and common closed-loop temperature regulation strategies. Chapter 3 explains the mathematical basis behind perturbation and advanced ESC algorithms, as well as the specific applications to VCS. Chapter 4 discusses the implementation and results from implementing three different ESC algorithms on the simulated integrated system. Chapter 5 details the development of the experimental integrated setup and ESC implementation on this system. The thesis concludes in Chapter 6 with a summary of research contributions and opportunities for future work.

Chapter 2
Modeling, Operation and Control of Integrated Vapor Compression Systems

Before investigating the use of performance improving algorithms, we first need to understand the nature of the system being optimized and also understand common control strategies used on such systems. This chapter first introduces the standard four-component vapor-compression system. Next, we discuss the modeling and validation of an integrated no-idle vehicle VCS: that is, the modeling of a VCS in conjunction with the vehicle cabin and battery pack in MATLAB/Simulink. This thesis provides a detailed, first-principles derivation of the cabin model, as it was developed specifically for this thesis. System model development is important because models can yield significant insight into system behavior and can be used to rapidly test and validate various control schemes, including ESC. Note that although ESC is referred to as a "model-free" control algorithm, this refers to the algorithm itself being model agnostic as opposed to being incompatible with a system model. After discussing the modeling of such systems, we detail the development of a simplified integrated experimental setup. Lastly, we examine the closed-loop control of both the modeled and experimental system and discuss opportunities for optimization.

2.1 Introduction and Review of the Standard Four Component VCS

On their most fundamental level, VCS's are built for the purpose of heating or cooling a specific space. This thesis looks solely at the cooling application of such systems. The VCS cools an area by cycling a refrigerant to absorb heat from one area and then rejecting that heat

to another space. The most commonly used refrigerant is R-134a, but other refrigerants such as CO2 and R-1234yf have also been successfully used in vehicle VCS.

Fig. 2.1 shows a schematic of the most basic VCS configuration along with its respective pressure-enthalpy (P-h) plot. From points 4 to 1, cool refrigerant flows through the evaporator and absorbs heat from the surrounding region. An evaporator blower blows warm air over the evaporator coils, facilitating this process. The refrigerant is compressed to a higher pressure and temperature as it passes through the compressor from points 1 to 2. Warm refrigerant passes through a condenser from points 2 to 3, where it rejects heat to its surroundings with the aid of a condenser fan. From points 3 to 4, the refrigerant passes through an expansion device where it decreases in pressure and temperature, and the cycle starts again.

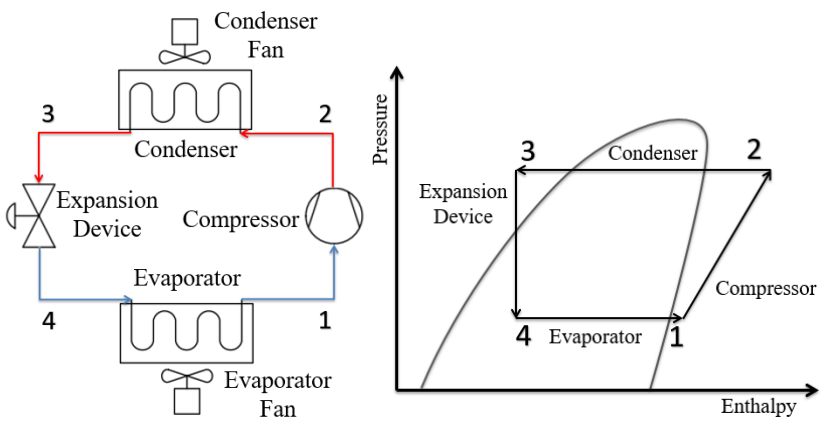

Figure 2.1 The standard four-component VCS along with its respective P-h plot.

As seen in the P-h plot, a number of refrigerant phase changes occur during an ideal cycle. Most importantly, the refrigerant enters the evaporator as a two-phase fluid and leaves as a superheated vapor. This is important because most compressors are not designed to pump liquid refrigerant for long periods of time. The degree of refrigerant superheat at the evaporator outlet is an important quantity because it serves as a buffer from two-phase refrigerant entering the compressor.

Traditionally, the VCS practitioner did not have the ability to adjust many of the system inputs. For example, components such as fans were powered by fixed speed motors. However,

modern VCS's have a number of adjustable inputs that allow the user to tune system performance. The compressor, along with the evaporator and condenser fans are commonly powered by variable speed DC motors, giving the operator the ability to precisely adjust the component speeds. Some expansion devices, such as the electronic expansion valve (EEV), allow the user to control its aperture, thereby controlling the amount of refrigerant that flows through the device. However, EEV's are very expensive, and so this thesis considers the use of a thermostatic expansion valve (TXV) instead. The TXV is a spring-loaded expansion device that adjusts its aperture depending on refrigerant pressure at the evaporator outlet, among other things, in an effort maintain a constant evaporator superheat.

2.2 Modeling of an integrated VCS

Advances in computer modeling allow users to accurately simulate the behavior of complex dynamical systems. This is particularly important because modeling can give insight into system behavior without requiring a physical test-bed to derive meaningful results. Furthermore, modeling also allows the user to simulate scenarios significantly faster than in real-time. In this section, we model the cooling of a vehicle cabin using a battery-powered, no-idle VCS in the MATLAB/Simulink environment. Specifically, this modeling effort attempts to replicate the typical application of the NITE Phoenix SSI (referred to in this thesis as the NITE), a no-idle air conditioning unit developed by Bergstrom Inc., an industry partner of the Air Conditioning and Refrigeration Center (ACRC) at the University of Illinois. The NITE is designed to provide cabin cooling for long-haul truck drivers overnight. The unit features a split condenser/evaporator design, where the condenser mounts outside the vehicle while the evaporator is housed inside the cabin to provide cooling. The unit delivers up to 2.2kW of cooling capacity and runs on a bank of four rechargeable lead-acid batteries with a run time of approximately 8 hours. [11]

Figure 2.2 A picture of the NITE Phoenix SSI [11]. The condenser coil and fan are housed in one unit (left), while all other components are housed in another (right). The units are connected together using long, flexible refrigerant tubes.

There are four main sections of the integrated system: the four component VCS, the vehicle cabin, the battery pack and the evaporator blower/condenser fan. Each section is described in detail below. The integrated model demonstrates the ability of the VCS to cool the vehicle cabin and generate a power draw from a bank of lead-acid batteries. Set point commands determine nominal operating values for the compressor, condenser fan and evaporator blower. Fig. 2.3 depicts the integrated model in the MATLAB/Simulink environment.

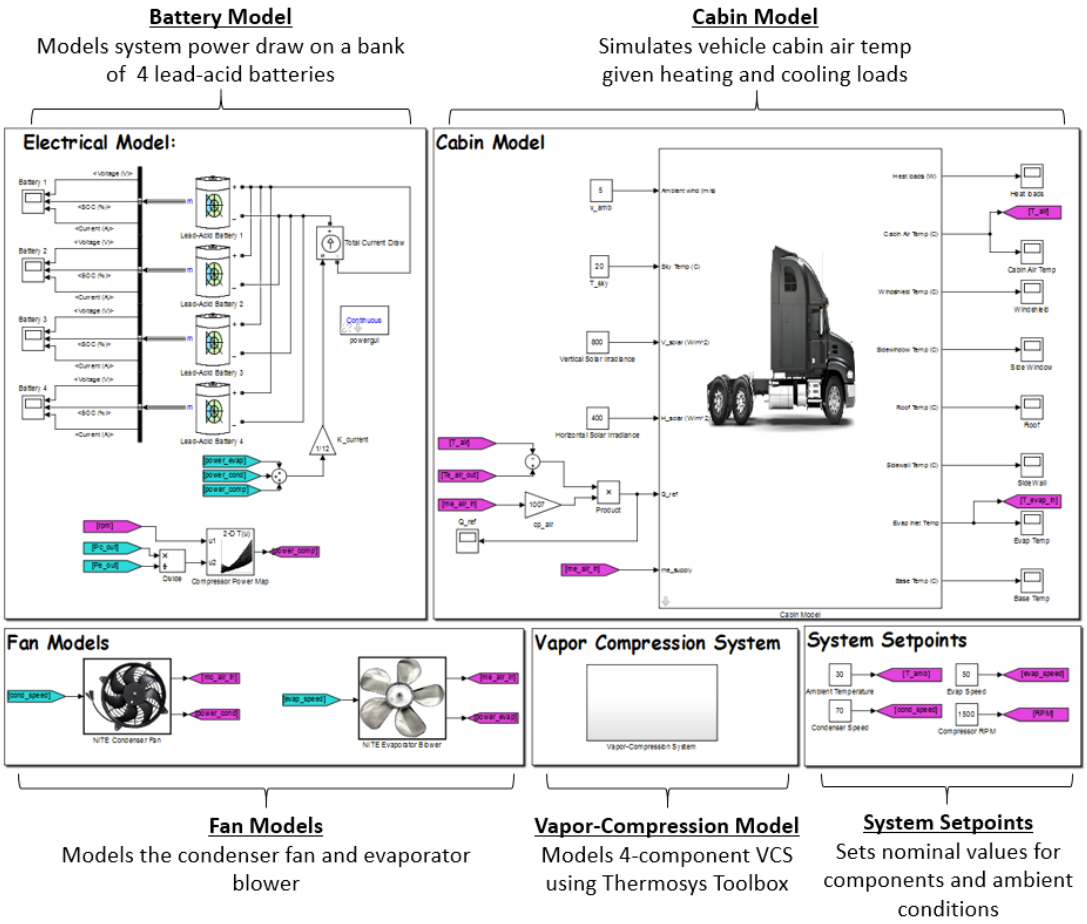

Figure 2.3 A model of the integrated VCS developed in MATLAB/Simulink. The user determines set point values for the compressor, condenser fan and evaporator blower and sets the ambient conditions. Given these inputs, the vapor compression system generates a corresponding cooling capacity, absorbing heat from the vehicle cabin.

2.2.1 The Vapor-Compression Model

Researchers in the Alleyne Research Group at the University of Illinois have developed Thermosys, a toolbox in MATLAB/Simulink used to simulate the dynamic behavior of vapor-compression systems [12]. Each VCS component (compressor, condenser, etc.) is modeled as an independent block that sends and receives signals such as pressure, mass flow rate and enthalpy. Component blocks are connected together by routing signals accordingly. Fig. 2.4 outlines the standard four-component system developed using Thermosys.

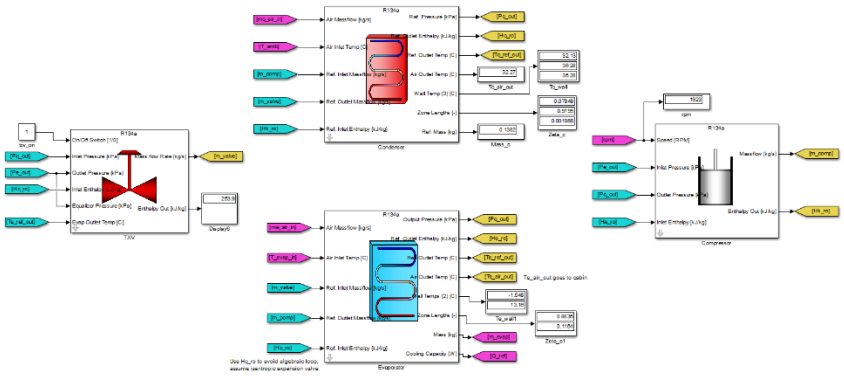

Figure 2.4 A four-cycle VCS consisting of the evaporator, compressor, condenser and TXV constructed using Thermosys, a toolbox in MATLAB/Simulink.

Heat exchangers such as the evaporator and condenser are modeled using a moving-boundary volume approach in conjunction with mass and energy conservation equations, while components such as the compressor and expansion valve are modeled using algebraic equations due to their fast dynamics. Thermosys also contains other auxiliary components such as accumulators and receivers; however, this thesis only examines the use of the four main vapor-compression cycle components. A detailed derivation and analysis of these models can be found in [13].

Each block has a list of tunable parameters that the user provides in order to model the component after a specific device. The heat exchangers allow the user to define the thermal and physical characteristics of the refrigerant tube, as well as the airside and refrigerant side surface areas. Furthermore, adjustable heat transfer coefficient factors help match model outputs with experimental data. The TXV model allows the user to define properties such as the spring pressure and the bulb time constant. The compressor model utilizes an empirical compressor performance map to define the volumetric and adiabatic compressor efficiencies for a given RPM and pressure ratio. This data is then used to calculate the compressor outlet states such as the enthalpy and mass flow rate. In addition, the compressor model allows users to define the compressor volume along with its outlet enthalpy time constant [28].

One of the goals of this thesis was to model the VCS after the NITE system. However, while there were some opportunities to parameterize the component systems, a number of

issues prevented the full parameterization of the VCS model. The NITE's condenser has an advanced microchannel design that is not currently congruent with the model structure in Thermosys, precluding its modeling. Modeling the TXV was not possible due to a lack of data available on its dimensions and performance characteristics. Modeling the NITE's compressor proved highly difficult, especially the compressor's performance map. In order to generate a compressor map, one needs to retrofit a compressor with pressure, temperature and mass flow rate transducers at the inlet and outlet to generate effective estimates of the volumetric and adiabatic compressor efficiencies. However, the physical NITE unit used in this thesis only had input/output temperature transducers along with a low-side pressure transducer, which is insufficient to generate a performance map. Therefore, the compressor model retained its original map. On the other hand, there was sufficient data available to modify most of the evaporator model's parameters. Table 2.1 lists the NITE system parameters used in the Thermosys models. All other parameters are left as default, which are based on a VCS test stand in our laboratory at the University of Illinois.

Table 2.1 NITE Parameters used for VCS modeling

Component	Parameter	Units	Value
Evaporator	Hydraulic Diameter	m	6.33E-03
	Length of One Refrigerant Pass	m	4.09
	Number of Parallel Passes		3
	Air Side Cross Sectional Area	m^2	0.01844
	Air Contact Surface Area	m^2	1.50
	Refrigerant Surface Area	m^2	0.081
	Refrigerant Pass Cross Sectional Area	m^2	3.15E-05
Compressor	Compressor Volume	m^3	7.1E-06

Because most of the model parameters were not from the NITE system, it precludes the ability to determine how well the model can match the behavior of the actual system. The different characteristics between the simulated and experimental systems manifests itself in significant ways, as will be seen in Chapters 4 and 5 when we implement ESC on these systems. Nevertheless, the model can still provide insight into how a VCS such as the NITE would respond to different inputs, changes in environmental conditions, or various control schemes. Furthermore, having models allow users to perform the above much more quickly than on an actual system, expediting control design. Thus, these models are sufficient for the purposes of this thesis effort.

2.2.2 Fan Models

The evaporator blower and condenser fan are used to blow air over the evaporator and condenser coils respectively in order to enhance the heat absorption of the evaporator, and the heat rejection of the condenser. These models were developed to replicate the performance of the NITE's evaporator blower and condenser fan. The reason for using a blower (also known as a centrifugal fan) for the evaporator is that blowers generate an increase in air pressure to overcome pressure drops in air ducts.

Fans and blowers operate similarly, using a DC motor to spin a series of blades that move air from one region to another at a certain velocity. The NITE system in particular allows the user to control the speed of this motor by sending a 0-255 PWM command. Because of the ease of data collection for these components, it is sufficient to model these devices based on experimental performance data. This was done by stepping the evaporator blower and condenser fan at different speeds and recording the corresponding power draw and mass flow rate using an anemometer. The evaporator blower and condenser fan performance data is depicted in Fig. 2.5 and 2.6 respectively.

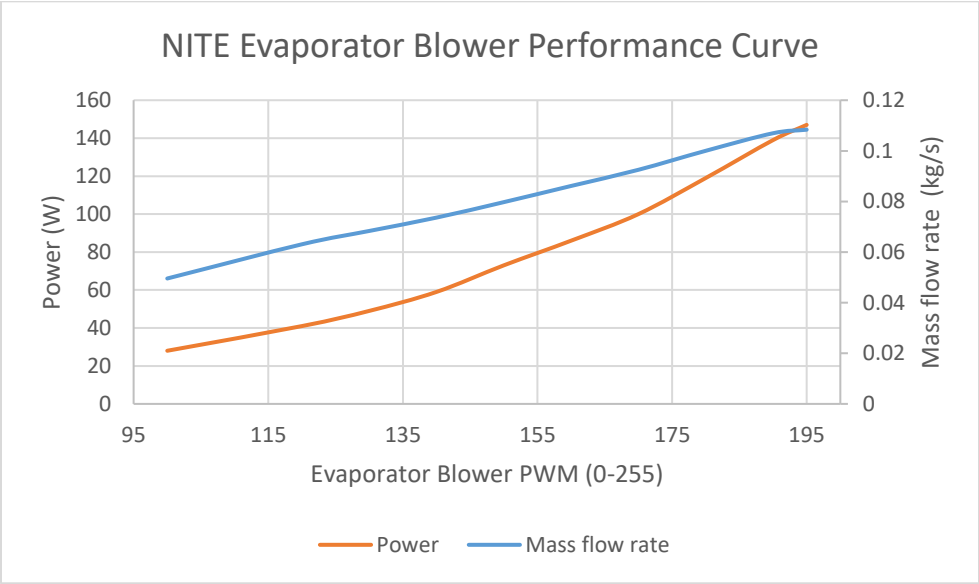

Figure 2.5 The mass flow rate and power draw of the evaporator blower over the nominal range of blower speeds commonly used during operation.

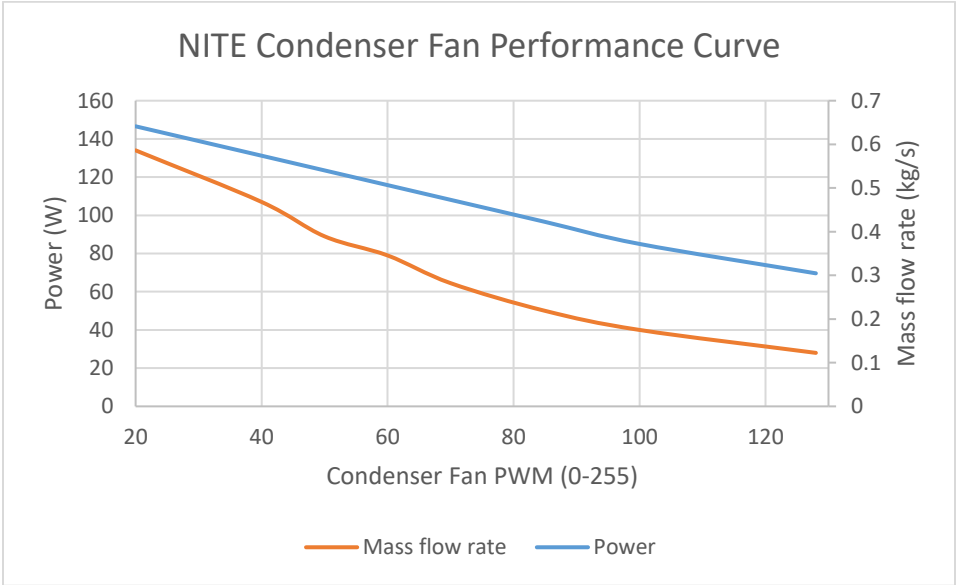

Figure 2.6 The mass flow rate and power draw of the condenser fan over the nominal range of condenser speeds commonly used during operation.

This performance data was used to model these components in Simulink using a MATLAB function block, as shown in Fig. 2.7. These models output an air mass flow rate to their respective heat exchangers and output a power consumption. Mass flow rates are linearly interpolated while power is interpolated using a cubic (spline) function.

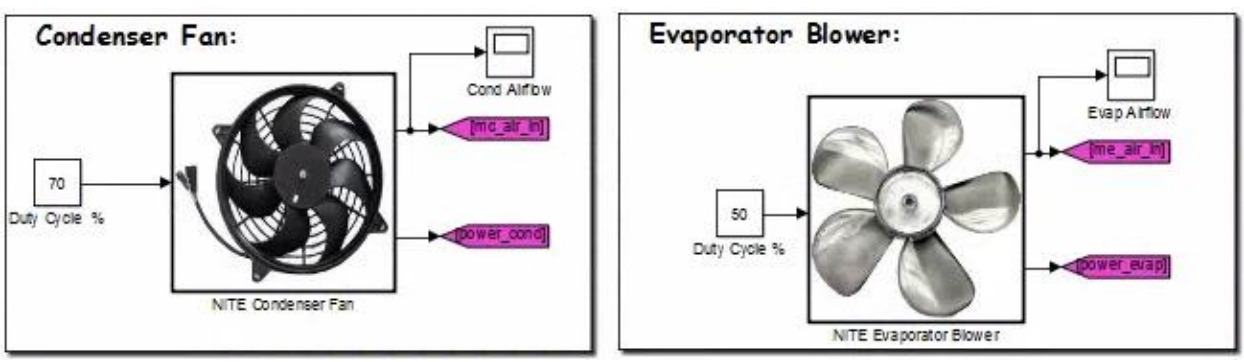

Figure 2.7 The condenser fan and evaporator blower modeled in Simulink.

2.2.3 Electrical Model

The electrical model has two purposes: calculating the total system power consumption and modeling the power draw from the NITE's batteries. The fan models from Fig. 2.16 output a power draw, but the Thermosys compressor model does not by default. Thankfully, [17] generated a power map for the compressor model by collecting experimental power data from the compressor over a range of pressure ratios and RPM's. The bottom of Fig. 2.8 depicts the compressor power map developed in Simulink.

Four 12V lead-acid batteries connected in parallel power the NITE system. Bergstrom recommends the use of the Trojan AGM 31, a deep-cycle lead-acid battery that provides approximately 86 amp-hours each at peak performance [8]. Lead-acid batteries are well-modeled using the battery model in the Simscape toolbox [29], and Fig. 2.8 illustrates the recommended NITE battery configuration. The battery model allows the user to select the type of battery, along with specifying its capacity, voltage and dynamic characteristics. Because there was a lack of empirical data on the battery dynamics, the only user-defined parameters were the voltage, capacity and initial state of charge. Fig. 2.9 shows the battery parameter dialog box for the lead-acid battery. The current draw from the batteries was calculated by dividing the power by 12 since we assume the battery operates at a constant 12V.

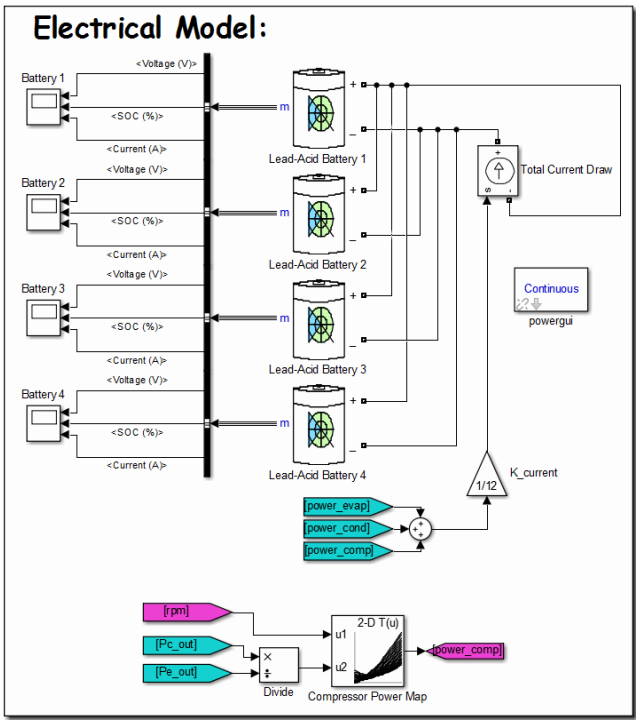

Figure 2.8 The electrical model developed in Simulink. The power consumption of all VCS components is divided by 12 to simulate the current draw on a bank of four lead-acid batteries modeled after the Trojan AGM31 battery. Each battery outputs a voltage, percent state of charge and current. The compressor power map at the bottom of the figure is an empirical map that calculates the compressor power consumption as a function of the RPM and pressure ratio.

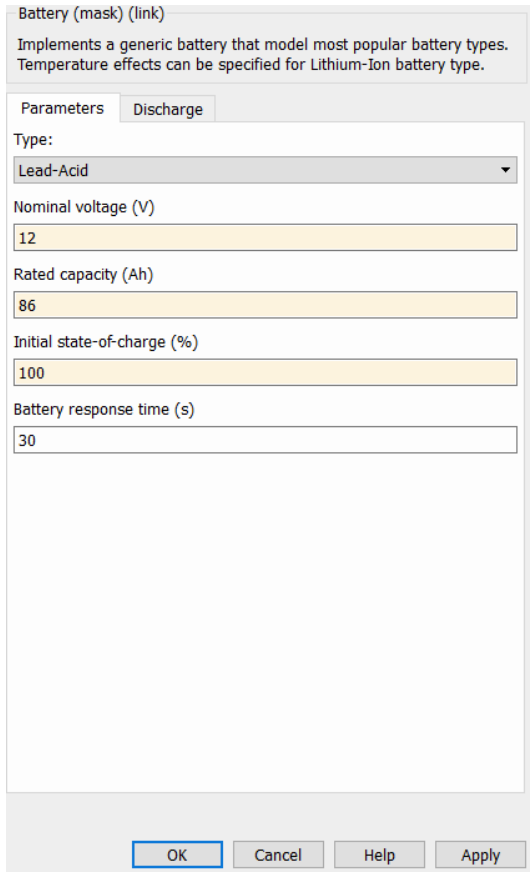

Figure 2.9 The battery parameter dialog box. The type of battery, along with nominal voltage, rated capacity and state of charge were based off the Trojan AGM-31, the recommended battery for no-idle vehicle VCS.

2.2.4 Vehicle Cabin Model

The cabin model was developed as part of this thesis effort, and its derivation will be discussed in detail. The cabin model is designed to simulate the dynamic temperature response of the vehicle cabin air to common external and internal heating and cooling loads, such as heating loads from solar radiation and from cooling loads such as the vehicle's air conditioning system.

2.2.4.1 Literature Review

Dynamic thermal modelling of vehicle cabins has been an area of significant research over the past several years due to increased awareness of thermal comfort issues in vehicles, as well as an increased interest in vehicle efficiency. The same principle also holds true for battery-powered truck VCS's – they must be able to maintain an appropriate level of thermal comfort for vehicle occupants over the operating lifespan. Therefore, there is a clear need to develop a high-accuracy cabin model to determine whether thermal comfort needs are met during operation.

There are two main approaches to cabin modelling in the literature: physics-based modeling and computational thermal modeling. The physics based modelling approach calculates the cabin air temperature using simple, fundamental heat transfer equations that model the heat flow between the cabin and the outside environment. In order to reduce calculation complexity, this approach often involves the use of simplifying assumptions such as using a lumped capacitance approach for calculating cabin air temperature (air has uniform properties). Marcos et al. wrote one of the most cited research papers using this approach in 2014 [14]. The authors of the paper developed a first-principles model of the cabin thermal dynamics in Simulink. The authors considered four main heat loads affecting the cabin air temperature: incoming solar radiation through the windows $(\dot{Q}_{windows})$, conductive heat loads passing through the vehicle's ceiling $(\dot{Q}_{ceiling})$, convective heat transfer between the vehicle's internal surfaces and the cabin air $(\dot{Q}_{base})$ and the heat generation from human bodies $(\dot{Q}_{human})$. The values for these heat loads are calculated at each time step. These heat loads are then used to solve for the new cabin air temperature using the following first order differential equation:

$$m_{air} c_{p,air} \dot{T}_{air} = \dot{Q}_{windows} + \dot{Q}_{ceiling} + \dot{Q}_{base} + \dot{Q}_{human} \tag{2.1}$$

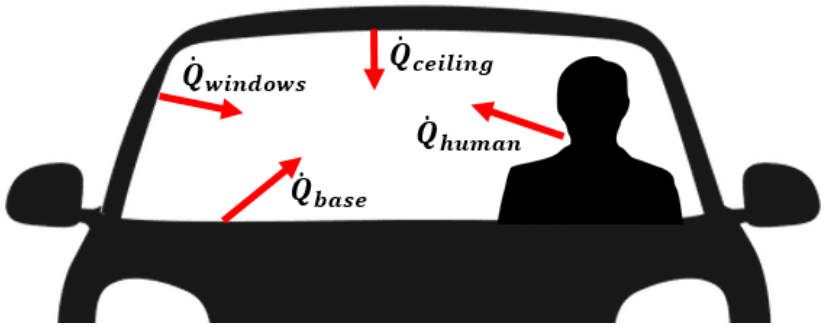

Figure 2.10 A visual representation of the heat loads considered by [14].

On the other hand, computational thermal modeling refers to the use of high-fidelity specialized software that uses numerical methods to calculate the temperature evolution of the vehicle cabin. This method is much more computationally intensive than physics-based modeling but is more accurate as it does not make many simplifying assumptions. These models are often able to capture the spatial variation in temperature inside the vehicle, whereas the first-principles models cannot.

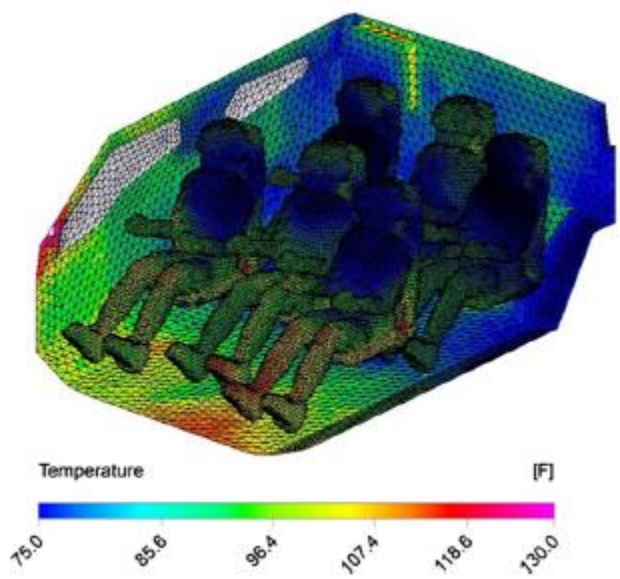

Figure 2.11 A model of the temperature profile inside a vehicle cabin using a numerical CFD approach as seen in [18].

2.2.4.2 Cabin Model Overview

The cabin model developed in this thesis utilizes a physics based modeling approach similar to [14] due to its simplicity and relative accuracy. The cabin model takes in inputs such as ambient temperature, solar radiation and ambient wind velocity and returns the cabin air temperature along with other states such as the temperature of the roof, walls, windshield and side windows, as illustrated in Fig. 2.12 and Table 2.2 below. The cabin model also determines the air temperature at the evaporator inlet depending on the percentage recirculation specified by the user. Parameters such as vehicle dimensions, passenger occupancy and material properties can be adjusted to simulate the use of different types of vehicles and passenger loading scenarios, as seen in Fig. 2.13.

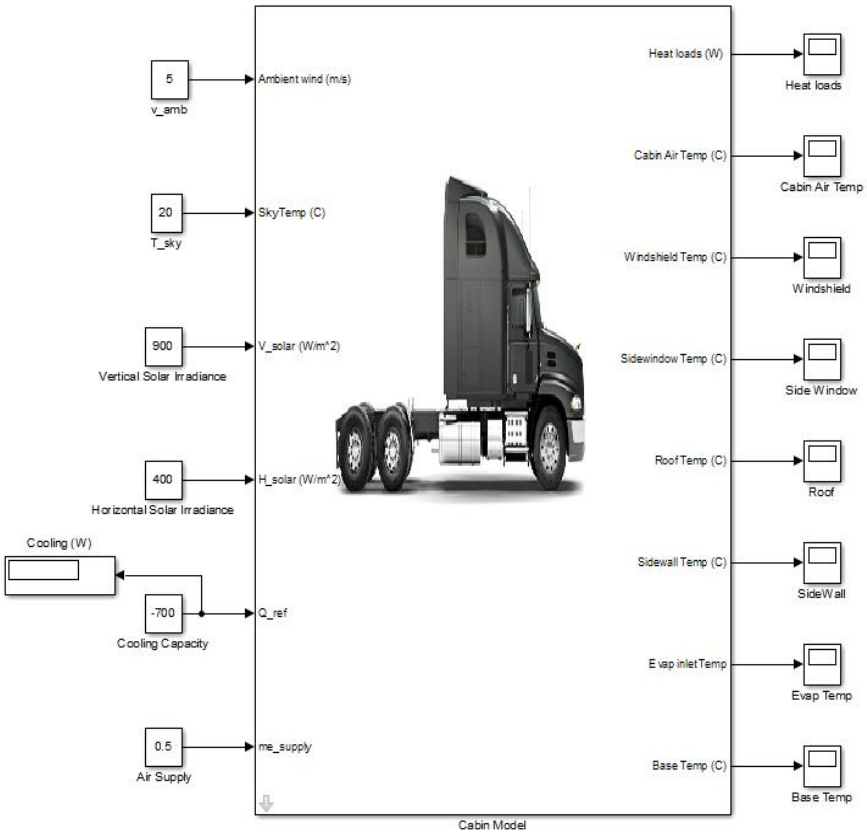

Figure 2.12 The cabin model developed in Simulink. The block accepts inputs such as the ambient conditions and the cooling capacity, and outputs states such as the cabin air temperature and the temperature of the vehicle surfaces.

Table 2.2 Cabin model inputs and outputs

	I/O	Units	Description
Inputs	v_{amb}	m/s	Ambient wind velocity
	T_{sky}	C	The temperature of the clouds, water vapor, and other atmospheric elements that make up the sky to which a surface can radiate heat.
	V_{solar}	W/m^2	Vertical solar irradiance
	H_{solar}	W/m^2	Horizontal solar irradiance
	$\dot{Q}_{ref}$	W	Cooling capacity of the VCS
	$\dot{m}_{ref}$	kg/s	Airflow from the VCS
Outputs	Heat Loads	W	Vector of heat loads on the vehicle cabin
	T_{air}	C	Internal cabin air temperature
	T_{ws}	C	Temperature of the windshield at five different points
	T_{sw}	C	Conductive heat load transmitted through the vehicle roof
	T_{roof}	C	Conductive heat load through the side windows
	T_{wall}	C	Conductive heat load through the side walls
	$T_{evap,i}$	C	Air temperature at the evaporator inlet
	T_{base}	C	Temperature of the vehicle's base

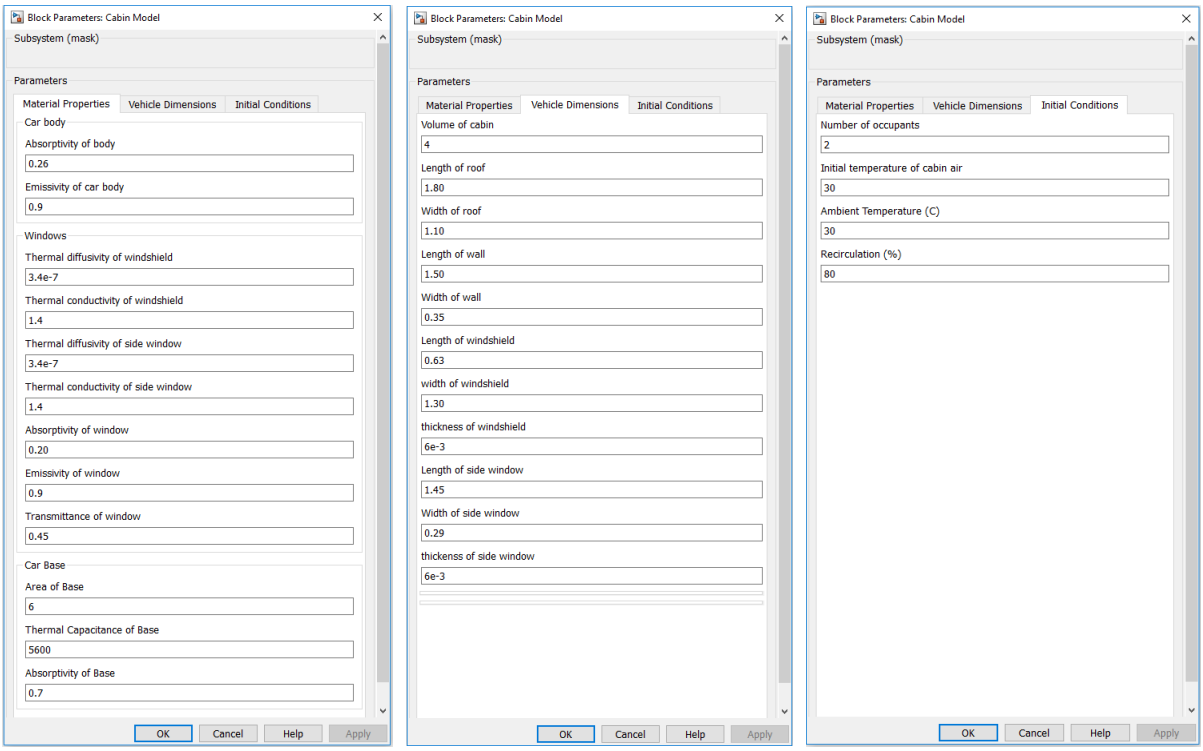

Figure 2.13 Parameter dialog for the cabin model. The user inputs values for the vehicle's material properties, vehicle dimensions as well as the initial conditions.

2.2.4.3 Cabin Model Derivation

In order to simplify derivation and analysis, the following assumptions are made [14]:

1) The air inside the vehicle cabin is considered as a lumped parameter (the air is assumed to have a uniform temperature). Although temperature gradients likely exist, a lumped parameter approach greatly simplifies calculations while maintaining sufficient model validity.

2) No mass accumulation of air occurs inside the cabin. Any air that infiltrates the cabin or enters from the vehicle VCS $(\dot{m}_{ref})$ is matched by an air leakage term $(\dot{m}_{leak})$. This mass balance is illustrated in Fig. 2.14.

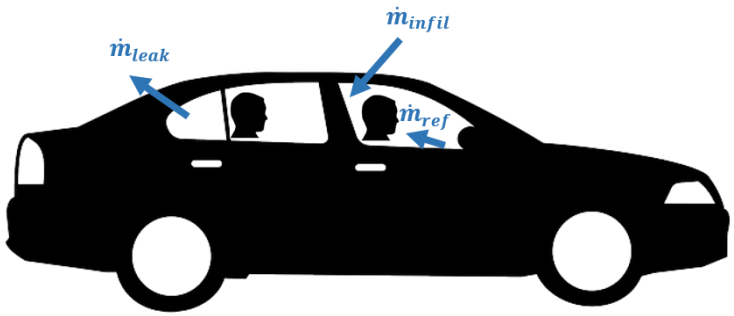

Figure 2.14. An illustration of the mass flow rates entering and leaving the cabin. The sum of the three air mass flow rates is zero.

3) Radiative heat transfer between interior surfaces can be neglected.

4) Vehicle windows are considered transparent, while all other vehicle surfaces are opaque to solar radiation.

5) Incident solar radiation can be broken down into two components: horizontal and vertical radiation. The vehicle roof, which is considered flat, receives vertical radiation, while all other surfaces, considered as vertical, receive horizontal radiation.

6) The vehicle base, consisting of the dashboard, seats, upholstery and other interior elements, has a thermal capacitance, C_{base} and a dynamic temperature state T_{base}. The base receives all solar radiation transmitted through the vehicle's windows. To simplify convective heat transfer calculations, the base is assumed to be flat.

7) No heat enters the vehicle from any other means (i.e. heat transfer through the floor or heat from the vehicle engine).

Using these simplifications in conjunction with mass and thermal conservation laws, the temperature dynamics of the vehicle cabin are governed by the following equation and illustrated by Fig. 2.15:

$$m_{air}c_{p,air}\dot{T}_{air} = \dot{Q}_{ref} + \dot{Q}_{gen} + \dot{Q}_{infil} + \dot{Q}_{surf} + \dot{Q}_{base} \tag{2.2}$$

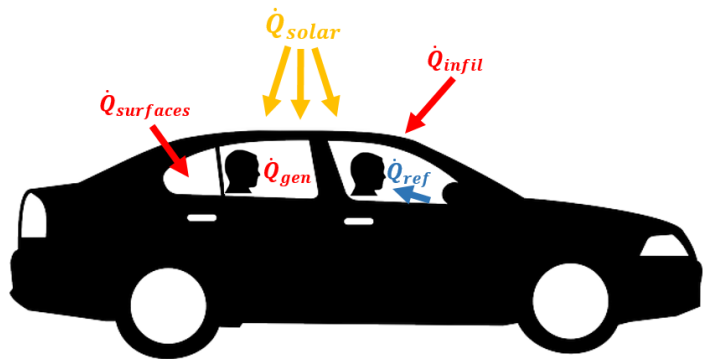

Figure 2.15 A visual illustration of the various thermal loads on the vehicle cabin.

$\dot{Q}_{ref}$**,** the cooling load provided by the vehicle's evaporator, is defined by the following equation:

$$\dot{Q}_{ref} = \dot{m}_{ref}c_{p,air}\left(T_{evap,o} - T_{air}\right) \tag{2.3}$$

Ref [16] determined the heat generation of an average adult human to be 108W. Thus, the total heat generation is given by the following equation:

$$\dot{Q}_{gen} = 108*(occupancy) \tag{2.4}$$

$\dot{Q}_{infil}$, the heat entering the cabin from infiltrating air, is defined by the following equation:

$$\dot{Q}_{infil} = \dot{m}_{infil}c_{p,air}\left(T_{\infty} - T_{air}\right) \tag{2.5}$$

$\dot{m}_{infil}$ is a function of the outside air velocity and is calculated using $\dot{m}_{infil} = \frac{0.8}{3600}v_{amb}$. This formula is derived from the work done by Fletcher and Saunders determining the air infiltration rate for a variety of vehicles [15].

$\dot{Q}_{base}$ is the heat exchange between the vehicle's base and the cabin air and is defined by the following equation:

$$\dot{Q}_{base} = h_{base}A_{base}\left(T_{base} - T_{air}\right) \tag{2.6}$$

Since the temperature of the base is a dynamic state, we then calculate the new temperature of the base by integrating the following dynamic equation:

$$C_{base}\dot{T}_{base} = \alpha_{base}\dot{Q}_{solar} - \dot{Q}_{base} \tag{2.7}$$

$\dot{Q}_{solar}$ refers to the solar radiation transmitted through the vehicle's windows, given by the equation $\dot{Q}_{solar} = \gamma\tau_{window}H_{solar}A_{windows}$ where γ is an adjustable parameter between 0 and 1 that accounts for radiative heat loss through the windows as well as for solar heat absorption by the interior surfaces. τ_{window} represents the transmittance of the windows, H_{solar} is the horizontal component of solar radiation on the window surfaces, and $A_{windows}$ is the sum of all window areas.

$\dot{Q}_{surf}$ refers to the 1-D transient conductive heat transfer through the vehicle's exterior surfaces into the cabin air. $\dot{Q}_{surf}$ is given by the following equation:

$$\dot{Q}_{surf} = \dot{Q}_{ws} + \dot{Q}_{sw} + \dot{Q}_{roof} + \dot{Q}_{wall} \tag{2.8}$$

$\dot{Q}_{ws}$ is the conductive heat transfer through the windshield, $\dot{Q}_{sw}$ is the heat transfer through the vehicle's side windows, $\dot{Q}_{roof}$ is the heat transfer through the vehicle's roof, and $\dot{Q}_{wall}$ is the heat transfer through the vehicle's side walls. Note that because truck cabins generally do not have a rear windshield, it was excluded from calculations but can be approximated by doubling the area of the windshield in the cabin model parameters.

1-D transient heat conduction is governed by the following partial differential equation and outer and inner boundary condition, equations 2.9 – 2.11 respectively:

$$\frac{1}{\alpha}\frac{dT(x,t)}{dt} = \frac{d^2T(x,t)}{dx^2} \tag{2.9}$$

$$-k\frac{dT(0,t)}{dx} = \alpha q''_{solar}(t) - h_{ext}\left(T(0,t) - T_\infty\right) - \varepsilon\sigma\left[T(0,t)^4 - T_{surr}^4\right] \tag{2.10}$$

$$-k\frac{dT(L,t)}{dx} = -h_{\text{int}}\left(T(L,t) - T_\infty\right) \tag{2.11}$$

Unfortunately, partial differential equations, especially with complex boundary conditions, are difficult to solve analytically. However, their solutions are approximated well by using discretization techniques and finite-difference methods. Finite-difference methods involve replacing derivatives in equations with discrete approximations. That is, the equations are solved at specific physical points within the material thickness. In this paper, we use the forward approximation finite difference method with a nodal energy balance to calculate the heat transfer through each of the four vehicle surfaces.

2.2.4.3.1 Cabin Surface Discretization Procedure

1) We spatially divide the given cabin surface thickness into five nodes with thermal conductivity k, thermal capacitance c_p and density ρ. We choose to discretize into five nodes because it accurately approximates the transient conduct through the surfaces without being too computationally intensive. The outer and inner nodes, labeled nodes 0 and 4 respectively, have thickness $\frac{\Delta x}{2}$, while the interior nodes 1-3 have thickness Δx. Fig. 2.16 illustrates the discretization process on the vehicle windshield.

2) Apply an energy balance to each of the five nodes as visualized by Fig. 2.17 and formalized by the following equation:

$$\sum_{n=1}^{5} \dot{q}_{in} - \dot{q}_{out} = \dot{E}_{storage} = \rho V c_p \frac{dT}{dt} \tag{2.12}$$

3) Evaluate all incoming and outgoing heat loads at time *i* and discretize the time derivative using the following forward approximation:

$$\frac{dT_i}{dt} \approx \frac{dT_i^{i+1} - dT_i^{i}}{dt} \tag{2.13}$$

4) Solve for dT^{i+1} at the outside node, the 3 interior nodes and the inner node, equations 2.14-2.16 respectively, and integrate to find new nodal temperature T^{i+1}

$$\text{Node 0: } \frac{\rho c \Delta x}{2}\frac{dT_0^{i+1}-dT_0^i}{dt} = q''_{solar} + k\frac{T_1^i - T_0^i}{\Delta x} + h_{ext}\left(T_\infty^i - T_0^i\right) + \varepsilon\sigma\left(\left(T_{surr}^i\right)^4 - \left(T_0^i\right)^4\right) \quad (2.14)$$

$$\text{Nodes 1-3: } \frac{\rho c \Delta x}{2}\frac{dT_n^{i+1}-dT_n^i}{dt} = k\frac{T_{n-1}^i - T_n^i}{\Delta x} + k\frac{T_{n+1}^i - T_n^i}{\Delta x} \quad (2.15)$$

$$\text{Node 4: } \frac{\rho c \Delta x}{2}\frac{dT_4^{i+1}-dT_4^i}{dt} = k\frac{T_3^i - T_4^i}{\Delta x} + h_{\text{int}}\left(T_{air}^i - T_4^i\right) \quad (2.16)$$

5) Solve for the conductive heat loads using the following equation:

$$\dot{Q}_{surf} = h_{\text{int}} A_{surf}\left(T_4^{i+1} - T_{air}^i\right) \quad (2.17)$$

6) After calculating the surface heat loads along with all other heat loads, $\dot{T}_{air}$ can be solved for using equation 2.2 and integrated to calculate the cabin air temperature T_{air}.

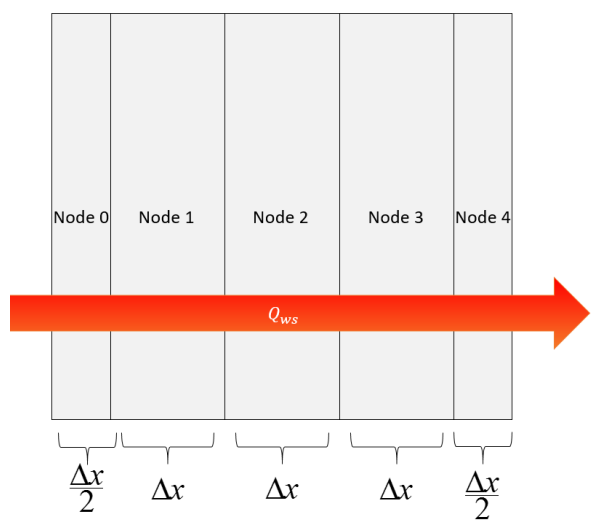

Figure 2.16 Transient conduction through a windshield discretized into five nodes.

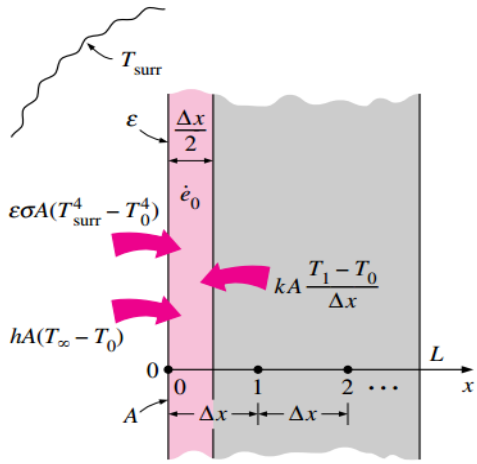

Figure 2.17 A visual example from [16] of the energy balance method applied to a discretized node.

2.2.4.3.1.1 A Note on the Wall and the Roof

Discretization becomes slightly more complicated when considering 1-D transient conduction through a heterogeneous thickness; that is, conduction through different material layers with different thicknesses. For example, [14] considers the roof of the vehicle to be composed of three different materials: steel, air and cotton, as seen as in Table 2.3. We utilize this material distribution in the cabin model as well. To deal with this, nodes 0 and 1 are defined to be steel, with thicknesses. Node 2 is defined to be air, and nodes 3 and 4 are considered as cotton. Since the materials have different thicknesses, the nodes will have different thicknesses as well as outlined in Fig. 2.18. Furthermore, we assume that the vehicle's walls has the same material composition as the roof.

Table 2.3 The material dimensions, composition and properties of the roof thickness as defined by [14].

Roof Material	Thickness (m)	Thermal Conductivity (W/m*K)
Steel	0.5E-03	14.9
Air	0.1E-03	2.6E-02
Cotton	5E-03	0.06

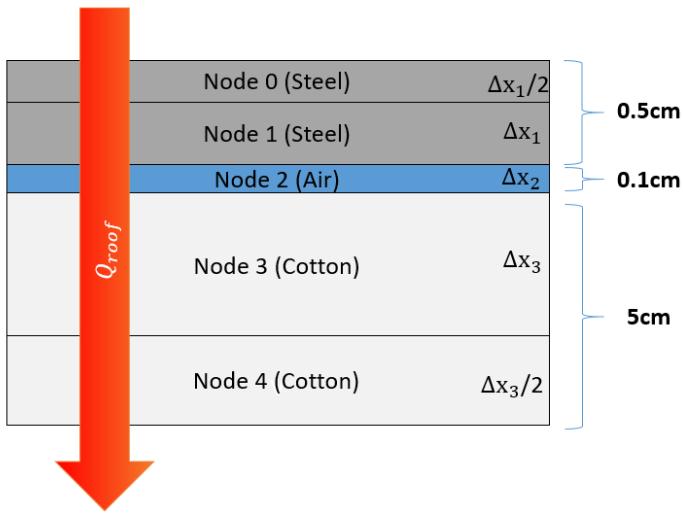

Figure 2.18 Transient heat conduction through the roof. Note that the nodes for steel, air and cotton are not uniform in thickness like for the windows.

2.2.4.4 Defining Interior and Exterior Heat Transfer Coefficients

The free convection heat transfer coefficient h_{int} refers to the heat transfer between the inside surface of the vehicle and cabin air. It is assumed that because air currents are small within the vehicle, all convective heat transfer between the inside vehicle surface and cabin air occurs as free convection, which is buoyancy-driven fluid motion generated by temperature gradients. Note that future work may consider the effect that air conditioning airflow has on this assumption.

The Rayleigh number, a dimensionless parameter associated with buoyancy-driven fluid motion governs free convection. Along with the Rayleigh number, the orientation and dimensions of the surface in contact with the fluid has the biggest impact on the heat transfer coefficient value.

The interior surface of the side windows, side walls and windshield can be approximated as a flat vertical plate. Note that this assumption is mostly true for trucks, but may not be true for cars or other vehicles with angled surfaces. The base of the vehicle is approximated as a flat horizontal surface.

The free convection heat transfer coefficient for a flat vertical plate with length L and air thermal conductivity k is given by the following equation:

$$h_{\text{int}} = \frac{k}{L}\left(0.825 + \frac{0.387Ra^{(1/6)}}{[1+(0.492/Pr)^{(9/16)}]^{(8/27)}}\right)^2 \tag{2.18}$$

where $Ra = \frac{g\Delta TL^3}{T_f \nu\alpha}$ is the Rayleigh number and Pr is the Prandtl number of the fluid.

The heat transfer between the vehicle base with length L and the cabin air is dependent on three things: the Rayleigh number, the cabin air temperature and the base temperature.

For $T_{base} > T_{air}$ and $10^4 < Ra < 10^7$:

$$h_{\text{int}} = \frac{k}{L}0.54Ra^{1/4} \tag{2.19}$$

For $T_{base} > T_{air}$ and $10^7 < Ra < 10^{11}$

$$h_{\text{int}} = \frac{k}{L}0.15Ra^{1/3} \tag{2.20}$$

For $T_{air} > T_{base}$

$$h_{\text{int}} = \frac{k}{L}0.52Ra^{1/5} \tag{2.21}$$

The heat transfer between the vehicle ceiling and the cabin air is governed by the same equations as above if one replaces T_{base} with T_{roof} and reverse the temperature inequality.

The exterior convective heat transfer h_{ext} can be governed by three separate flow regimes: free, laminar and turbulent. As stated before, free convection refers to density driven fluid motion and its heat transfer is governed by the Rayleigh number. On the other hand, laminar and turbulent fluid motion refers to fluid moving with an externally induced velocity v and the Reynolds number governs its heat transfer. In most instances, the flow on the exterior surfaces is either laminar or turbulent depending on the ambient wind speed.

For free convection on the roof, the heat transfer coefficients are governed by equations 2.19-2.21, by replacing T_{base} with T_{roof}. For free convection on all other exterior surfaces, including the windshield, side windows and side walls, the heat transfer coefficient is governed by equation 2.18.

For laminar flow on all exterior surfaces ($\mathrm{Re} = \frac{\rho v L}{\mu} < 10^5$) the heat transfer is governed by the following equation:

$$h_{ext} = \frac{k}{L} 0.664 \mathrm{Re}^{1/2} \mathrm{Pr}^{1/3} \tag{2.22}$$

For turbulent flow on all exterior surfaces ($\mathrm{Re} = \frac{\rho v L}{\mu} > 10^5$):

$$h_{ext} = \frac{k}{L}\left(0.037 \mathrm{Re}^{4/5} - 871\right) \mathrm{Pr}^{1/3} \tag{2.23}$$

2.2.4.5 Defining the Evaporator Inlet Temperature

Another feature of the cabin model is the ability to determine the evaporator air inlet temperature depending on the air recirculation percentage desired by the user. Vehicles often recirculate 70-90% of the cabin air through the evaporator because cabin air is often cooler than the air outside, decreasing the heat load on the VCS. However, it is important that some

percentage of the air at the evaporator inlet be fresh to prevent an unhealthy buildup of CO2.The evaporator air inlet temperature is given by the following equation:

$$T_{evap,i} = \frac{\dot{m}_{\infty} T_{\infty} + \dot{m}_r T_{air}}{\dot{m}_{ref}} \tag{2.24}$$

$\dot{m}_r$ is the mass flow rate of the recirculated air and is defined by the following equation:

$$\dot{m}_r = \frac{recirc/100}{\dot{m}_{ref}} \tag{2.25}$$

where *recirc* is given as a percentage.

$\dot{m}_{\infty}$ is the mass flow rate of the recirculated air and is defined by the following equation:

$$\dot{m}_{\infty} = \frac{100 - recirc}{100 * \dot{m}_{ref}} \tag{2.26}$$

2.3 Open Loop Cabin Model Validation

Proper open-loop validation of the cabin model would require comparing model outputs to experimental data collected from a vehicle. However, doing so would be very time intensive and is left for future work. Nevertheless, some model validity can be shown in two ways: 1) showing that the cabin temperature dynamics generally follow a first order response and 2) comparing cabin temperature data to experimental data found online for a given set of initial conditions.

2.3.1 Simulation Parameters

Before discussing the cabin model validation, it is important to establish the cabin model simulation parameters. Marcos et al. develop their thermal cabin model based on a four-door BMW 1-series car and their paper contains a comprehensive list of all the vehicle's parameters. A table of this data can be found below. On the other hand, our goal is to develop a model of a generic truck sleeper cabin. Obtaining vehicle dimensions for such was fairly straightforward using a datasheet for a ProStar® Sky-Rise truck found online [30]. However, the material properties and material thicknesses were not available for the truck. Therefore, we

used the parameters from [14] for all missing values. Furthermore, there are two exceptions to the above rules. Because [14] does not consider heat conduction from the walls, wall dimensions were not listed in the paper, so instead we estimated the dimensions based on images of the vehicle available online. Additionally, the base thermal capacitance value was taken from [20] because the value from [14] resulted in aberrant cabin temperature dynamics. Going forwards, the sleeper cab parameters listed in Table 2.5 will be used in all cabin model simulations in this thesis unless explicitly written otherwise.

Table 2.4 Cabin Model Parameters for a Four Door Vehicle used in [14].

Component	Parameter	Units	Value
Material Properties	Absorptivity of body		0.26
	Emissivity of body		0.9
	Thermal diffusivity of windshield	m^2/s	3.40E-07
	Thermal conductivity of windshield	W/(m*K)	1.4
	Thermal diffusivity of side window	m^2/s	3.40E-07
	Thermal conductivity of side window	W/(m*K)	1.4
	Absorptivity of window		0.2
	Emissivity of window		0.9
	Transmittance of window		0.45
	Area of base	m^2	6
	Thermal capacitance of base	J/K	5600
	Absorptivity of base		0.7
Vehicle Dimensions	Volume of cabin	m^3	3.11
	Length of roof	m	1.8
	Width of roof	m	1.1
	Length of wall	m	1.5
	Width of wall	m	0.35
	Length of windshield	m	0.63
	Width of windshield	m	1.3
	Thickness of windshield	m	6.00E-03
	Length of side window	m	1.45
	Width of side window	m	0.29
	Thickness of roof	m	5.6E-03
	Thickness of side window	m	6.00E-03

Table 2.5 Cabin Model Parameters for a Truck Cabin from [30].

Component	Parameter	Units	Value
Material Properties	Absorptivity of body		0.26
	Emissivity of body		0.9
	Thermal diffusivity of windshield	m^2/s	3.40E-07
	Thermal conductivity of windshield	W/(m*K)	1.4
	Thermal diffusivity of side window	m^2/s	3.40E-07
	Thermal conductivity of side window	W/(m*K)	1.4
	Absorptivity of window		0.2
	Emissivity of window		0.9
	Transmittance of window		0.45
	Area of base	m^2	6
	Thermal capacitance of base	J/K	5600
	Absorptivity of base		0.7
Vehicle Dimensions	Volume of cabin	m^3	8.9
	Length of roof	m	2.54
	Width of roof	m	1.83
	Length of wall	m	2.81
	Width of wall	m	1.02
	Length of windshield	m	0.808
	Width of windshield	m	1.524
	Thickness of windshield	m	6.00E-03
	Length of side window	m	0.768
	Width of side window	m	0.768
	Thickness of roof	m	5.6E-03
	Thickness of side window	m	6.00E-03

2.3.2 Open Loop Simulation Results

The temperature dynamics of a given object is typically modeled by the following first order transfer function:

$$\frac{T(s)-T_o}{Q(s)}=\frac{K}{\tau s+1} \tag{2.27}$$

where $T(s)$ is the object's temperature, T_o is the temperature of the object with no external heat loads, $Q(s)$ is the net heat load on the object, K is the transfer function gain, and τ is the system time constant. In essence, this equation states that for this transfer function, any change in temperature is proportional to the incident heat load on the object.

To determine whether the cabin model follows this behavior, the relationship between heat loads and final temperature for the cabin model was explored. In particular, the dependence of final cabin temperature on passenger occupancy was examined. Because of the superposition principle of linear transfer functions, we can ignore the other system heating/cooling loads for the time being and look solely at the effect of increasing passenger occupancy on the change in final cabin temperature, which is equal to the product of K and $Q(s)$. As mentioned previously, each passenger produces approximately 108W of heat. Therefore, one would expect the change in cabin temperature to be proportional to passenger occupancy.

For no passengers, the steady state cabin temperature is $T_o = 9.47°C$. *For N=2, N=3* and *N=4*, the steady state temperatures are 16.6°C, 20.4°C and 24.3°C respectively, as seen in Fig. 2.19. The proportional gain K for each passenger-loading scenario is 0.0333, 0.0340 and 0.0347 respectively. The closeness of these K values indicates the high proportionality between the cabin temperature and passenger accuracy and suggests that equation 2.25 can accurately model the temperature dynamics of the cabin interior.

Another characteristic of the first order response is the time constant τ that represents the time it takes for the temperature to reach 63.2% of its final steady state value. For *N=2*,

N=3 and *N=4*, the time constants are τ =373s, τ =315s, and τ = 229s respectively. The discrepancy between τ values indicates that there are dynamic nonlinearities at play, but the relatively close time constants indicate that the cabin temperature can be generally modeled by a first order transfer function. Table 2.6 summarizes these results.

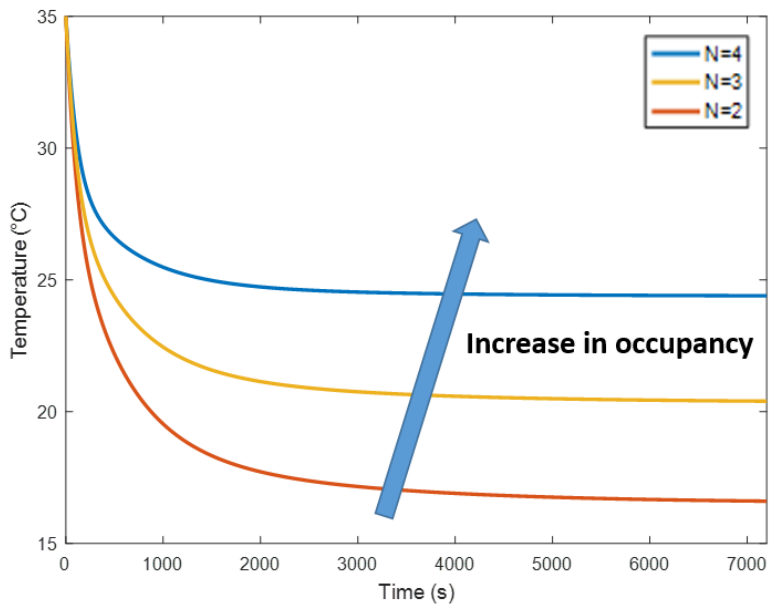

Figure 2.19 Cabin air temperature for different number of vehicle occupants, *N*, given a cooling load of $\dot{Q}_{ref} = -1000W$ and an initial cabin temperature of 35 degrees.

Table 2.6 Open Loop Cabin Temperature Dynamics.

Number of Occupants (N)	**Occupant Heat Load (W)**	**Final Cabin Temperature (°C)**	**Gain (K)**	**Time Constant (s)**
2	216	16.6	0.0333	373
3	324	20.4	0.0340	315
4	432	24.39	0.0347	229

2.3.3 Other Cabin Model Results

Figures 2.20 and 2.21 below depict the cabin model response to varying solar and cooling loads respectively.

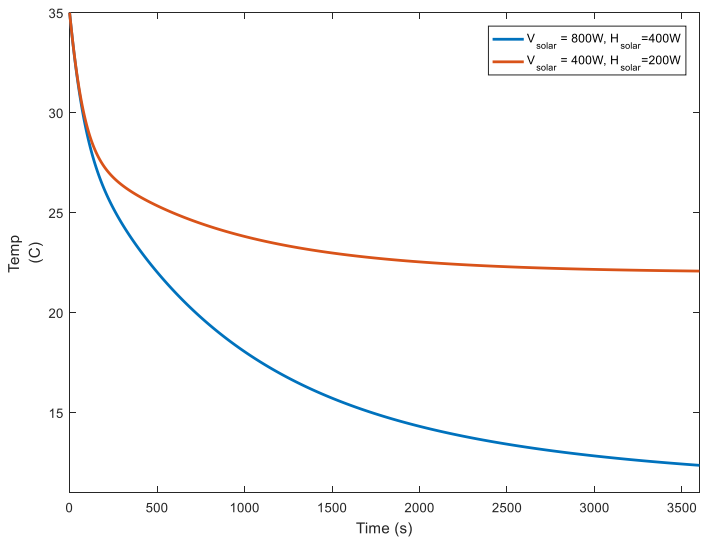

Figure 2.20 Cabin air temperature over time for two different sets of horizontal and vertical solar radiation.

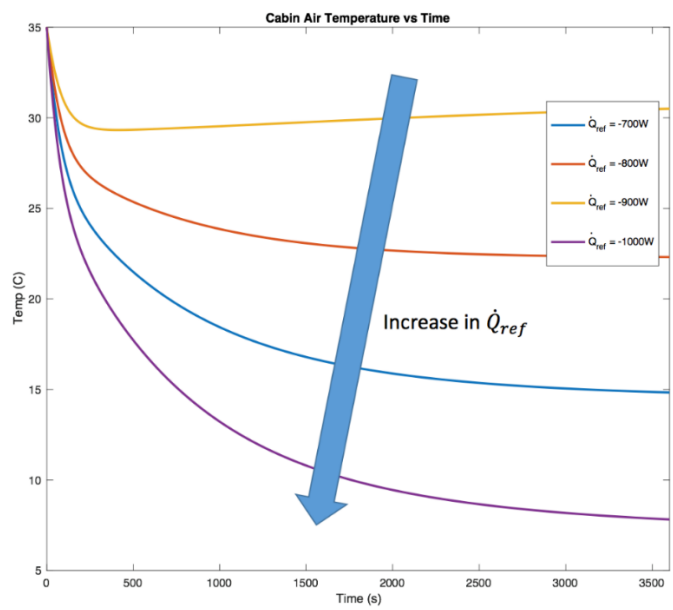

Figure 2.21 Cabin air temperature over time given different evaporator cooling loads.

Fig. 2.22 illustrates the evolution of various cabin heat loads over time, with most of them reading steady state within one hour. Of all the cabin heat loads, $\dot{Q}_{base}$ has the most unique heat load evolution over time. Initially, the base heat load is zero, because the simulation assumes that the cabin air and base temperature are the same to start. However, because all transmitted solar radiation is assumed to fall on the base, the base becomes much warmer than the surrounding air. As time goes on, the heat load settles as the cabin air and base reach their respective steady state temperatures.

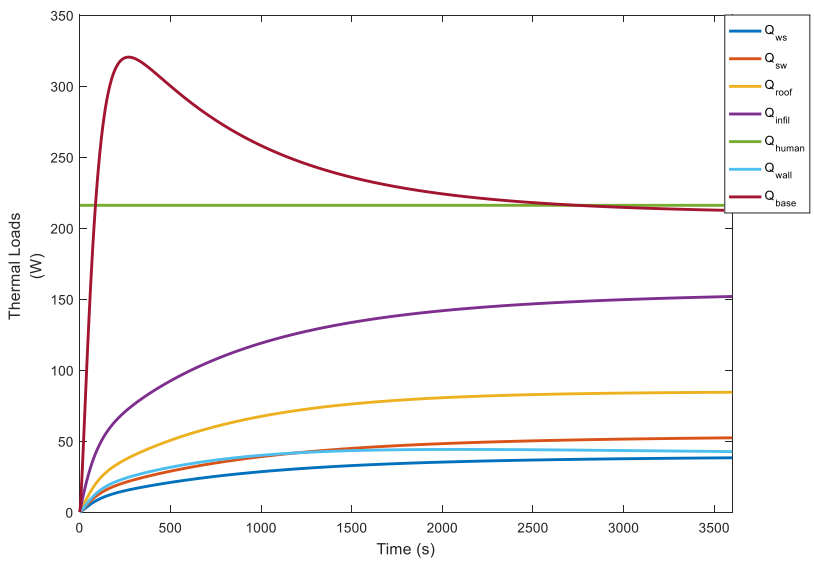

Figure 2.22 Evolution of cabin heat loads over time

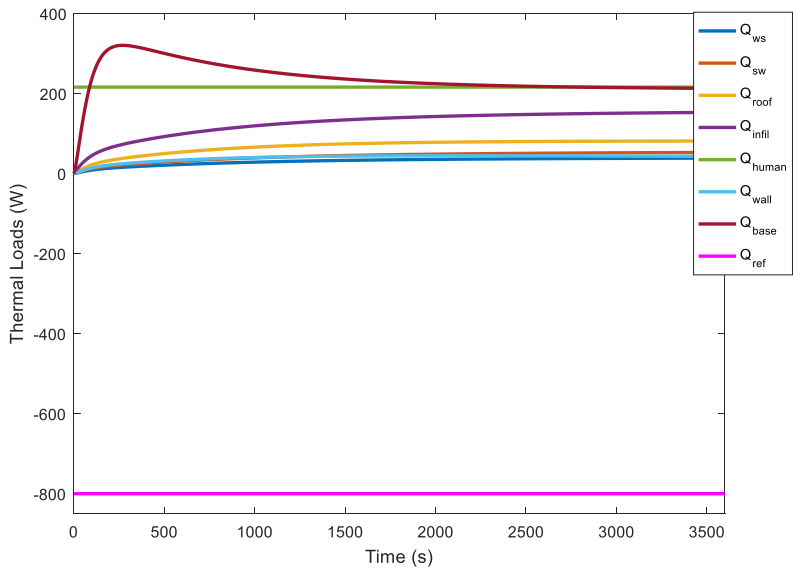

Figure 2.23 All of the cabin heat loads over time, including the cooling capacity.

2.3.4 Experimental Validation of Simulation Data

Another validation method is to compare the cabin model temperature response to experimental data for a given set of environmental conditions. There is extensive literature on the rapid increase in vehicle cabin temperature of a closed vehicle on a hot, calm, sunny day due to the risk posed to small children and pets. One example of such data is shown in Fig. 2.24. These conditions can be modeled by assuming a horizontal solar flux of 400W/m^2, a vertical solar flux of 800 W/m^2, which is typical of a clear summer day, and a light wind speed of 1 m/s. The sky temperature can be estimated by subtracting 20 degrees from the ambient temperature [19]. Furthermore, we assume the vehicle used in the experiment is similar to the four-door vehicle modeled in [14], thus we use the parameters listed in Table 2.4 in this instance. A list of all inputs can be found in Table 2.7. The temperature evolution of the cabin model over an hour is compared to experimental data from for three different ambient temperatures. The results indicate a high level of model accuracy, as detailed in Table 2.8 and further illustrated in Fig. 2.25.

Estimated Vehicle Interior Air Temperature v. Elapsed Time

Elapsed time	Outside Air Temperature (F)					
	70	75	80	85	90	95
0 minutes	70	75	80	85	90	95
10 minutes	89	94	99	104	109	114
20 minutes	99	104	109	114	119	124
30 minutes	104	109	114	119	124	129
40 minutes	108	113	118	123	128	133
50 minutes	111	116	121	126	131	136
60 minutes	113	118	123	128	133	138
> 1 hour	115	120	125	130	135	140

Courtesy Jan Null, CCM; Department of Geosciences, San Francisco State University

Figure 2.24 An estimate of the cabin air temperature of a closed vehicle on a hot summer day from [21] given a number of different ambient temperatures.

Table 2.7 Inputs used for the Simulation Case Study

Input	**Value**
Ambient wind (m/s)	1
Sky Temp (C)	$T_{sky} = T_{amb} - 20$
Vertical Solar Irradiance (W/m^2)	800
Horizontal Solar irradiance (W/m^2)	400
Cooling Capacity (W)	0
Evaporator Air Supply (kg/s)	0

Table 2.8. A Limited Validation for the Cabin Model, with an Average RMS Error under 2.5 Degrees F.

	Time (Min)	Actual Cabin Temperature ($°F$)	Model Cabin Temperature ($°F$)	Error ($°F$)
$T_0 = 70°F$	T = 20	99	94.93	-4.07
	T = 40	108	106.71	-1.3
	T = 60	113	112.91	-0.09
$T_0 = 80°F$	T = 20	109	104.74	-5.26
	T = 40	118	116.22	-1.78
	T = 60	123	122.23	-0.77
$T_0 = 90°F$	T = 20	119	114.55	-4.45
	T = 40	128	125.76	-2.24
	T = 60	133	131.58	-1.42

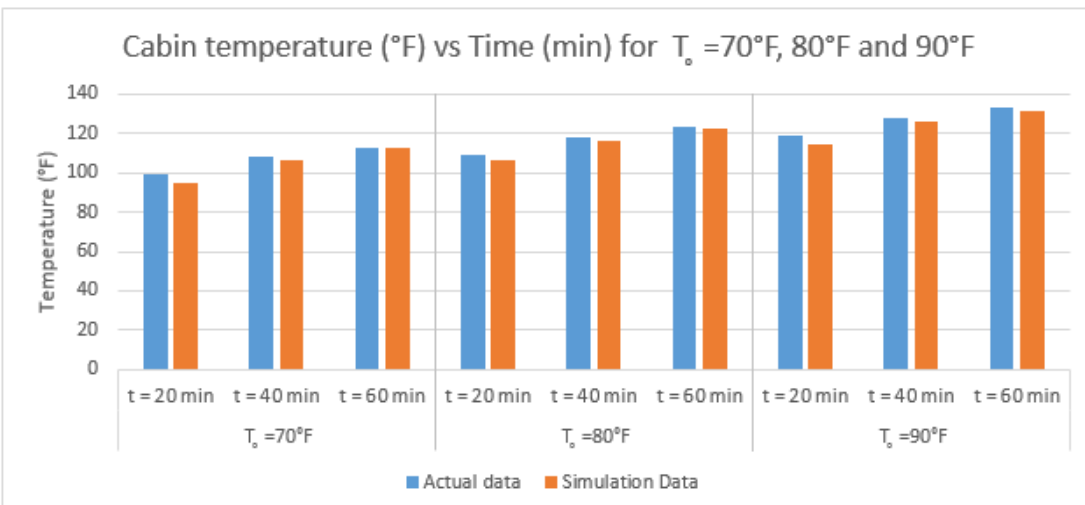

Figure 2.25 Comparing the experimental data to the cabin model temperature over the course of an hour. Cabin model data closely matches the empirical data.

2.4 Overview of the Experimental System

The experimental system is an integrated setup designed to replicate a no-idle VCS unit cooling a truck sleeper cabin. Some aspects of the integrated system were simplified due to time and budget constraints. For example, instead of cooling an actual vehicle cabin, we construct an insulated, enclosed rectangular space to cool. Furthermore, the main heat load imposed on the cabin originates only from internal infrared heat lamps as opposed to the dynamic combination of radiative, conductive and convective heat loads that vehicle cabins are subjected to in outdoor conditions. The clear differences between the experimental and simulated integrated systems precludes any cross validation between the two; however, the experimental system is similar enough in design that dynamic behavior and response reflects that of the NITE system in nominal operation. Fig. 2.26 highlights the key system components and a detailed explanation is presented below.

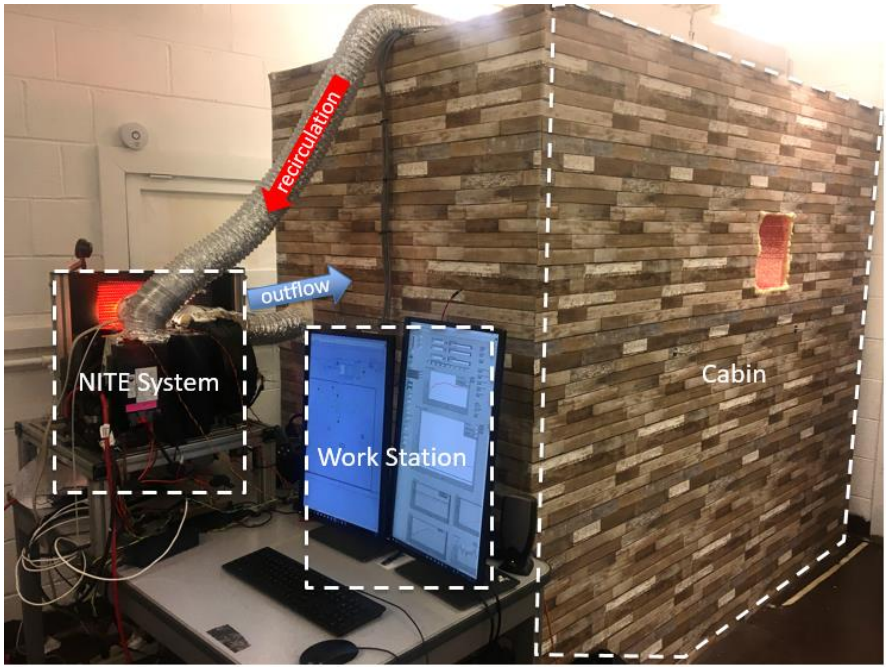

Figure 2.26 A labeled picture of the experimental setup.

2.4.1 The NITE System

As mentioned previously, the NITE system is a no-idle VCS unit developed by Bergstrom Inc, a company specializing in the development of cabin climate systems for trucks and other vehicles. The VCS used in this setup is the NITE Phoenix SSI, which is a battery operated VCS unit featuring a split condenser/evaporator system as seen in Fig. 2.27. The compressor, TXV and evaporator are enclosed in a single unit, with the evaporator blower attached to the side of the unit. The condenser coil and fan unit are enclosed in a unit together and are connected to the other unit via refrigerant tubes. The evaporator blower feeds cool air to the cabin via an outflow duct, and the warm evaporator air inflow is supplied via a recirculation duct from the cabin. In normal application, the condenser unit is housed outside in warm ambient conditions while the evaporator is placed inside the vehicle. In order to replicate the ambient heat load on the condenser coils, a 250W heat lamp is shone on the condenser coils as seen in Fig. 2.28.

The NITE system is equipped with various sensors in order to monitor system behavior. In particular, LM35 analog temperature sensors were installed at the evaporator air inlet/outlet, the compressor inlet/outlet and a pressure transducer was installed at the compressor inlet. Importantly, the pressure and temperature readings at the compressor inlet are used to verify that the refrigerant is superheated prior to entering the compressor.

The NITE system sends and receives signals using the CAN bus protocol, which is an automotive communication standard used by vehicle subsystems to communicate with each other. Using the CAN bus protocol, one can read messages from the NITE system such as component speeds as well as read any warning messages coming from the system. Furthermore, the NITE system's compressor, condenser and evaporator fan speeds can be overridden by writing a CAN message to the system. This feature is crucial for closed-loop control of the system.

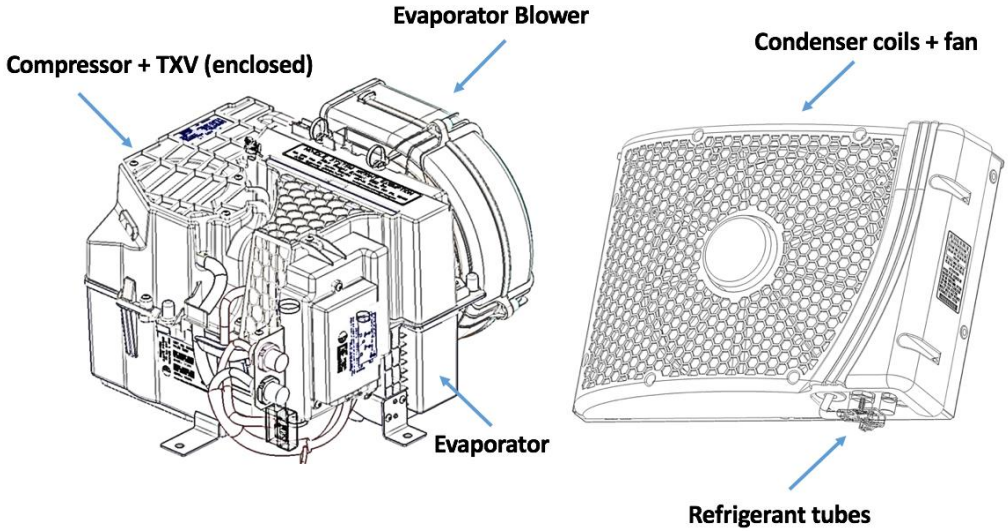

Figure 2.27 A labeled schematic of the NITE Phoenix SSI given by Bergstrom Inc.

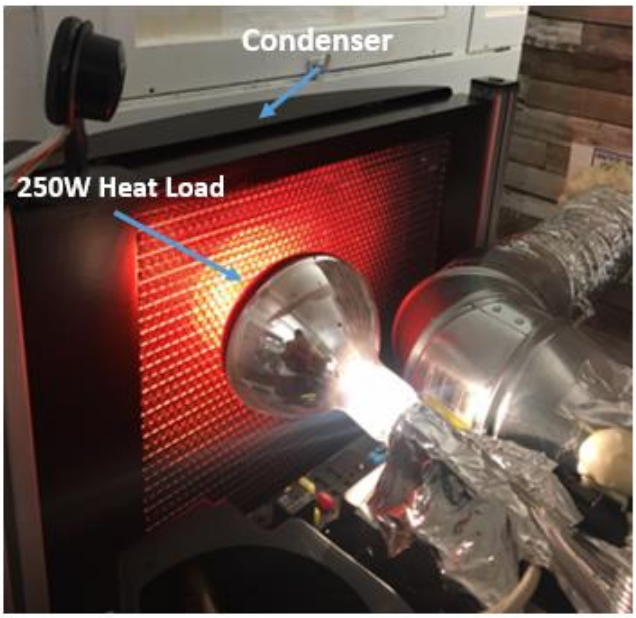

Figure 2.28 A picture of the condenser unit along with the 250W heating load.

2.4.2 The Cabin

The cabin is an 8ft x 4ft x 7ft enclosed space that is cooled by the NITE system. The cabin has a large volume of 6.3 cubic meters designed to emulate the volume inside a truck cabin. The cabin is constructed out of eight insulating foam board panels joined using heavy-

duty tape and metal rods for stability. To replicate the heat loads on the cabin, two pairs of heat lamps are suspended inside the cabin, each generating 500W and 375W of heat respectively, as seen in Fig. 2.29. A small fan is placed inside the cabin to mix the inside air. Two LM35 analog temperature sensors are suspended inside the cabin in order to generate an accurate estimate of the average cabin temperature, as seen in Fig. 2.30. One duct enters the cabin bringing in cool air from the evaporator, while the other duct recirculates warm cabin air from the cabin to the inlet of the NITE's evaporator, as is done in standard practice.

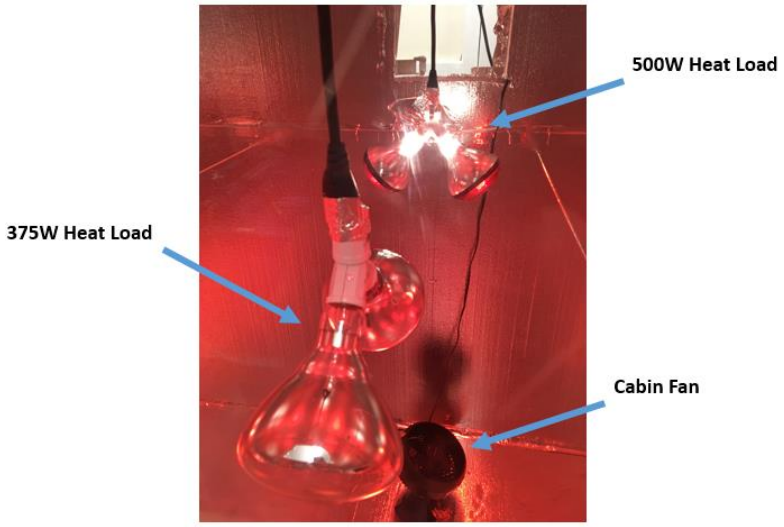

Figure 2.29 A picture of the 500W and 375W heat loads suspended inside the cabin. The 500W heat load consists of two 250W heat lamps, while the 375W heat load consists of a 250W and 125W heat lamp. Each heat load is individually controllable by the user depending on desired head load. Also seen is the cabin fan that mixes the air inside to increase temperature uniformity.

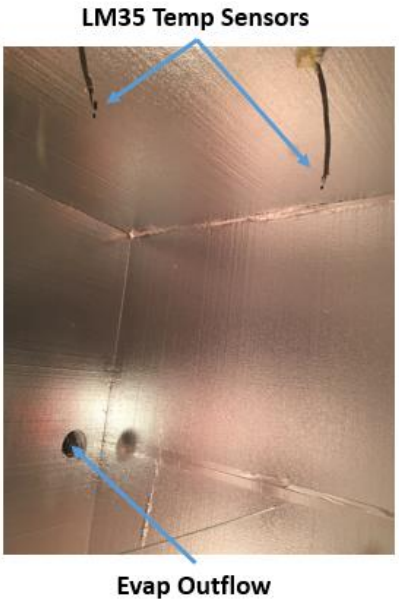

Figure 2.30 A picture taken inside the cabin, showing the two LM35 temperature sensors. Also visible is the evaporator outflow duct that brings cool air inside from the NITE system.

2.4.3 The Workstation

The workstation consists of the computer as well as necessary components to process, send and receive data between the NITE and the computer. Fig. 2.31 outlines those components.

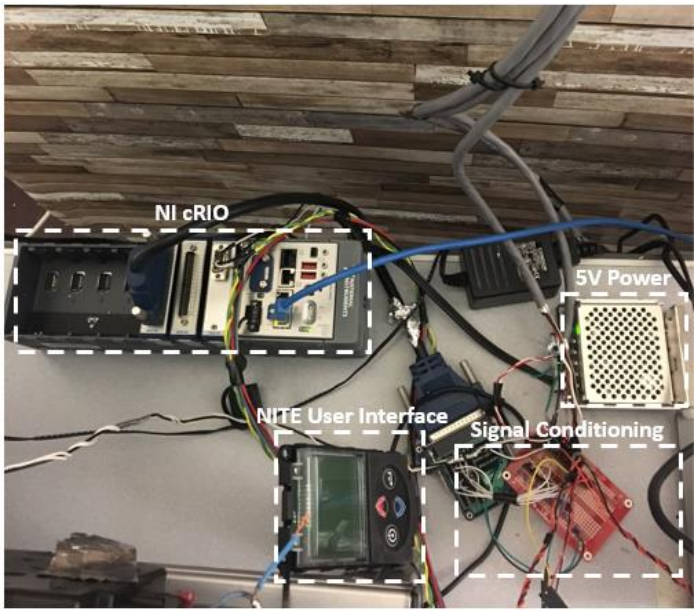

Figure 2.31 A picture of the workstation components. The NI cRIO communicates to the NITE over CAN and receives analog signals from various system transducers. The signal conditioning circuit, in conjunction with the 5V supply, powers the sensors and filters their outputs to reduce noise. The NITE User Interface is used to turn the NITE on or off.

A National Instruments (NI) cRIO 9035 equipped with a CAN bus interface module (the NI-9862) is connected to the CAN bus cable from the NITE system. The cRIO is also equipped with the NI-9205 analog input module that receives data from the sensors after going through signal conditioning. In turn, the NI cRIO is connected to the computer over Ethernet. This configuration can be seen in detail in Fig. 2.32.

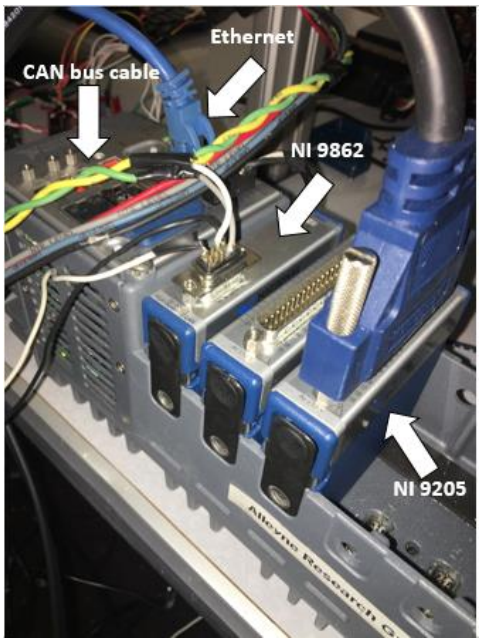

Figure 2.32 A close-up of the NI cRIO 9035

All six sensors used in the system are analog sensors, which require signal conditioning in order to remove noise. The signal conditioning board, as seen in Fig. 2.31, contains six low-pass filters to attenuate high-frequency noise. A 5V power supply is used to power all of the analog sensors. Table 2.9 lists all of the sensors used in this setup.

Table 2.9 Analog Sensors used in the Experimental Setup.

	Sensor Name	Location	Purpose
Sensor 1	TI LM35	Outlet of evaporator blower	Record cabin temperature
Sensor 2	TI LM35	Taped behind computer monitor	Record cabin temperature
Sensor 3	TI LM35	Inside cabin	Estimate refrigerant temperature at compressor inlet
Sensor 4	TI LM35	Inside cabin	Estimate refrigerant temperature at compressor outlet
Sensor 5	DWYER 628-05	Inserted in compressor inlet	Record low-side system pressure
Sensor 6	TI LM35	Inlet of evaporator blower	Record room (ambient) temperature
Sensor 7	TI LM35	Taped to compressor inlet tube	Approximate compressor inlet refrigerant temperature

The NI cRIO and the computer communicate via the NI software package LabVIEW. This software package allows users to read CAN messages from the NITE and send user-defined CAN messages to control the speed of its actuators (the speed of the compressor, condenser fan and evaporator blower is set by a user-defined PWM signal with a value between 0-255). Furthermore, LabVIEW allows the user to read and manipulate analog signals sent from sensors. A basic LabVIEW file sending and receiving CAN signals and reading analog sensor data is shown in Fig. 2.33.

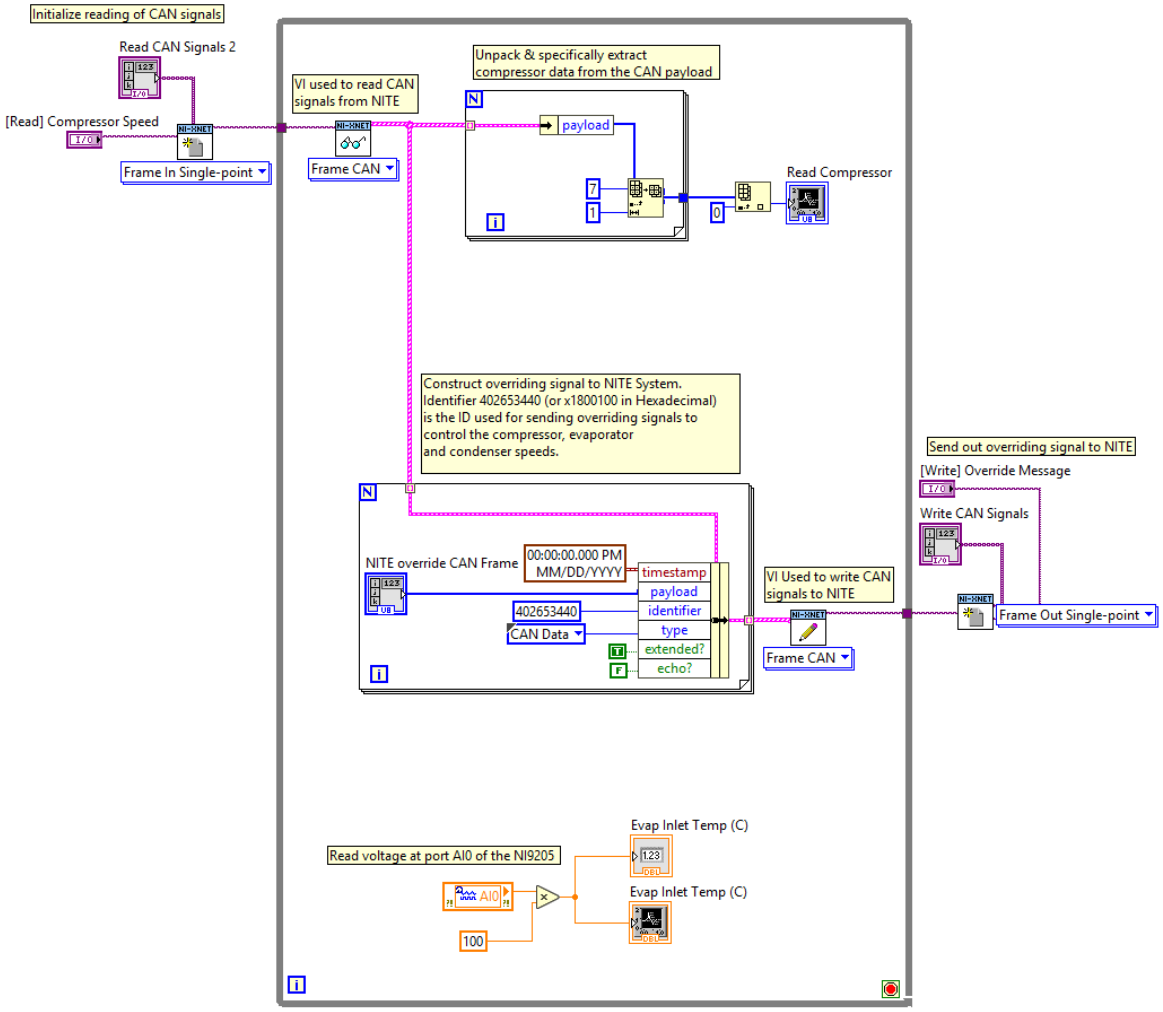

Figure 2.33 A basic script (called a VI) developed in LabVIEW that sends and receives CAN signals and also reads and manipulates analog sensor signals.

2.5 Basic Control Strategies and Closed Loop Validation

The main goal of a VCS is to regulate the temperature of a given space. A simple example of this is a thermostat: its goal is to keep the temperature of a room at a certain value. This raises some important questions. How do we control a VCS such that it regulates the temperature of a given space? What actuator(s) do we manipulate to achieve this goal? Furthermore, what control architecture do we utilize?

The main VCS output of interest is the cooling capacity, which is the amount of heat absorbed by the evaporator. The equation for the cooling capacity is defined as follows:

$$\dot{Q}_{ref} = \dot{m}_{ref} c_{p,air} (T_{evap,o} - T_{evap,i}) \tag{2.28}$$

There are two variables that affect the value of $\dot{Q}_{ref}$: the evaporator mass flow rate, $\dot{m}_{ref}$ and the temperature differential between the inlet and outlet air streams, $(T_{evap,o} - T_{evap,i})$ .The evaporator mass flow rate is controlled directly by the evaporator fan speed setting, while VCS literature shows that the air temperature differential is most correlated with the speed of the compressor [17]. As the compressor speed is increased, more heat is rejected through the condenser, which in turn leads to a cooler refrigerant being passed through the evaporator coils that absorbs more heat. The basic strategy commonly used in industry is to control the cooling capacity by only modulating the compressor speed. Hence, this section looks at the control of the VCS through compressor modulation with a constant evaporator speed, but future sections will address the important role that both components play in optimal cooling.

2.5.1 Closed Loop Structure

Fig. 2.34. details the common VCS control structure.

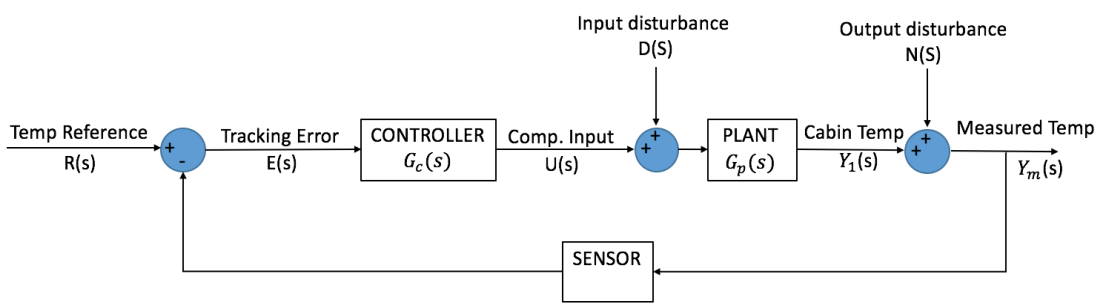

Figure 2.34 A block diagram of closed loop control implemented on a VCS, where the controller aims to track a given reference temperature for the cabin.

The goal of this closed loop control architecture is to track a reference temperature $R(s)$ by measuring the tracking error and using a controller $G_c(s)$ to steer the cabin temperature to the reference value. The closed loop transfer function is described as follows:

$$Y_m(s) = \frac{1}{1+G_p(s)G_c(s)}\left(N(s)+G_p(s)D(s)+G_p(s)G_c(s)R(s)\right) \tag{2.29}$$

For this to be a robust control design, we need to determine whether the system can reject noise *N(s)*, input disturbances *D(s)* and track the reference signal *R(s)* (note that noise and disturbances only apply to the experimental system). But before doing so, we need to determine the characteristics of $G_p(s)$ for the experimental system and also determine $G_c(s)$. $G_p(s)$ is the plant model that characterizes the relationship between the compressor input and the cabin temperature. Prior simulation results indicated that $G_p(s)$ is well modeled by a first order transfer function, and we use this model structure for deriving the physical cabin's temperature dynamics. After characterizing the plant, we determine the robustness of the closed loop controller.

2.5.2 Determining the Physical Cabin Model

In order to determine the gain and time constant of the physical cabin model, we step the compressor speed at different values and observe the dynamic temperature response. In this case, the input to this system is the compressor PWM, a value between 0-255 which is proportional to the compressor RPM. The evaporator blower and condenser fan speeds were held constant at 160 and 70 PWM respectively. To characterize the system dynamics, we observed the cabin temperature response to three different compressor speeds and determined its first order characteristics. The initial cabin temperature was set at 35 degrees C. Results are outlined in the following table and in Fig. 2.35.

Table 2.10 Open Loop Experimental Cabin Temperature Dynamics.

Compressor PWM (0-255)	Cooling Capacity (W)	Final Cabin Temperature (°C)	Gain (K)	Time Constant (s)
40	-1153	28.48	0.2178	201
60	-1200	27.15	0.1308	218
100	1231	26.30	0.0870	220

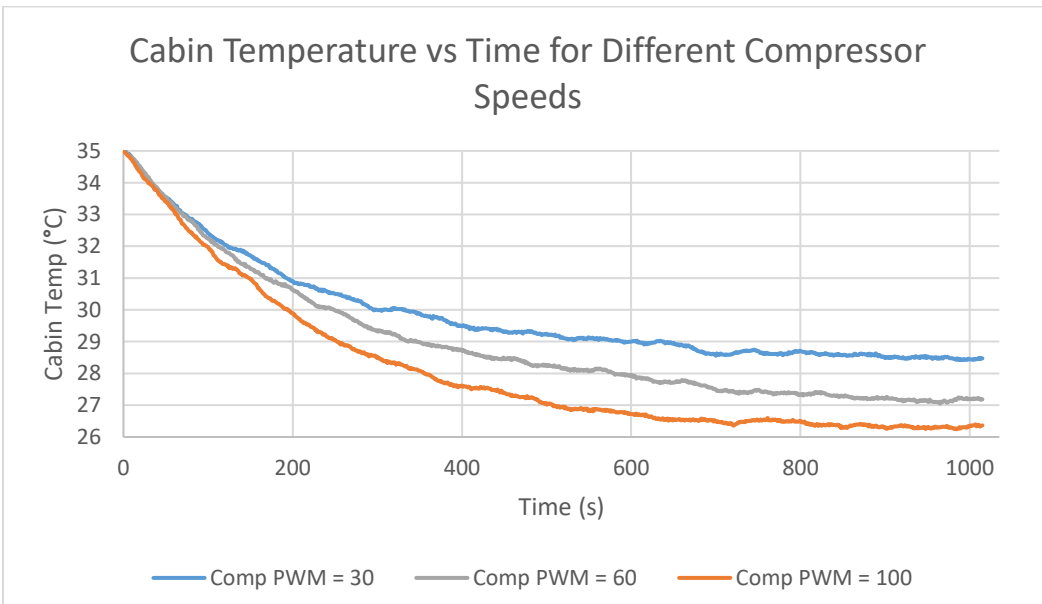

Figure 2.35 Cabin temperature response to different compressor PWM speeds, starting from an initial temperature of 35 degrees C.

Despite some gain variability, the results show that the cabin is modeled well by a first order system, just as the simulated cabin was modeled by a first order system. Averaging the gain and time constant values, we get the following physical cabin transfer function:

$$G_p(s) = \frac{T_o - T(s)}{U(s)} = \frac{K}{\tau s + 1} = \frac{0.1453}{213s + 1} \tag{2.30}$$

2.5.3 Determining the Controller Transfer Function

The controller transfer function, $G_c(s)$, is a user defined function designed to shape the closed loop system response. We examine two common control algorithms, proportional and proportional-integral control.

Proportional control, also known as P control, outputs a control input to the plant that is proportional to the tracking error. The transfer function of such a controller is given as follows:

$$G_c(s) = \frac{U(s)}{E(s)} = K_p \tag{2.31}$$

where $K_p(s)$ is the proportional gain. For some systems, P control may be sufficient to track a given reference signal. However, this is not the case for first order plants such as the cabin model. In fact, P control on a first order system results in a steady state error that is defined by the following equation:

$$e_{ss} = \frac{R}{1 + K_p K} \tag{2.32}$$

Proportional-integral control, also known as PI control, outputs a control input to the plant by sending a control input proportional to both the error and the error integral. By adding integral action, the steady state error is zero for first order systems. It is defined by the following equation.

$$G_c(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} \tag{2.33}$$

where $K_i(s)$ is the integral gain.

2.5.4 Closed Loop Simulation Results

The first control algorithm used on the simulated system was P-control. As stated before, P-control should generate a steady state error for any given K_p. Fig. 2.36 and 2.37 shows the cabin temperature and compressor RPM respectively for three different proportional

gains. As seen below, proportional control fails to eliminate the steady state error, further validating the first order characterization of the cabin model.

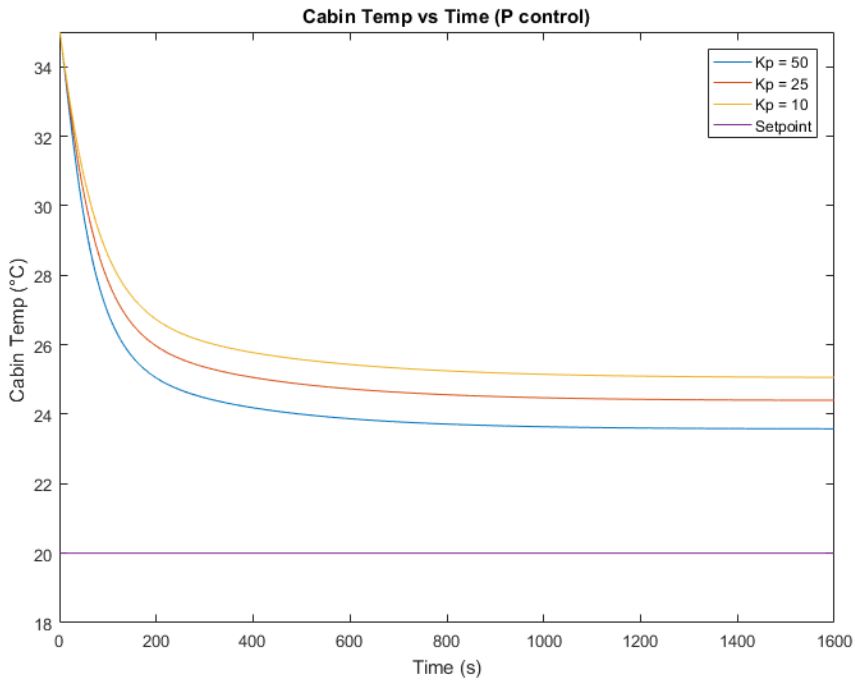

Figure 2.36 The cabin temperature over time for three different proportional gains. None of the proportional controllers successfully track the temperature set point, yielding a steady state error.

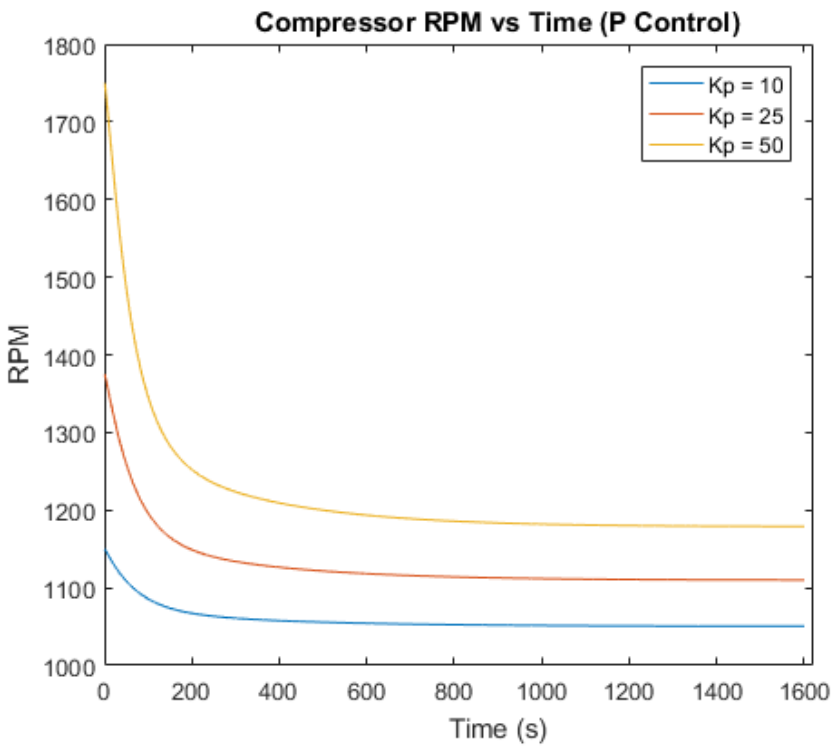

Figure 2.37 The compressor RPM over time for the three different proportional gains. The compressor RPM does not reach the speed necessary to cool the cabin to the temperature set point.

On the other hand, utilizing PI control on a first order transfer plant should allow successful reference tracking. Figures 2.38 and 2.39 show the cabin temperature and compressor RPM respectively for the simulated system. As seen below, the cabin temperature quickly converges to the desired temperature set point of 20 degrees C, as the compressor RPM converges to the value necessary to maintain that temperature.

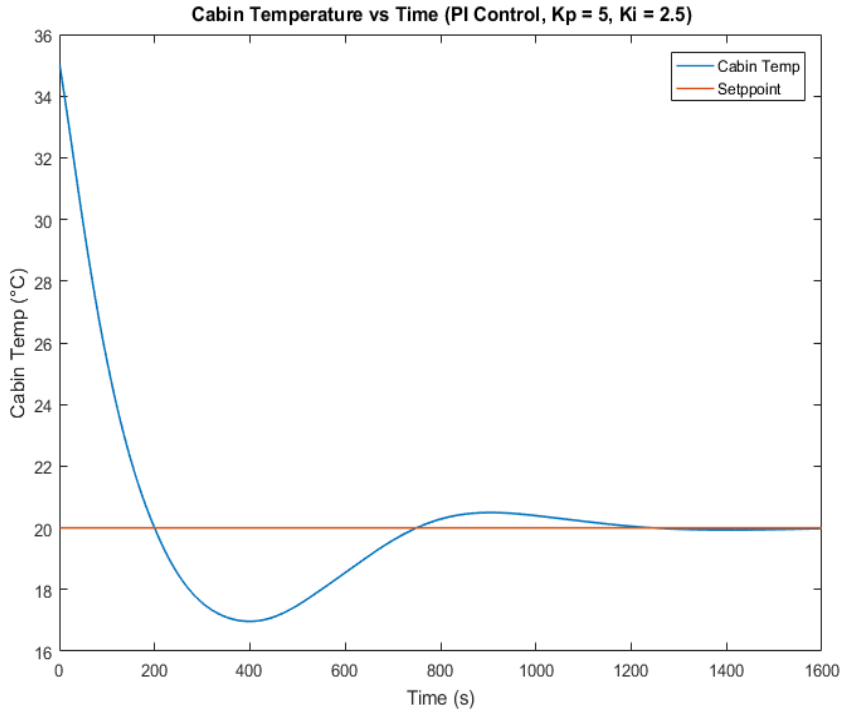

Figure 2.38 The simulated cabin temperature over time. The cabin temperature, initially at 35 degrees C, converges to the temperature set point in approximately 20 minutes.

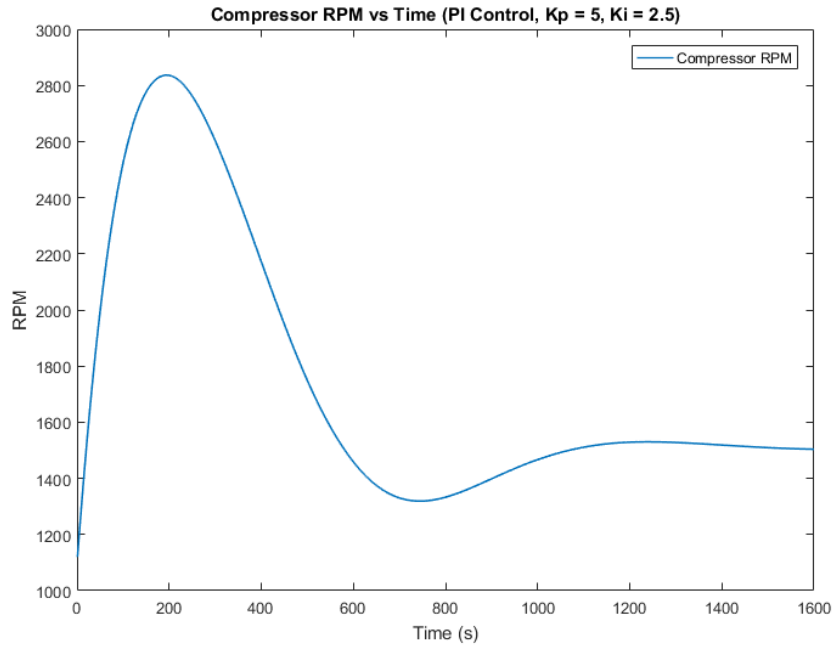

Figure 2.39 The compressor RPM over time. The compressor speed initially increases to pull down the cabin temperature and levels off once the cabin temperature reaches the desired value.

2.5.5 Closed Loop Experimental Results

The previous section demonstrated the ability of PI control to track a temperature reference, while P control was found to be insufficient due to steady state error. Despite the differences between the modeled and experimental system, we expect the same general trends to apply to the experimental system as well. Like before, we initialize the cabin temperature at 35 degrees C and attempt to drive the temperature to a set point of 27 degrees C, given a cabin heat load of 875W. Fig. 2.40 and 2.41 shows the cabin temperature and compressor PWM over time when only using proportional control. As the simulation results predicted, the cabin temperature does not track the reference temperature.

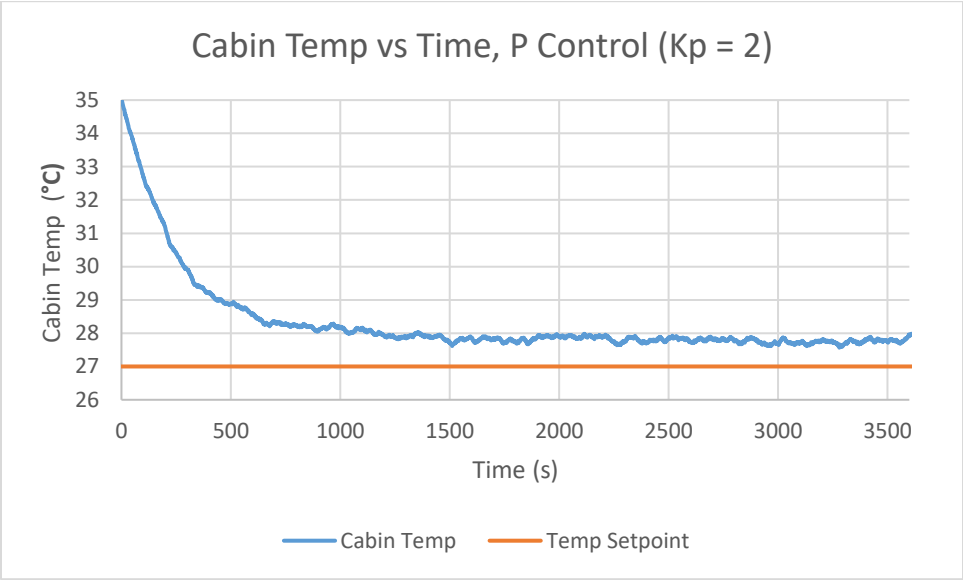

Figure 2.40 The cabin temperature over time, along with the temperature set point. The cabin temperature fails to track the reference temperature, resulting in a steady state error.

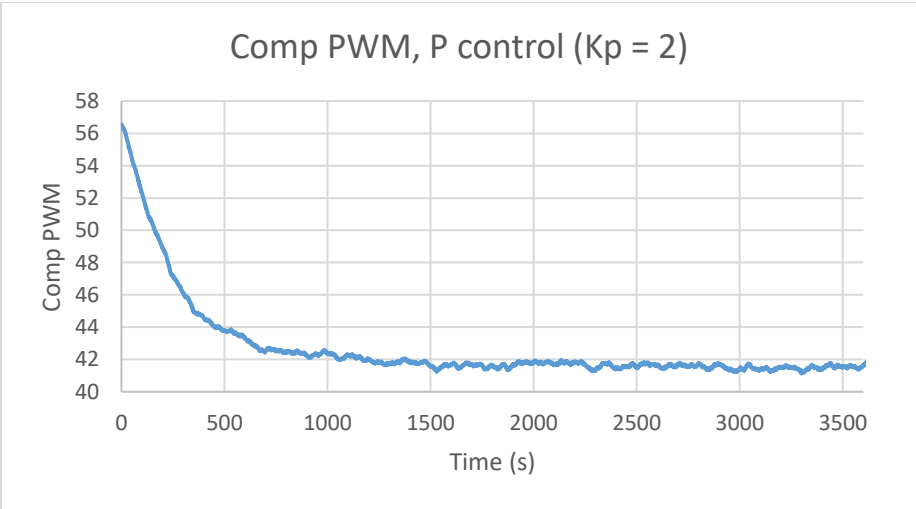

Figure 2.41 The compressor PWM over time. The compressor PWM is greatest at the beginning due to the large initial error between the cabin temperature and temperature reference. The PWM never reaches a high enough value to cool the cabin sufficiently

On the other hand, we expect the PI controller to track the temperature reference successfully based on theory and simulation results. This assumption holds true; Fig. 2.42 shows the cabin temperature converging to the desired set point, while Fig. 2.43 shows the compressor PWM speeds converging to the necessary value to track the reference temperature.

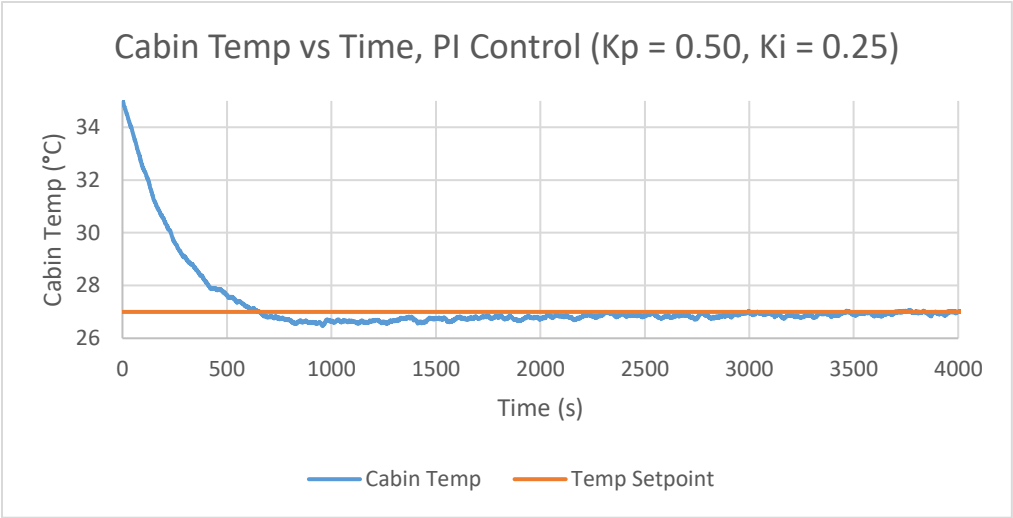

Figure 2.42 The cabin temperature over time, starting from an initial temperature of 35 degrees C. The cabin temperature converges to the set point in roughly 30-40 minutes.

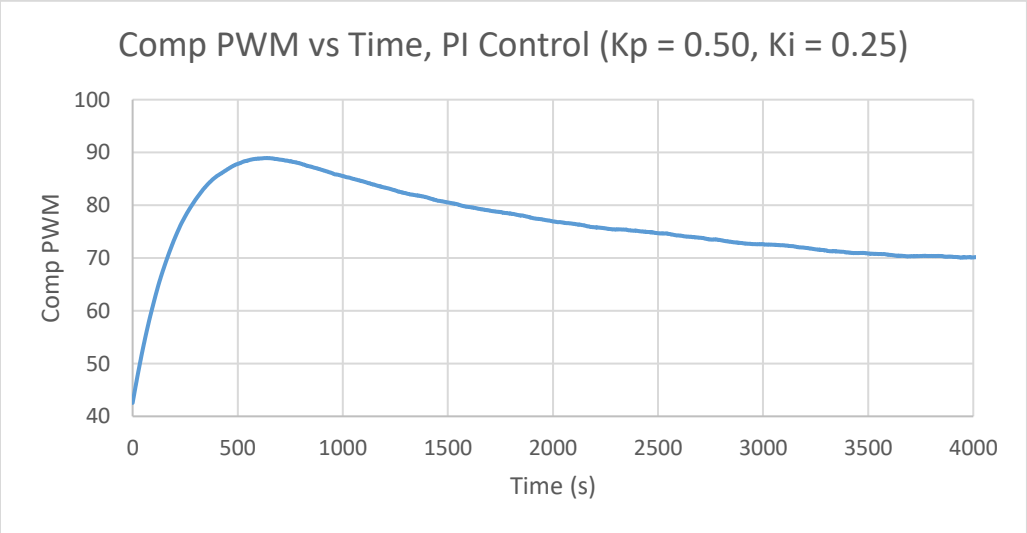

Figure 2.43 The compressor PWM over time. The PWM initially increases to bring down the cabin temperature and stabilizes once the cabin temperature reaches the temperature set point.

It is clear that PI control successfully tracks a given reference signal. However, this is not the only important criteria in judging a control scheme's effectiveness. An effective control scheme must also be robust to noise and input disturbances. To determine controller robustness, we first rewrite Eq. 2.29, the closed loop transfer function. The closed loop transfer function is as follows:

$$Y_m(s) = \frac{G_p(s)G_c(s)}{1+G_p(s)G_c(s)}R(s) - \frac{1}{1+G_p(s)G_c(s)}N(s) + \frac{G_p(s)}{1+G_p(s)G_c(s)}D(s) \qquad (2.34)$$

Eq. 2.34 shows that the plant output is a function of the reference signal $R(s)$, the output noise $Y(s)$, and the input disturbance $D(s)$. For the system to be robust to noise and disturbances, we need the transfer functions $\frac{1}{1+G_p(s)G_c(s)}$ and $\frac{G_p(s)}{1+G_p(s)G_c(s)}$ to have low gains for the respective noise and disturbance frequencies, effectively attenuating those signals. $\frac{G_p(s)}{1+G_p(s)G_c(s)}$ is referred to as the sensitivity transfer function, $S(s)$, while $\frac{G_p(s)}{1+G_p(s)G_c(s)}$ is referred to as the disturbance rejection transfer function, $D(s)$. Substituting in the physical plant's transfer function and controller transfer function with the same gains as used in the experimental system ($K_p = 0.5$, $K_i = 0.25$), we get the following disturbance rejection and sensitivity transfer functions, Eq. 2.35 and 2.36 respectively.

$$D(s) = \frac{\frac{0.1452}{213s+1}}{1+\left(\frac{0.1452}{213s+1}\right)\left(0.5+\frac{0.25}{s}\right)} = \frac{30.93s^2+0.1452s}{45369s^3+441.5s^2+8.804s+0.0363} \qquad (2.35)$$

$$S(s) = \frac{1}{1+\left(\frac{0.1452}{213s+1}\right)\left(0.5+\frac{0.25}{s}\right)} = \frac{213s^2+s}{213s^2+1.0726s+0.0363} \qquad (2.36)$$

Fig. 2.44 and 2.45 show the bode plots of the disturbance rejection and sensitivity transfer functions respectively, generated using the bode command in MATLAB. The disturbance rejection transfer function clearly attenuates disturbances of all frequencies. On the other hand, the sensitivity transfer function indicates that the closed loop control attenuates most low-

frequency signals, amplifies signals with frequencies in a narrow range (between 0.01 and 0.025 rad/sec) and passes signals with frequencies higher than 0.025 rad/s. Because noise is generally a high frequency phenomenon, the transfer function passes most noise. However, this does not pose an issue because the low pass filter described in section 2.4.3 already attenuates most signal noise. Thus, the controller demonstrates robustness along with successful reference tracking.

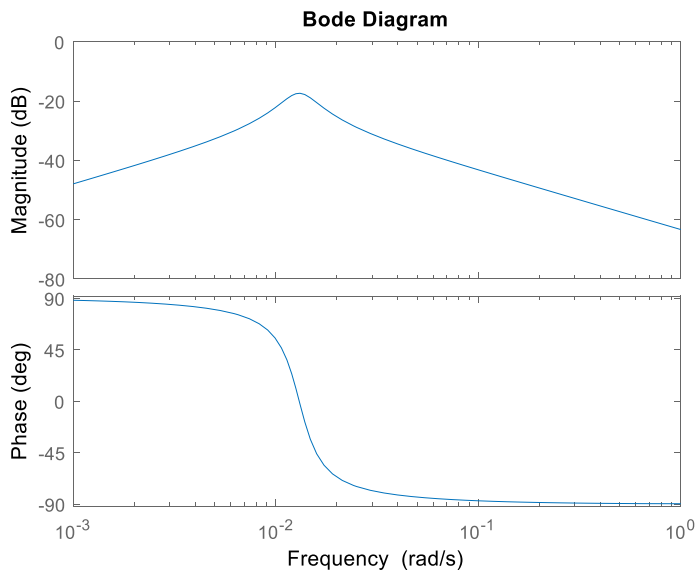

Figure 2.44 The bode plot of the disturbance rejection transfer function. The transfer function attenuates disturbances over all frequencies.

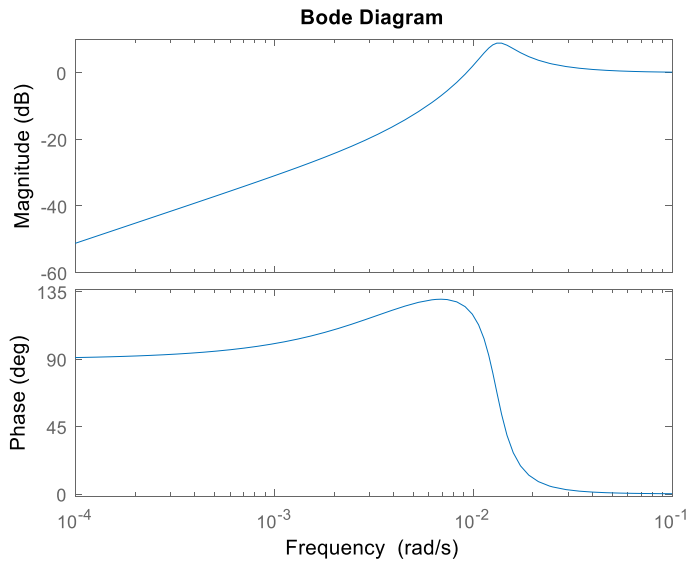

Figure 2.45 The bode plot of the sensitivity transfer function. The transfer function attenuates low frequency signals, amplifies a narrow range of signals with frequencies between 0.01 and 0.025 rad/s, and passes signals with frequencies higher than 0.025 rad/s. The transfer function passes most noise, but this is permissible because noise is already attenuated due to prior signal conditioning

2.6 Optimization Opportunities

Thus far, we have demonstrated the operation, modeling and validation of an integrated VCS in simulation and experimentally. Furthermore, we have demonstrated the ability of PI control to regulate the cabin temperature, which is the main function of the vehicle VCS. However, one question remains: are we regulating the cabin temperature optimally? That is, are we regulating the cabin temperature while consuming as little power as possible? As touched on in section 2.5, the VCS cooling capacity is a function of the evaporator air mass flow rate and the inlet and outlet air temperature differential, which is correlated with the compressor speed. Thus far, we have looked at cabin temperature regulation by only manipulating the compressor speed, while keeping the blower speed constant. However, is that fixed blower speed optimal? Is there another combination of compressor and blower speeds that yields the same cooling capacity while consuming less power? How can we determine this

optimal combination? These questions are addressed in chapter 3, by examining the use of extremum seeking control, an algorithm that can determine the optimal combination of inputs that meets required objectives while minimizing power consumption.

Chapter 3
Extremum Seeking Control

Thus far, we have demonstrated the ability to control a VCS in order to regulate the temperature of a space. This is done by continually manipulating the compressor speed using a PI controller while leaving other inputs constant. However, some important questions remain. Is the cooling capacity unique with respect to the inputs? That is, are there other sets of VCS inputs that yield the same VCS cooling capacity? If so, is there a unique set of inputs that also minimizes the power consumption?

Before analyzing VCS behavior, we need to understand the theory behind dynamic system optimization. We seek an optimization algorithm that can identify the system inputs that minimize a desired quantity. Gradient descent is a popular optimization algorithm that minimizes a function by moving in the direction opposite of the gradient value at a given point. However, we also need an algorithm that can perform optimization on a dynamic system with no prior knowledge of the system's performance function gradient. Extremum seeking control (ESC) is one such class of optimization algorithms with a wide range of academic and industrial applications. We examine its theoretical underpinnings along with its applications to VCS optimization.

3.1 Optimization via Gradient Descent

We first begin by discussing the fundamentals behind mathematical optimization, which is broadly defined as the selection of an element that best meets some chosen criteria. Optimization is commonly used to determine the input(s) to a function that minimizes or maximizes its value. In this thesis, we only consider function minimization. Gradient descent is the most common optimization algorithm, and is used extensively in machine learning, finance and engineering applications. Gradient descent converges to the minimum of a desired

function by evaluating the function's gradient at a given point, and then moving in the direction opposite to the gradient's value. In order to simplify analysis, we make the following assumptions:

1) The function of interest is globally convex: Mathematically, a function $J : \mathrm{R}^n \rightarrow \mathrm{R}$ is globally convex $\forall \theta_1, \theta_2 \in \mathrm{R}$, if the following equation is true:

$$\lambda \in [0,1]: \quad J(\lambda\theta_1 + (1-\lambda)\theta_2) \leq \lambda J(\theta_1) + (1-\lambda) J(\theta_2) \tag{3.1}$$

In other words, the above statement states that for any two points θ_1, θ_2, J evaluated at any convex combination of those two points should be no larger than the convex combination of the function values at the two points. Graphically, this means that if we connect two points on the function surface with a line, then the function must lie below this line between those points. A convex function in R^2 is depicted in Fig. 3.1. Furthermore, the convex function has a single, global minima θ^* such that $J(\theta^*) \leq J(\theta) \quad \forall \theta \in \mathrm{R}$. The gradient at the minimum value $\nabla J(\theta^*) = 0$.

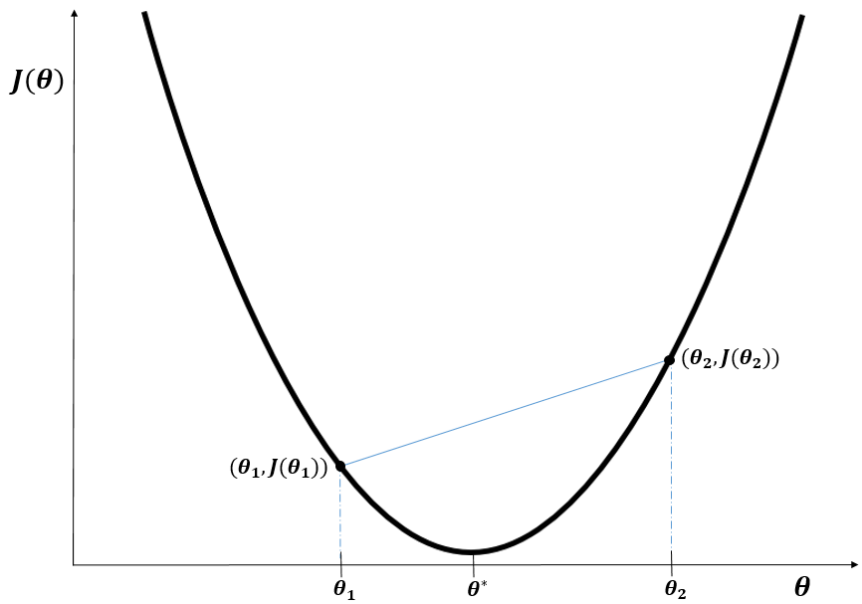

Figure 3.1 A graphical illustration of a convex function. Between any two coordinates $(\theta_1, J(\theta_1))$, and $(\theta_2, J(\theta_2))$ the function must lie below a line connecting these two points.

Optimization is often performed on convex functions because otherwise we may never converge to a final value. We desire that convexity holds globally so that we always converge to a function's lowest value rather than a local minima. An example of a non-convex function is shown in Fig. 3.2. The non-convex function shown has two local minima, and thus parts of the function lie above a line connecting two points on the function surface. Fig. 3.3 shows an example of a globally convex function in R^3.

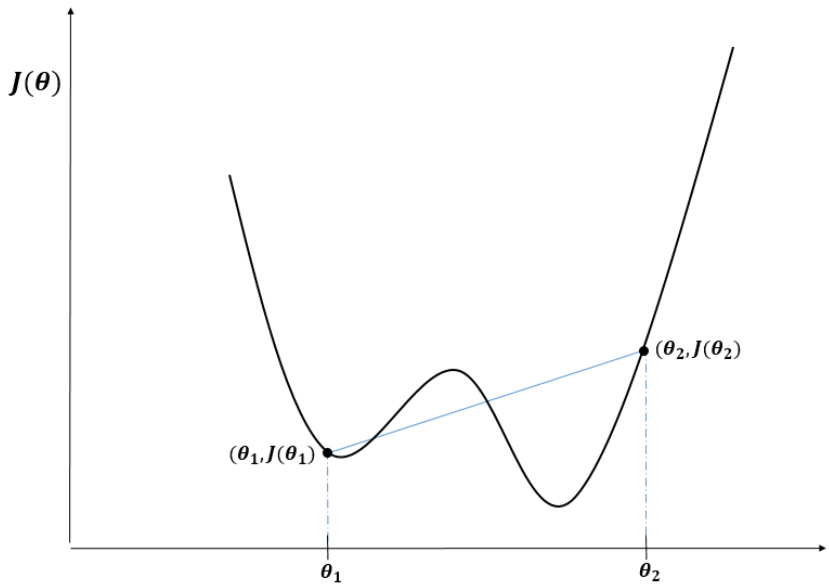

Figure 3.2 A non-convex function. A line drawn between two points on the function does not always lie above the function evaluated between those points.

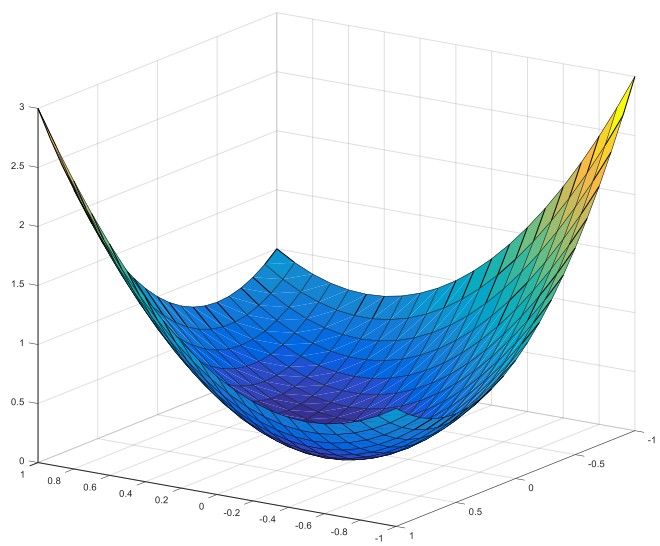

Figure 3.3 $f(x,y) = x^2 + xy + y^2$ **is a globally convex function in** R^3 **with a minimum at** $(0,0)$ **.**

2) We assume $J \in C^1$, or in other words, the first derivative of the function is continuous.

3) For now, we assume the performance function J is a static function (i.e. J does not vary with respect to time). We also assume to know the value of the gradient for all θ. We talk more about how valid this assumption is further below.

With these conditions in place, the gradient descent formula is given as following:

$$\dot{\theta} = -\Gamma \nabla J(\theta) \tag{3.2}$$

where Γ is a positive definite scaling matrix, and $\nabla J(\theta)$ is the gradient vector evaluated at θ. For the discrete scalar case, we can rewrite the gradient descent algorithm as follows:

$$\theta_n = \theta_{n-1} - c\left(\frac{dJ}{d\theta}\right)_{n-1} \tag{3.3}$$

where c is a positive scaling constant.

Fig. 3.4 illustrates the discrete gradient descent algorithm in action. For the first iteration, the value of the gradient at the initial value of θ is very negative. Using the equation above, this results in a large positive increase in the value of θ. For the second iteration, the gradient evaluated at the new value of θ is less negative than before, resulting in a smaller positive increase in θ. This process repeats until we converge to the minimum of J , where the gradient is zero.

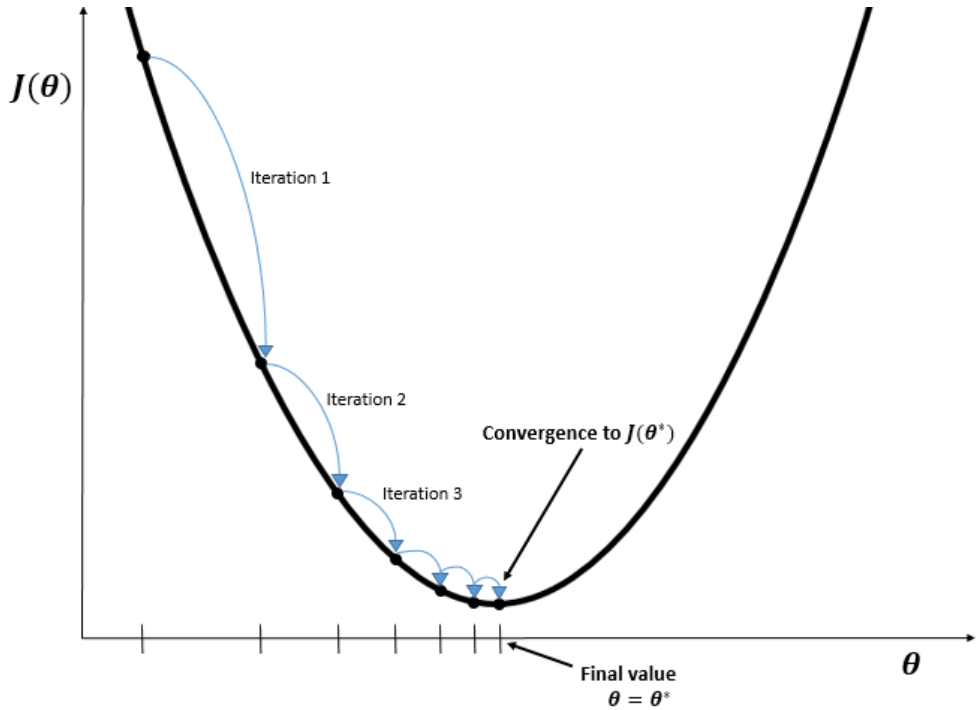

Figure 3.4 A visual example of a discrete scalar gradient descent algorithm applied to $J(\theta)$, a convex function in R^2. The algorithm successfully converges to the value of θ that minimizes J.

3.2 Gradient Estimation

From the example above, it is clear that once we know the function's gradient at a given point, converging to the optimal value is fairly straightforward. However, there are two issues to tackle before we can perform gradient descent on an actual system. The first problem is that we have no prior knowledge of the performance function, other than that we assume it to be convex. Therefore, we also have no knowledge *a priori* about the performance function gradient with respect to the system's input(s). The second problem is that we have assumed that the performance function is static (i.e. the function does not change with respect to time). This, however, is not true for dynamic systems. The output of a dynamic system can depend on the rate of change of the system's input, which has repercussions when performing gradient descent. If we descend towards the minimum too quickly, we could excite system dynamics that will throw off the gradient estimate and result in improper convergence. Thus, any changes

in the input must occur slowly enough that we do not generate a significant transient response in the system output. This is referred to as quasi-static or quasi-steady state behavior. Assuming a quasi-static system with respect to the input dynamics is referred to as a **time scale separation**.

Fig. 3.5 illustrates the differences between a dynamic, quasi-static and a static output of a second order dynamic system $Y(s)/U(s) = 1/(s^2 + 0.01s + 1)$ for $U(s) = [0, 2]$. The static system response is the plant's DC gain over the input range, with a constant gradient $dY/dU = 1$. The quasi-static response is generated by slowly varying the input 0 to 2 over 100 seconds, and follows the static response very closely. The dynamic response is generated by varying the input from 0 to 2 over 10 seconds. Changing the input this rapidly excites this system's dynamics, and the output does not match the static or quasi-static response. The gradient thus does not approximate the static response's gradient, precluding effective system optimization.

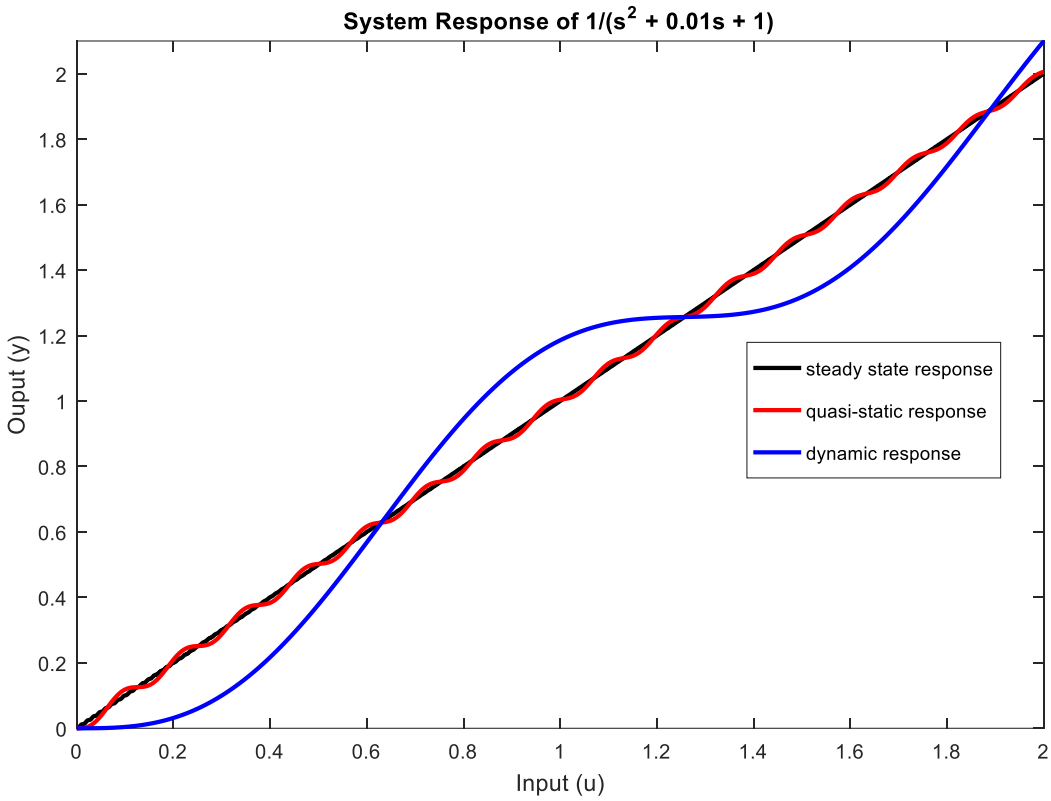

Figure 3.5 The static, quasi-static and dynamic responses of the second order transfer function $Y(s)/U(s) = 1/(s^2 + 0.01s + 1)$ for $U(s) = [0, 2]$.

Our optimization method must be able to identify the unknown performance function derivative at a given point and perform gradient descent without exciting the system's dynamics.

3.3 Extremum Seeking Control

Extremum seeking control is one such algorithm that accomplishes these goals. The basic algorithm works by slowly perturbing the system's input to generate a local gradient estimate of the quasi steady state performance function of a given nonlinear plant and uses it to perform gradient descent to determine the set of inputs that minimizes the function. ESC traces its origins to a paper written by LeBlanc in 1922 [3], and was used by industry practitioners in the 1950's and 1960's. After that, the popularity of ESC waned until the turn

of the century, when Krstic and Wang published a proof of stability for the classical perturbation ESC algorithm described below [4]. Since then, ESC has become highly popular in academia and industry. ESC has been used to optimize a wide range of systems, from maximizing photovoltaic power point tracking [5] to minimizing the power consumption of VCS [2], which we investigate in detail towards the end of this chapter.

3.3.1 The Basic Single-Variable Perturbation ESC Algorithm

Fig. 3.6 outlines the standard single-variable perturbation based ESC used to determine the optimal inputs to a general nonlinear plant $\dot{x} = f(x,\theta)$ that minimizes its associated convex performance function $y = J(x)$. For simplicity, we analyze the algorithm in continuous time, but later chapters will address discrete implementation of this algorithm in software. The derivation presented below was sourced from [31].

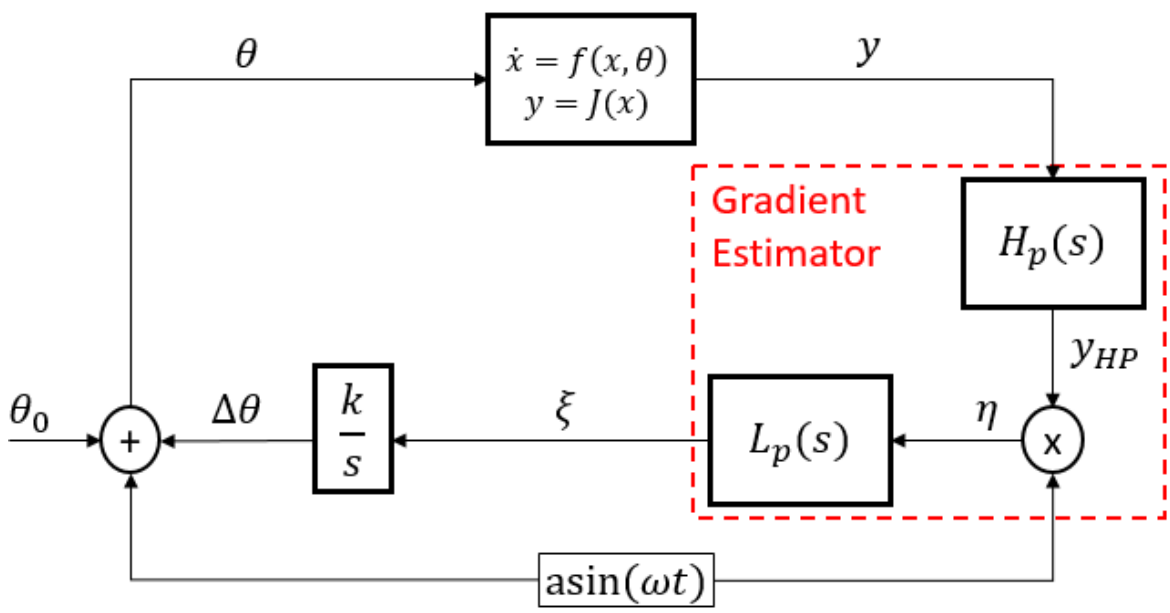

Figure 3.6 A block diagram of the classical perturbation ESC algorithm.

The first step of perturbation ESC is to inject a low amplitude and low frequency dither signal, commonly a sinusoid $a\sin(\omega t)$, into the nominal plant input θ_0. This generates a quasi-

steady state sinusoidal output that contains information on the local performance function gradient. The plant output is approximated by the following equation:

$$y(t) \approx y_0 + J'a\sin(\omega t) \tag{3.4}$$

where y_0 is the nominal plant output, and J' is the local performance function gradient $\frac{dJ}{d\theta}$. Note that we assume we are perturbing the system slowly enough that the phase shift induced by the plant can be neglected. Next, the plant output passes through a high pass filter $H_p(s) = \frac{s}{s+\omega_{HP}}$, ω_{HP} being the cutoff frequency, that removes the DC part of the response. The high pass filter output, $y_{HP}(t)$, is given by the following equation:

$$y_{HP}(t) \approx G_{HP} J' \sin(\omega t + \phi_{HP}) \tag{3.5}$$

where G_{HP} is the high pass filter gain, given by $G_{HP} = mag(H_p(j\omega)) = \frac{1}{\sqrt{1+\left(\frac{\omega_{HP}}{\omega}\right)^2}}$, and ϕ_{HP} is the filter phase shift, given by $\phi_{HP} = \tan^{-1}\left(\frac{\omega_{HP}}{\omega}\right)$. The next step is to multiply the output by a demodulation signal $a\sin(\omega t)$ to extract gradient information from the signal. After performing some trigonometric manipulation, the product of the two signals is given by the following equation:

$$\eta(t) \approx G_{HP} J' \frac{a^2}{2}\left[\cos(\phi_{HP}) + \cos(2\omega t + \phi_{HP})\right] \tag{3.6}$$

Although the integrator attenuates the high frequency component of $\eta(t)$, we pass the output through a low pass filter $L_p(s) = \frac{\omega_{LP}}{s+\omega_{LP}}$,where ω_{LP} is the cutoff frequency, to further improve gradient estimation. Plugging in the formulas for G_{HP} and ϕ_{HP}, we get the following low pass filter output.

$$\xi(t) \approx G_{HP} J' \frac{a^2}{2}\left[\cos(\phi_{HP})\right] = \frac{J' a^2}{2\sqrt{1+(\omega_{HP}/\omega)^2}} \cos(\tan^{-1}(\omega_{HP}/\omega)) \tag{3.7}$$

Using the trigonometric identity $\cos(\tan^{-1}\beta) = \frac{1}{\sqrt{1+\beta^2}}$, we rewrite the above equation as follows:

$$\xi(t) \approx J'\left[\frac{a^2\omega^2}{2(\omega^2+\omega_{HP}{}^2)}\right] \tag{3.8}$$

Armed with an estimate of the local performance function gradient, we can now perform gradient descent. We scale and integrate the derivative accordingly and obtain $\Delta\theta$. We add this term to the nominal input such that the new input to the plant moves towards the optimal value. Finally, we inject the sinusoid $a\sin(\omega t)$ back into the input to repeat the process all over again. The new input to the plant is given as following:

$$\theta(t) = \theta_0 + \Delta\theta + a\sin(\omega t) \tag{3.9}$$

3.3.1.1 Choosing Algorithm Parameters

For perturbation ESC to work well, we need to carefully choose the algorithm parameters. We consider the dither signal, filter design and integrator gain.

Dither signal: As mentioned previously, the choice of the dither signal is very important. A sinusoidal dither is most common in literature, but other forms of dither signals such as square waves have been utilized successfully [17]. The dither signal must have a small amplitude relative to the plant gain and have a frequency slower than the dominant plant dynamics to ensure quasi-static (or quasi-steady state) performance. Varying the input too quickly may excite the plant which is problematic because the resulting system output may not be indicative of steady-state plant performance, and we may erroneously converge to suboptimal inputs.

Filters: Filter design is also an important part of building an effective ESC. It is important for the high pass filter to attenuate the DC signal component, while passing the

sinusoidal plant response. On the other hand, we wish for the low pass filter to attenuate oscillating components while passing the low frequency gradient estimate. In general, literature suggests that setting $\omega_{HP} = \omega_{LP} = \omega$ is sufficient [22].

Integrator gain: The integrator gain k determines the rate of adaption. Too high of a gain, and the system dynamics may be excited, while too small of a gain results in sluggish performance. For minimization, the gain should be negative, while for maximization, the gain should be positive. The integrator gain is often user-tuned through trial-and-error.

3.3.2 Shortcomings of Perturbation ESC

Perturbation ESC has been used to effectively optimize the behavior of a wide range of dynamic systems. However, there are some shortcomings of this approach. The first issue is that perturbation ESC has a large number of tunable parameters, from the integrator gain to the filter cutoff frequencies to the sinusoid's characteristics. In order to achieve optimal convergence, the user must tune all of these parameters perfectly which is inherently difficult. The second problem is that using an oscillating dither signal results in practical, but not asymptotic stability around some optimal point. Although there are algorithms that asymptotically reduce the dither amplitude as one gets closer to the minimum value, this doesn't address the other shortcomings of perturbation ESC. Lastly, perturbation ESC induces a sinusoid in the system's output at a frequency slower than the plant dynamics. This signal is then filtered and averaged when generating a gradient estimate. However, averaging the effect of the perturbation on the system induces a second, slower time scale separation in the optimization procedure. This is problematic, especially when performing ESC on vapor compression systems due to their inherently slow dynamic behavior [9]. Convergence to the optimal value could take as long as several hours, which is not ideal. We thus seek a simple, yet effective ESC that doesn't use a slowly varying perturbation to generate a gradient estimate. One such method is referred to as least squares based extremum seeking and is described below.

3.3.3 Least Squares Based Extremum Seeking

Least squares based extremum seeking is an ESC algorithm developed by Hunnekens et al. in 2014 [10]. The algorithm's block diagram is outlined below.

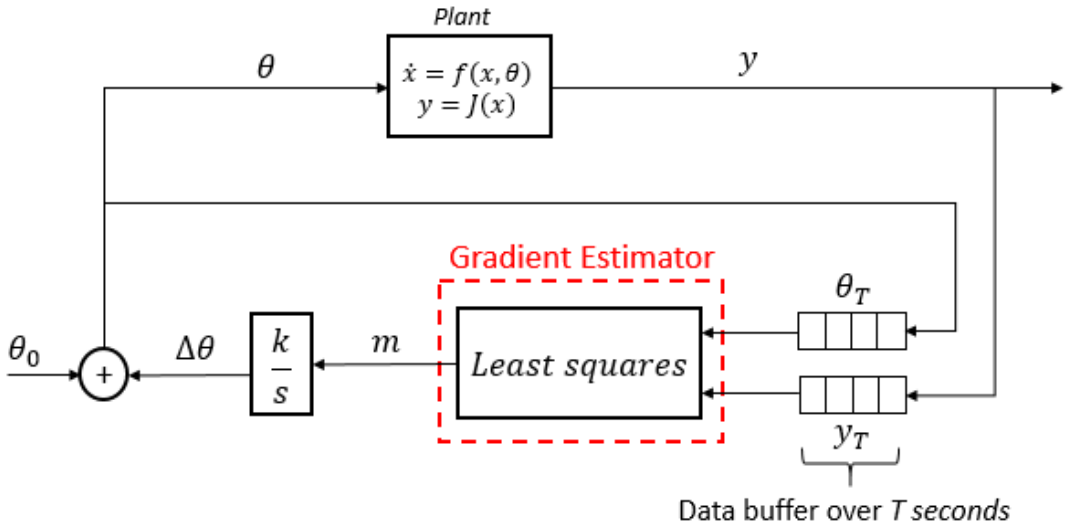

Figure 3.7 A closed loop block diagram of the algorithm described in [10].

This algorithm works by continually generating a first order least squares fit on a moving buffer of past performance data over the last T seconds. This least squares fit contains an estimate of the performance function gradient that can be used in a gradient descent algorithm to converge to the optimal value. Because no perturbation is utilized in this algorithm, the controller can achieve asymptotic stability with one less time scale separation than perturbation ESC, leading to potentially faster convergence [10]. We detail the algorithm's fundamentals below.

3.3.3.1 Least Squares

The backbone of this algorithm is the least squares method. Least squares is a well-known method used to approximate the solution of an overdetermined system. We use the ordinary least squares method, where the approximate solution to an overdetermined system is a linear function of the form $y = m\theta + b$. This is also referred to as the line of best fit. We wish to determine the coefficients m and b that minimizes the sum of the distances squared between

the individual points and the approximate linear solution. This is done by projecting the vector of data points onto the subspace spanned by the linear function. An example of this is shown in Fig. 3.8.

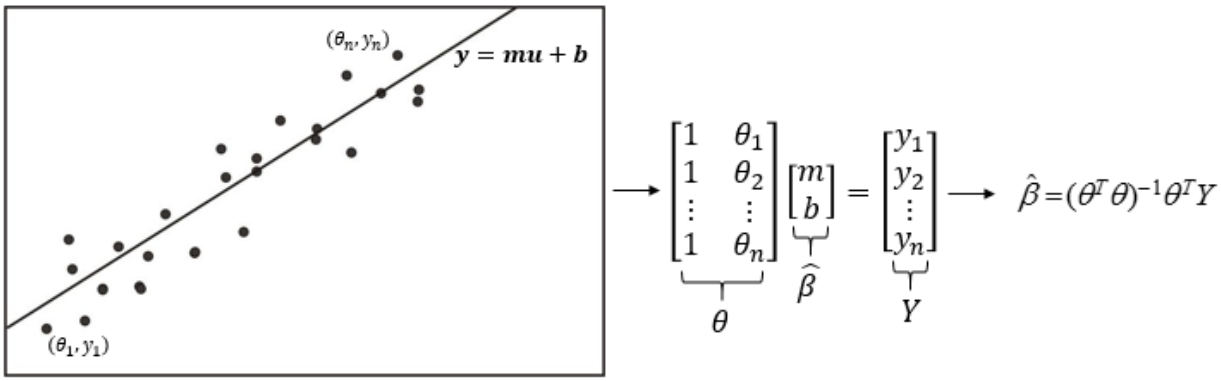

Figure 3.8 An example of ordinary least squares applied to a set of data points $(\theta_1, y_1), (\theta_2, y_2) ... (\theta_n, y_n)$. We organize the data points into corresponding θ and Y matrices, and then use a matrix projection operator $\hat{\beta} = (\theta^T\theta)^{-1}\theta^T Y$ to determine the linear coefficients that minimizes the sum of the squared distances between data points and the linear approximation.

The least squares method easily lends itself to extremum seeking. If we have some available data on the performance function in some region over time *T*, we can easily generate a least squares fit on the data and obtain a gradient estimate, which is simply the parameter *m* from the last example. With this gradient estimate, we can perform gradient descent and determine a more optimal input value. We continuously repeat this procedure until we converge to the optimal value that minimizes the performance function.

An important distinction between perturbation ESC and least squares ESC is that least squares ESC requires initialization. If the system is at steady state when the least squares ESC is activated, then the calculated gradient will be zero and the system will not adapt over time. However, if we set $\dot{\theta} = \frac{P}{T_i}$, where P is a change in input value over a time period T_i, and collect the resulting system power consumption over that period, we allow the controller to calculate

performance function gradient with respect to some input range prior to activating the controller (allowing it to send an input adjustment signal to the plant). Once it is activated, the controller starts with a good initial estimate of the performance function gradient and adjusts the input accordingly. This effectively sets the algorithm "in motion" and it will converge to the optimal value over time.

3.3.3.2 Choosing Algorithm Parameters

Unlike perturbation ESC, least squares ESC only has three parameters to tune: the integrator gain k, the time buffer length T and the controller initialization value P. Like before, we want the gain to be small enough to not excite the system dynamics but also be large enough so that behavior isn't sluggish. We want the buffer length T to be small enough to yield a somewhat local least squares fit on a section of the performance function. On the other hand, we don't want T to be so small that the controller becomes susceptible to noise or disturbances. And finally, we want the initialization value P to be small enough such that it doesn't change the input too quickly, but large enough to generate a robust estimate of the performance function gradient.

3.3.3.3 Issues with Least Squares ESC

Despite the simplicity and ubiquity of the least squares approach, there are a few intrinsic problems with the algorithm. We detail these below.

Computational expense: An issue with the current implementation of least squares is that computing the least squares solution is expensive, especially for large data buffers. Performing a matrix inverse on large data matrices isn't a problem for a computer, but for smaller embedded controllers, this may prove to be a significant barrier to implementation. An ideal algorithm would not use costly computations every iteration.

A lack of persistence of excitation: Another issue with the algorithm presented above is that estimating the gradient is only possible if there is enough data to generate a linear fit. This problem is avoided by initializing the controller as mentioned before, but if the values of θ in the buffer were instead all the same or very close to one another at some point in time, then the linear fit could be undefined or erroneously large in magnitude. This could happen,

for example, when we get very close to the optimal input value. Furthermore, if the gradient measured at some time is zero, then the gradient descent algorithm would essentially be "turned off" for all future time. If the optimal function value changes over time, as is true for many real life dynamical systems, then gradient descent may not converge to the optimal value. Therefore, we need a persistently excited system to generate sufficiently data for all time *t* such that we can always generate accurate gradient estimates. For an arbitrary input signal $u(s)$, persistence of excitation is defined as following:

$$\int_{t}^{t+\Delta t} u^2(s)ds \geq \alpha_0 \Delta t \tag{3.10}$$

where Δt is an arbitrary time range, and α_0 is a positive constant.

With these problems in mind, we seek an algorithm that can generate a gradient estimate as quickly as least squares ESC can while minimizing computation cost and guaranteeing persistence of excitation. Recursive least squares is one such algorithm as is described below.

3.3.4 Recursive Least Squares (RLS) ESC

Recursive least squares ESC (also referred to as time varying ESC in the literature [32]), is an advanced, discretely implemented controller that utilizes a recursively formulated least squares algorithm to identify the performance function gradient and converge to the optimal value. This approach also entails one less time scale separation than classical perturbation ESC, which allows for fast convergence.

3.3.4.1 Recursive Least Squares (RLS) Gradient Estimation with Forgetting Factor

In traditional least squares, when receiving a new data point (θ_n, y_n) at time *n*, we have to recalculate the least squares solution for all *n* data points to determine the gradient *m*. This is an extremely expensive computation, especially when calculating the matrix inverse $(\theta^T\theta)^{-1}$ for each time step. On the other hand, when receiving a new data point (θ_n, y_n) at time *n*, the RLS algorithm can recalculate the new performance function gradient only using the current

data and data from the previous time step, *n-1*. A detailed derivation of the RLS algorithm can be found in [23]. The RLS algorithm is as follows:

$$\theta_n = new\ input\ value \tag{3.11}$$

$$y_n = new\ output\ value \tag{3.12}$$

$$V_n = \frac{1}{\lambda}\left(V_{n-1} + \frac{V_{n-1}\theta_n^T \theta_n V_{n-1}}{1+\theta_n V_{n-1}\theta_n^T} \right) \tag{3.13}$$

$$\gamma_n = V_n \theta_n^T \tag{3.14}$$

$$e_n = y_n - \theta_n \hat{\beta}_{n-1} \tag{3.15}$$

$$\hat{\beta}_n = \hat{\beta}_{n-1} + \gamma_n e_n \tag{3.16}$$

$$m_n = \hat{\beta}_n(1) \tag{3.17}$$

λ is the forgetting factor, a value between 0-1 that exponentially reduces the weight of previous data points. A value closer to 0 means that we have a more local gradient fit, but too low of a value makes the algorithm susceptible to noise. A value closer to 1 is more robust to noise but the gradient estimate is often less accurate.

A new variable introduced in the algorithm is V which is simply the projection operator $(\theta^T\theta)^{-1}$ rewritten using the Woodbury Matrix identity. This identity saves us from re-computing the matrix inverse every iteration. Again, more detail on this can be found in [23]. In order to operate the algorithm recursively, we need to continually feed the values V and $\hat{\beta}$ back into the algorithm once we have determined the new gradient value. We initialize the recursive algorithm by defining V_0 and $\hat{\beta}_0$. In many cases, setting V_0 as the identity matrix and $\hat{\beta}_0$ as a column vector of one's prior to activating the controller is sufficient. However, generating good initial values of V_0 and $\hat{\beta}_0$ can significantly accelerate gradient estimation. This can be done by using the initialization approach used in traditional least squares, by setting

$\dot{\theta} = \frac{P}{T_i}$ for some period of time T_i allowing the algorithm to determine the gradient value and then activating the algorithm, allowing it to adjust the input.

With RLS being our new gradient estimator, the RLS ESC algorithm is illustrated as follows:

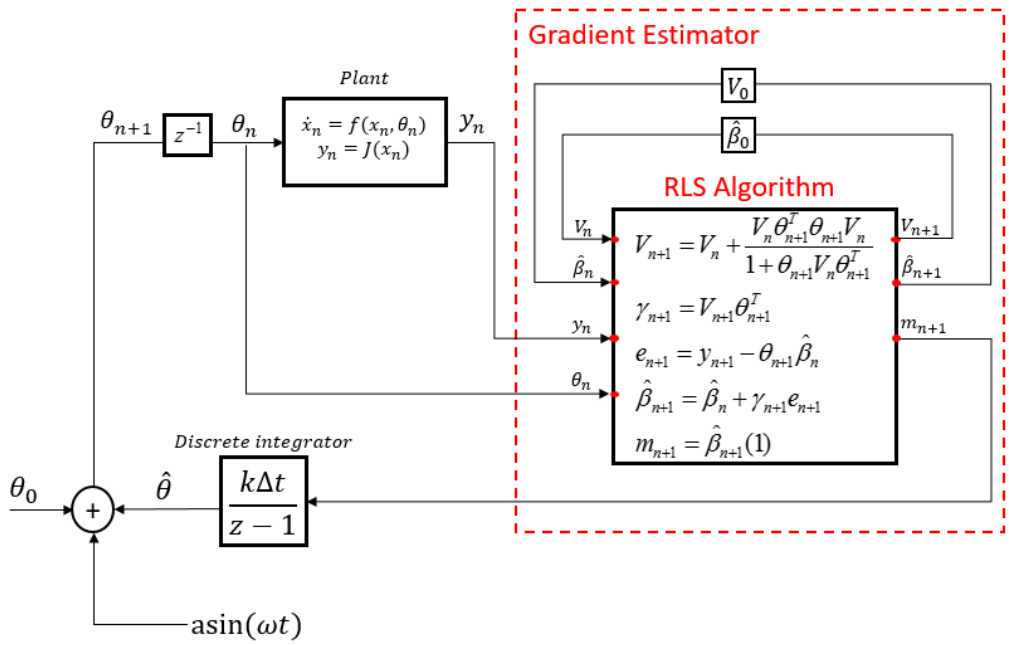

Figure 3.9 A block diagram detailing the implementation of discrete RLS ESC on a sample plant.

Persistence of excitation is achieved in this framework by injecting a small amplitude sinusoid $a\sin(\omega t)$ into the function input, as seen above, so that the input always varies by some amount, but not large enough to disrupt near-asymptotic converge to the optimal value. The input to the system is thus defined as $\theta_{n+1} = \theta_0 + \hat{\theta}_n + a\sin(\omega t)$.

It is clear that RLS ESC has a number of advantages over perturbation ESC. RLS ESC has theoretically fast performance while also incurring smaller computational costs than ordinary least squares ESC. Furthermore, the user only has to tune three algorithm parameters: the gain k, the forgetting factor λ and the initialization constant P. This makes RLS ESC an attractive optimization algorithm. However, before we implement this algorithm on the

simulated or experimental VCS described in Chapter 2, we first need to parse the VCS literature to determine whether ESC is a viable optimization strategy for this class of systems, as well as to determine the specific control architecture used to achieve beneficial results.

3.4 ESC Applications to VCS

The first substantive research paper written on the use of ESC on vapor compression systems was written by Burns and Laughman in 2012 [2]. Burns and Laughman observed that the cooling capacity of VCS was not unique with respect to its inputs; a number of different combination of evaporator fan and compressor speeds yielded the same cooling capacity. Furthermore, [2] determined that the power consumption, the performance metric of interest, was convex with respect to the input combinations. In other words, there was a unique combination of inputs that yielded a given cooling capacity while also minimizing the system's power consumption. Fig 3.10 is a diagram from [2] illustrating power convexity with respect to the VCS input space.

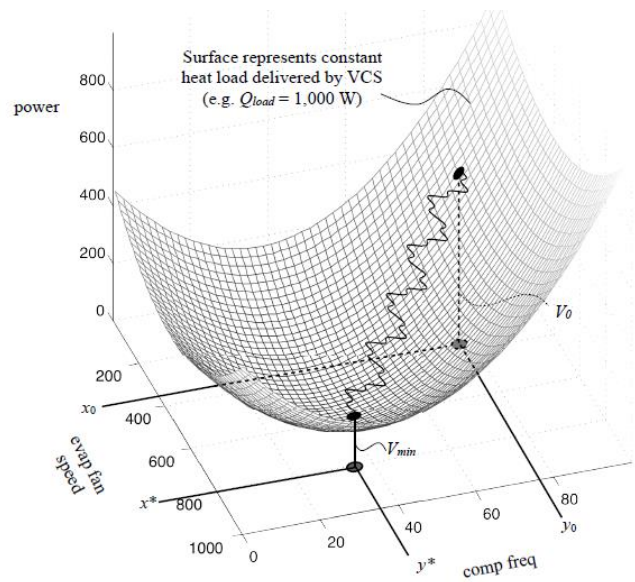

Figure 3.10 A graphical representation from [2] of power convexity with respect to the VCS input space. The convex function represents a constant VCS cooling capacity of $\dot{Q} = 1000W$. The goal of ESC is to minimize this function by going from a suboptimal input combination V_0 to $V_{\min}$.

Burns and Laughman then implemented a perturbation-based ESC on an experimental VCS setup that reduced the VCS power consumption by from 750W to 400W while maintaining a zone at a given reference temperature [2]. The ESC was added to an existing closed loop PI control architecture that maintains the zone temperature by manipulating the compressor speed (the same PI architecture was used to regulate cabin temperature in the previous chapter). Fig. 3.11 details the control architecture used in this paper. The stabilized VCS refers to PI control applied to the VCS such that it tracks a reference temperature. The ESC forms the "outer loop" of this control scheme as it slowly perturbs the stabilized system to search for the energy minimum. Since we assume our performance function is quasi-static, the PI control must be able to stabilize the system quickly in response to disturbances and changes in evaporator fan speeds without generating large transients that could throw off the ESC. Note that the condenser fan speed remains a constant value.

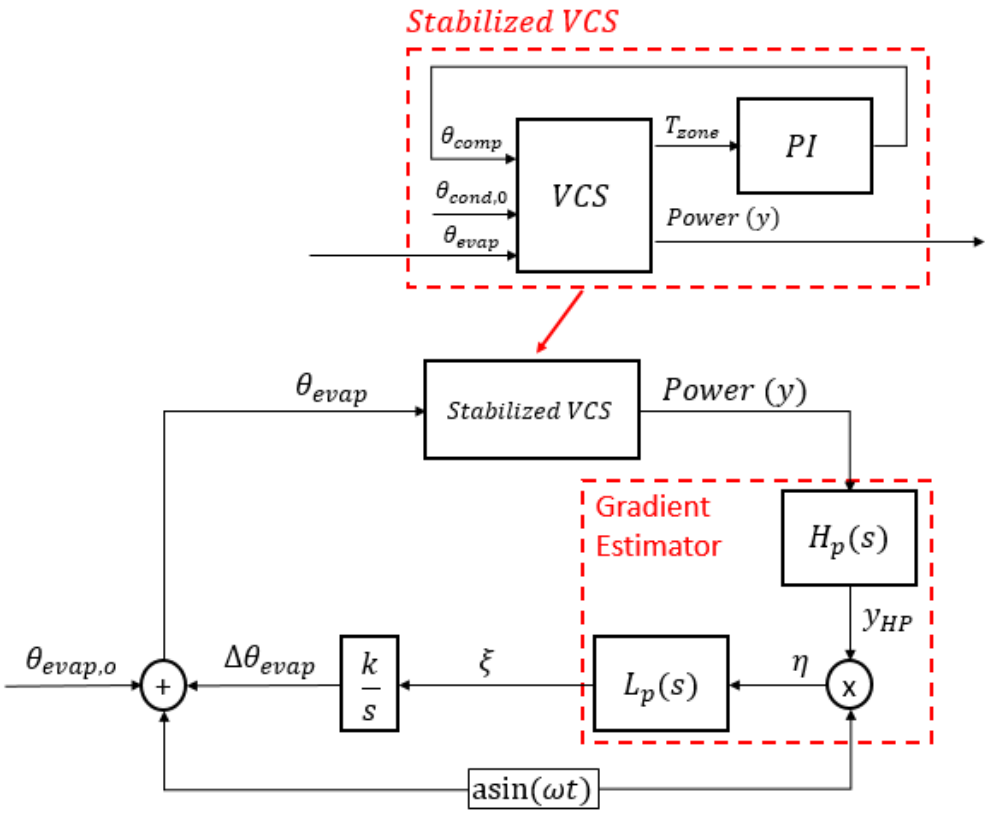

Figure 3.11 The control architecture utilized by [2].

The ESC slowly perturbed the evaporator fan speed that was previously held constant. When the fan speed was increased, the cooling capacity also increased, resulting in a drop in zone temperature. To track the temperature set point, the PI controller decreases the compressor speed until the zone temperature tracked the temperature set point again. This new combination of compressor and evaporator fan speed yielded a lower power consumption than the prior input combination. The ESC interpreted the increase in fan speed with a decrease in power, generating a corresponding estimate of the cost function gradient and further increased the evaporator fan speed until the power consumption reached a minimum at 400W, 35% lower than its original value [2]. Fig 3.12 shows the experimental results from [2] showing the two actuator speeds converging to their respective optimal values while maintaining a constant zone temperature.

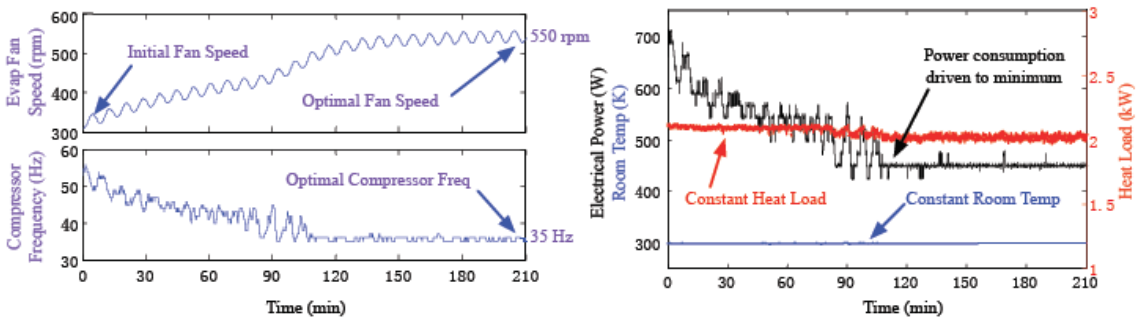

Fig 3.12 Experimental results from [2]. Over the course of two hours, the ESC determines an optimal combination of compressor and evaporator fan speeds that minimizes the power consumption. Even as this process occurs, the VCS successfully keeps the room temperature at a pre-determined constant value.

Since this paper was published, ESC has become widely utilized in the VCS controls community, with a proliferation of new algorithms and applications. ESC has been used to optimize VCS subcooling [22], maximize the COP of transcritical CO2 heat pumps [24], and minimize the power consumption of chilled water systems [25], to name a few examples. Advances in ESC algorithm design have also been leveraged to further improve VCS performance. Multivariable ESC is one such example of this, where the controller

simultaneously modulates more than one actuator. For VCS's, this often entails modulating both the evaporator and condenser fans, whereas traditional ESC modulates one fan speed while often holding the other constant. Some examples of multivariable ESC use on a VCS can be found in [26] or [27]. The use of RLS ESC (also referred to in the literature as time-varying ESC) also extends to VCS. Burns et al. applied a time-varying ESC approach on a VCS by modulating the evaporator fan speed while using the compressor to control a zone temperature, an architecture similar to his paper in 2012 [9]. Using time varying ESC resulted in convergence to optimal parameters in under an hour, as opposed to two hours for perturbation ESC [9]. These experimental results validate the use of ESC on VCS, and experimentally validate the faster convergence of time varying ESC over perturbation ESC.

3.5 Optimization of No-Idle VCS

ESC has been established as a viable and effective optimization strategy through theoretical analysis and experimental validation performed on a wide range of vapor compression systems. The next step, naturally, is to determine whether these results also extend to no-idle battery operated VCS such as the NITE. Can ESC minimize the power consumption of these types of systems, and maximize their battery life? We explore the implementation of ESC on the simulated system in Chapter 4, and implementation of ESC on the experimental setup in Chapter 5. In particular, we examine the use of single variable perturbation ESC, least squares and RLS ESC on these systems, as all three of these techniques are simple to implement and the literature has established their efficacy.

Chapter 4

Extremum Seeking Control on the Simulated Integrated NITE System

After detailing the simulated and experimental integrated NITE system in Chapter 2, and analyzing ESC and its applications in Chapter 3, we now have the tools to implement ESC on the simulated and physical system. This chapter will detail implementation in simulation, while the following chapter will discuss experimental implementation.

The basic VCS control scheme presented at the end of chapter 2 was PI control regulation of the cabin temperature by compressor speed modulation. We utilize this control architecture because, as outlined in section 2.5, the compressor speed directly affects VCS cooling capacity. However, the evaporator fan speed, an actuator that also directly influences the cooling capacity, remained constant. We now know from literature that VCS power consumption is convex with respect to these two inputs for a given cooling capacity [2]. That is, there exists an optimal combination of these two actuators that simultaneously minimizes power consumption and achieves desired temperature regulation. For our system, this means that determining the optimal input combination could extend battery life while meeting passenger cooling requirements. Researches have utilized various types of ESC on VCS in conjunction with PI control architecture to meet these two goals, as discussed towards the end of Chapter 3. Furthermore, research suggests that RLS and least squares (LS) ESC may converge to these inputs faster than perturbation ESC (P-ESC) can. We wish to determine whether these results hold true for the NITE system as well. In this chapter, we discuss the design, implementation and analysis of three different ESC algorithms (P-ESC, LS-ESC and RLS-ESC) on the simulated system.

4.1 NITE System Power Convexity

ESC minimizes a function with respect to its inputs as long as this function is convex. Although [2] proved that VCS power is convex with respect to inputs, it is a useful to map out the NITE's performance function ourselves prior to ESC implementation. Generating a performance function map will allow us to verify the convexity of the function and to learn the location and value of the function's minimum. This way, we can know with certainty whether the algorithms tested in this chapter converge to the most optimal point.

To determine this, we consider the following scenario: We have a modeled truck cabin initially at 35°C that we want to cool to 21°C. We have two vehicle occupants, an ambient temperature of 35°C and an air recirculation of 90%. The rest of the integrated model is parametrized as outlined in Chapter 2 for a sleeper cabin. Using basic PI control, we can modulate the compressor speed to pull down the cabin temperature to the desired value from *T= 0 to 3000s*, while keeping the evaporator fan speed fixed at some suboptimal value (note that the condenser fan is always fixed at 70PWM in this thesis since we are considering only single variable ESC). Once the cabin temperature reaches steady state, at *T=3000s* we start to slowly ramp the evaporator speed up at a rate of 0.001PWM/sec until *T=30000s*, as seen in Fig. 4.1. Increasing the blower speed increases the cooling capacity, which lowers the cabin temperature, resulting in the PI controller decreasing the compressor speed in order to track the set point, as also seen in Fig. 4.1. This new combination of compressor and evaporator speeds yields a corresponding power consumption, shown in Fig. 4.2. Because the evaporator speed changes slowly, this procedure generates a quasi-steady state map of the system power consumption with respect to the compressor and blower speeds. We performed this procedure on the simulated system, and the power consumption was found to be mostly convex with respect to the two inputs; there was a unique combination of compressor and evaporator fan speeds, 1565rpm and 126-128PWM respectively, which minimized the total power consumption to roughly 523W while maintaining the vehicle cabin temperature at 21°C. Fig. 4.3 illustrates the convex relationship between power and system inputs. Fig. 4.4 shows the cabin temperature over the course of the mapping procedure, and Fig. 4.5 depicts the PI controller implemented in simulation used to maintain the cabin temperature at the set point.

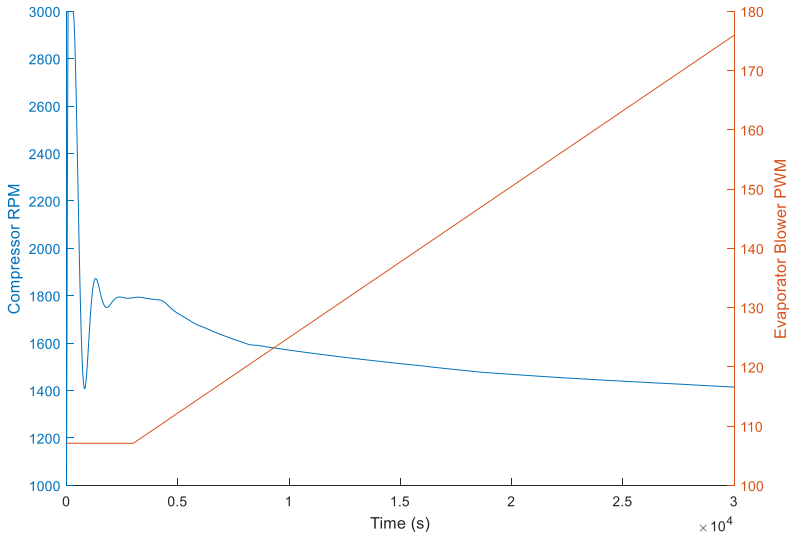

Figure 4.1 The change in compressor and blower speeds over the course of the mapping procedure. The large initial compressor RPM transient is a result of pulling down the cabin temperature to the set point. As the blower speed increases, the compressor speed decreases in order to maintain a constant cabin temperature.

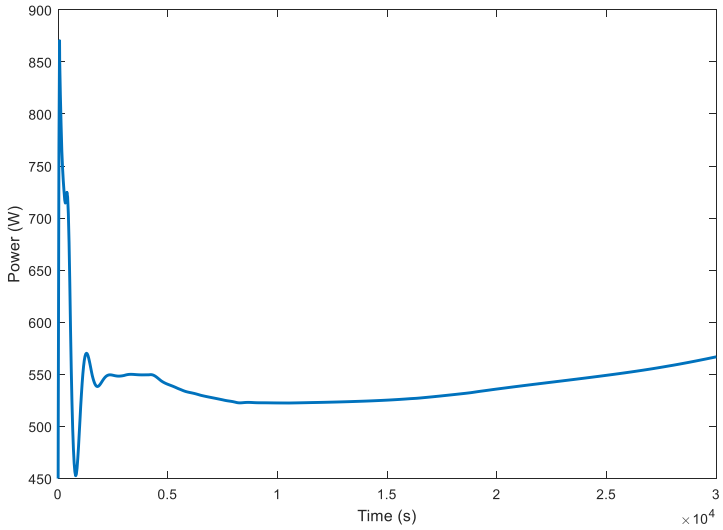

Figure 4.2 The total VCS power over time. The initial power transient is due to the high compressor speeds. From 3000 to 30000 seconds, the power curve can be approximated as a convex function.

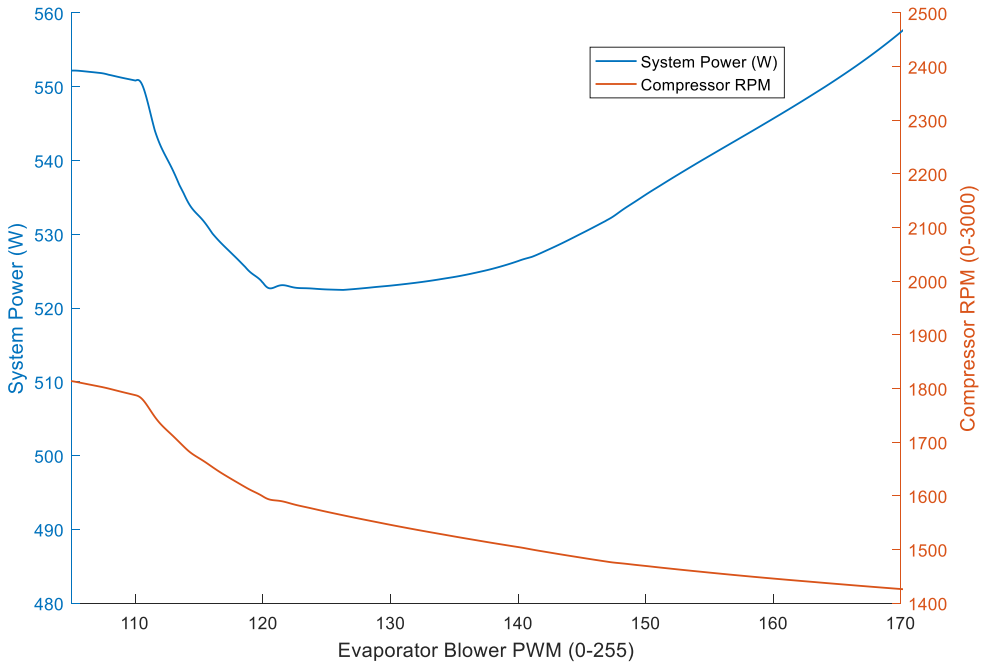

Figure 4.3 The quasi-static system power curve with respect to the evaporator blower speed. As the blower speed increases, the compressor speed decreases in order to maintain a constant vehicle cabin temperature. The total power consumption is minimized around a blower speed of 126-128 PWM and a compressor speed of 1565 RPM.

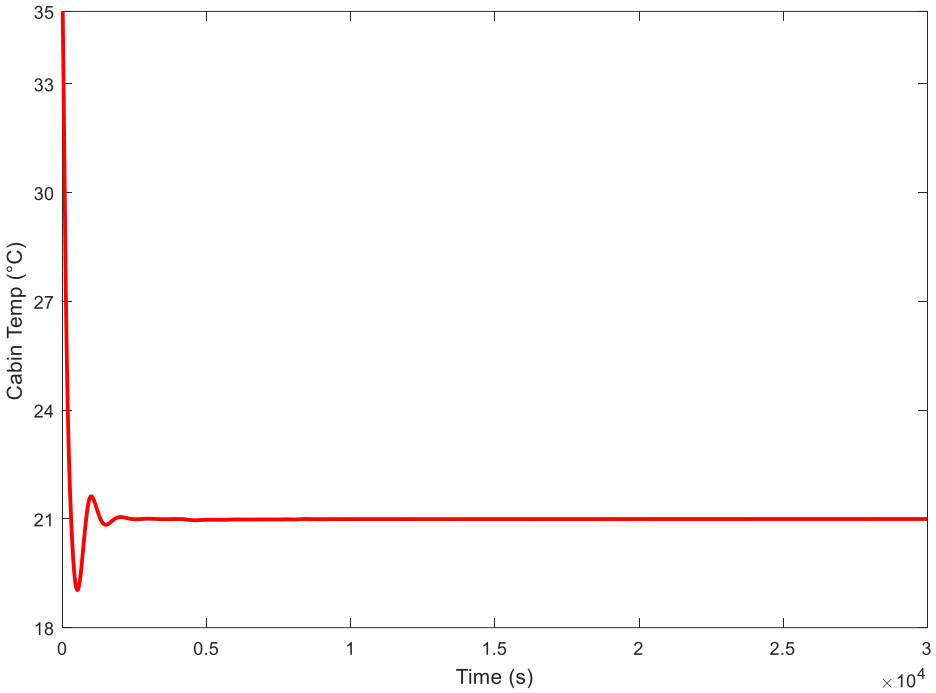

Figure 4.4 The cabin temperature over time, starting from an initial temperature of 35°C. As the blower speed increases during the mapping process, the PI controller decreases the compressor speed to keep the cabin temperature at 21°C.

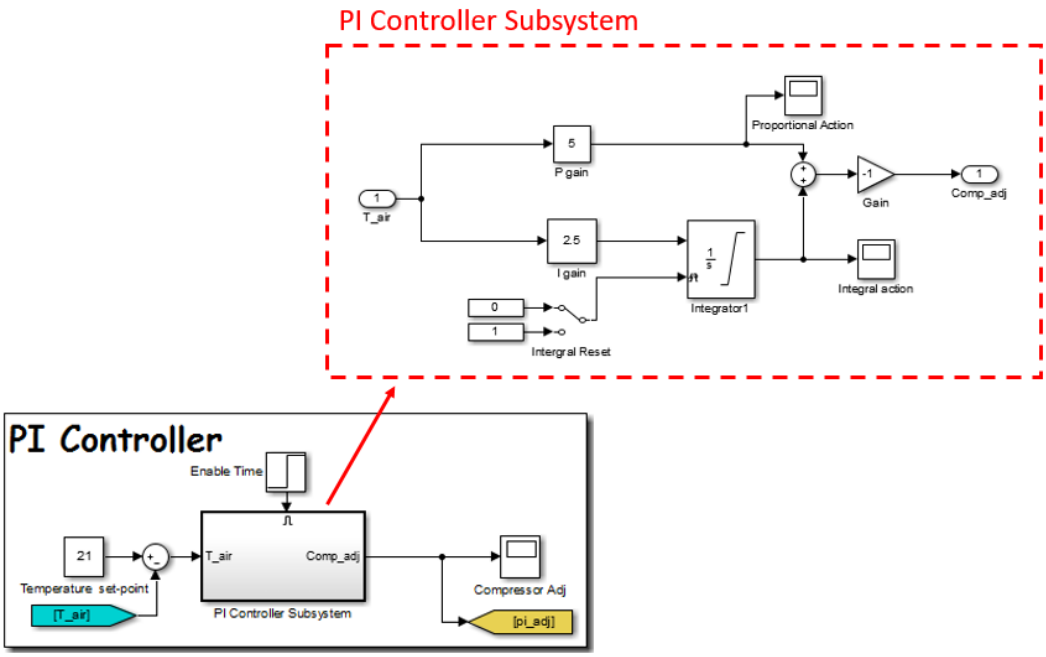

Figure 4.5 The PI controller utilized in simulation.

4.1.1 A Note on the Condenser Fan Speed

The condenser fan speed was held constant in the previous example, but what if we repeated the procedure above for the condenser fan instead while holding the evaporator speed constant? Would this also yield a convex relationship? The answer to this is no. As seen in Fig. 4.6, the power is not convex with respect to the condenser fan speed (note that for the NITE's condenser fan, lower PWM corresponds with a higher speed). Therefore, our intuition in using the evaporator blower as the primary input variable of interest is correct. Note that, as mentioned in the previous chapter, some researchers have used the condenser fan for optimization purposes, but we assume it remains a constant. NITE manufacturers set the condenser fan speed at a fixed 70PWM in operation.

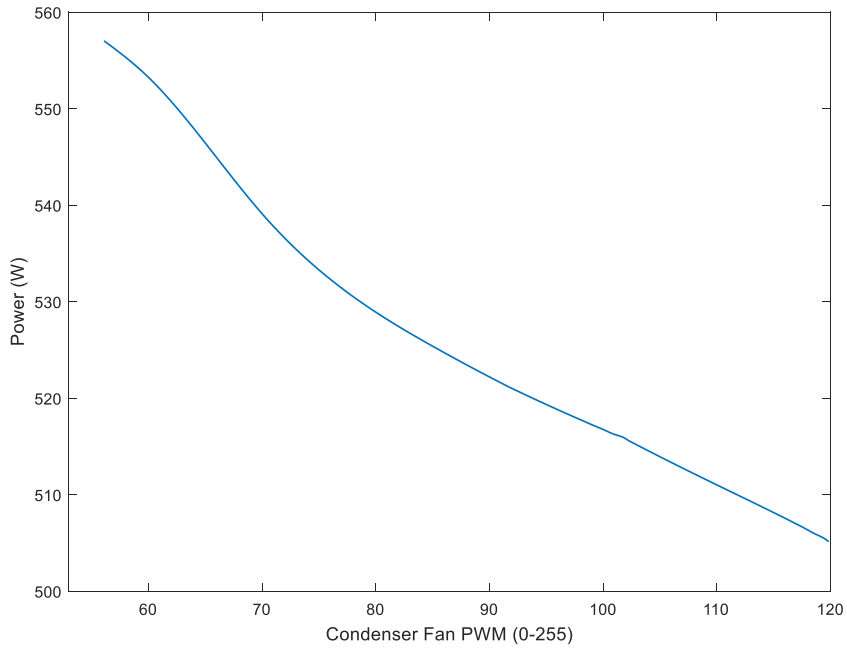

Figure 4.6 Total power consumption with respect to the condenser fan speed is not convex. The evaporator blower is held at an arbitrary constant 153 PWM.

4.2 Implementing ESC on the Simulated System

Having demonstrated power convexity, we can now utilize ESC techniques to identify and converge to input combinations that minimize power consumption and extend battery life. We implement and examine the performance of the three types of ESC highlighted in the previous section: perturbation ESC (P-ESC), least squares ESC (LS-ESC) and recursive least squares ESC (RLS-ESC).

The simulation scenario is similar to the one used to determine the power convexity earlier. The evaporator blower speed is initially set at an energy suboptimal 107PWM. From 0 to 3000 seconds, we use PI control to pull down the cabin temperature from 35°C to 21°C. From 3000 seconds onwards, we turn on the ESC and allow it to determine the optimal set of inputs. Note that for both LS-ESC and RLS-ESC, we need to initialize the controller with data prior to activating the controller, as outlined in Chapter 3. In this case study, this is done by slowly increasing the blower speed from 107 to 112 PWM from 3000 to 5000 seconds. At

5000 seconds, the ESC is started by activating the "enable_adj" step function seen in the figures below.

Table 4.1 lists all simulation parameters used for the three different ESC controllers. These values were chosen using the guidelines listed in the previous chapter along with trial and error to achieve the best possible performance for each algorithm. Figures 4.7, 4.8 and 4.9 show the P-ESC, LS-ESC and RLS-ESC respectively implemented in Simulink. These algorithms output a change in evaporator speed “ESC_adj” that is sent to the evaporator blower Simulink block as highlighted Figure 4.10.

Table 4.1 ESC Parameters used in Simulation

ESC METHOD	Parameter	Variable	Value
P-ESC	High Pass Filter Cutoff Frequency	ω_{HP}	0.002
	Low Pass Filter Cutoff Frequency	ω_{LP}	0.001
	Integrator Gain	k	-0.05
	Sinusoidal Dither Amplitude	a	1
	Sinusoidal Dither Frequency	ω	0.002
RLS-ESC	Forgetting Factor	λ	0.997
	Integrator Gain	k	-0.0006
	Initialization Time Range	T_i	2000s
	Initialization Input Range	P	2
	Persistent Excitation Signal Amplitude	a	0.01
	Persistent Excitation Signal Frequency	ω	0.02
LS-ESC	Data Buffer Length	T	1000s
	Integrator Gain	k	-0.00063
	Initialization Time Range	T_i	2000s
	Initialization Input Range	P	2
PI	Proportional Gain	K_p	5
	Integral Gain	K_i	2.5

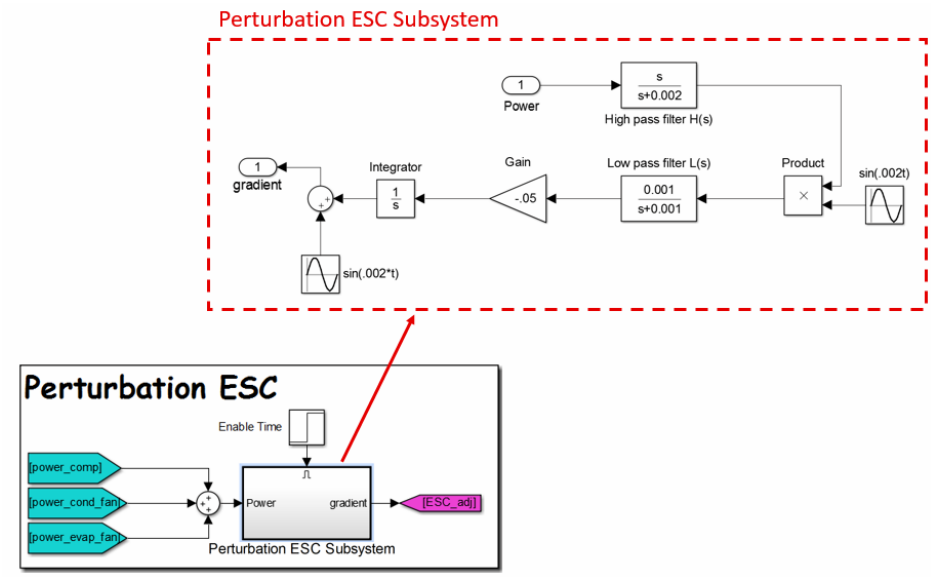

Figure 4.7 P-ESC implemented in Simulink. The block receives the total system power and outputs an adjustment in the blower speed in the direction of a decrease in power.

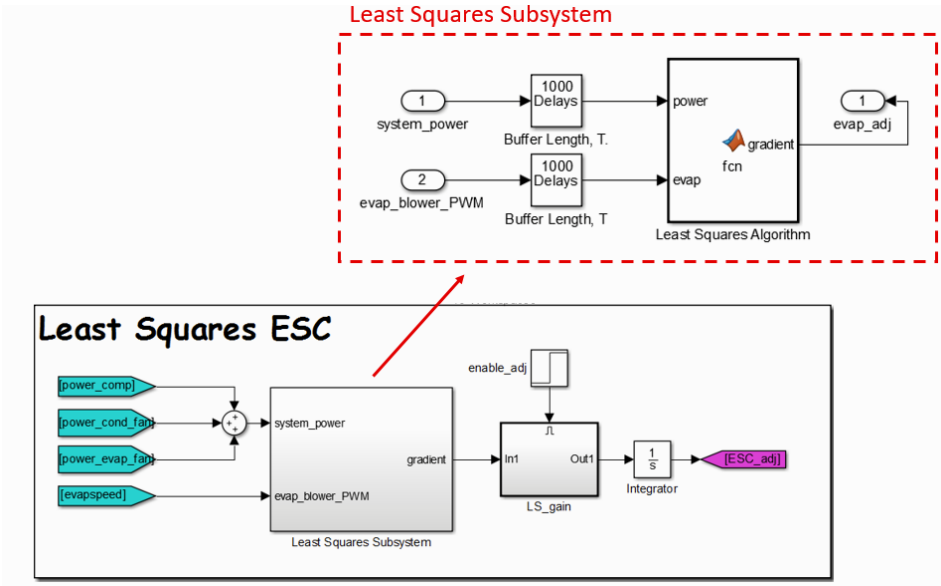

Figure 4.8 LS-ESC implemented in Simulink. The block receives the system power consumption and the blower speed. These two quantities are each stored in a corresponding data buffer which is used to generate a corresponding gradient value using the least squares algorithm. The gradient is then scaled and integrated to generate an evaporator speed adjustment used to minimize the power consumption.

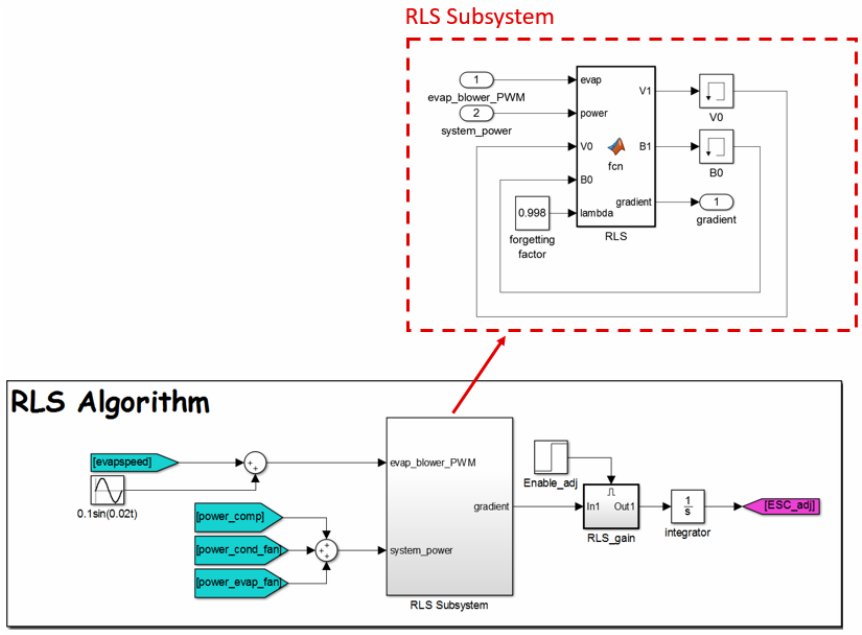

Figure 4.9 The RLS algorithm implemented in Simulink. The algorithm receives the current evaporator blower speed and system power consumption, and uses the RLS algorithm to generate a corresponding gradient estimate that is scaled and integrated to generate an adjustment to the blower speed.

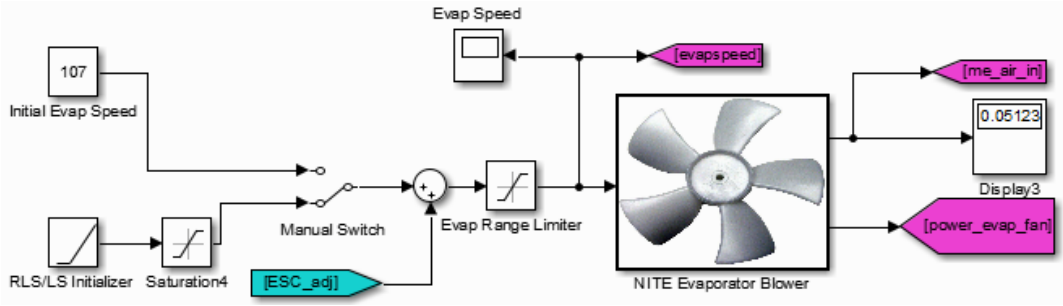

Figure 4.10 The evaporator blower configuration in Simulink. The default speed is set at an energy suboptimal 107PWM. When implementing P-ESC, we use the manual switch to select the top case, which is a constant blower speed. However, when implementing RLS/LS-ESC, we select the bottom case, which ramps the blower speed from 107 to 112 PWM from 3000 to 5000 seconds. The ESC_adj tag is sent from the respective ESC algorithm chosen for the simulation, and adjusts the blower speed correspondingly to minimize the total power consumption.

4.3 Simulation Results

All three ESC algorithms successfully converged to the optimal blower speed (126-128 PWM) which minimizes the power consumption to roughly 523W. As expected, the RLS-ESC/LS-ESC discovered the optimal blower speed fastest; both methods reached the optimal blower speed in roughly 4000 seconds when including the 2000 seconds needed to initialize the controller. On the other hand, P-ESC took 12000 seconds to reach the optimal blower speed. Fig. 4.11 shows the blower speeds over time for each of the three algorithms. Fig. 4.12 shows the corresponding system power consumption over time, along with the power consumption of the suboptimal baseline case (PI control with the blower speed fixed at 107PWM for all time). Fig. 4.13 depicts the compressor speeds for the three ESC algorithms, as well as for the baseline case. Fig. 4.14 shows the cabin temperature over time for the three ESC algorithms.

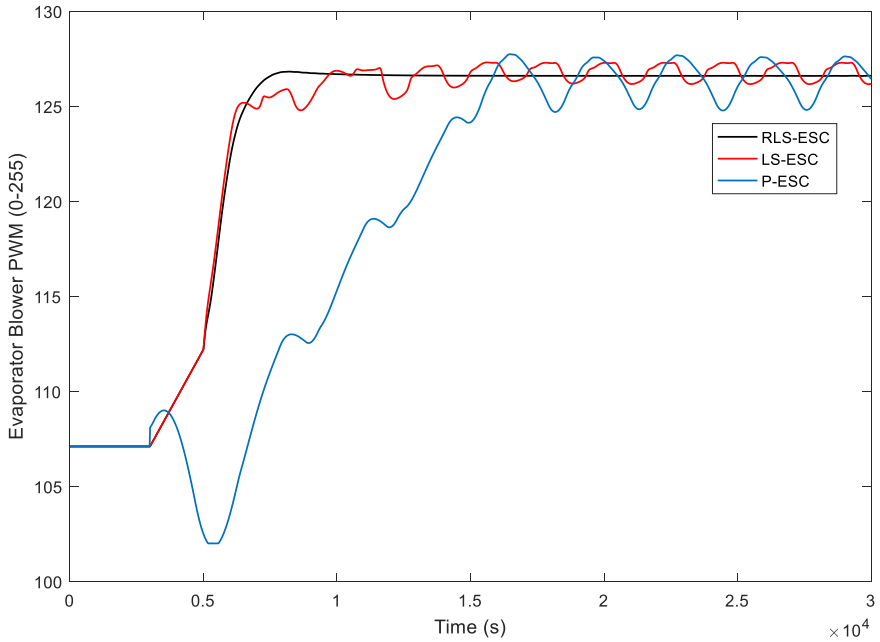

Figure 4.11 Blower speeds over time for the three ESC algorithms. All three algorithms start from a suboptimal 107PWM and converge to the optimal blower speed, with the RLS/LS-ESC algorithms converging faster than P-ESC. The small oscillations in the blower speed generated by LS-ESC is a benign byproduct of the relatively long time buffer length, which is ideal for dynamic systems with relatively long time constants.

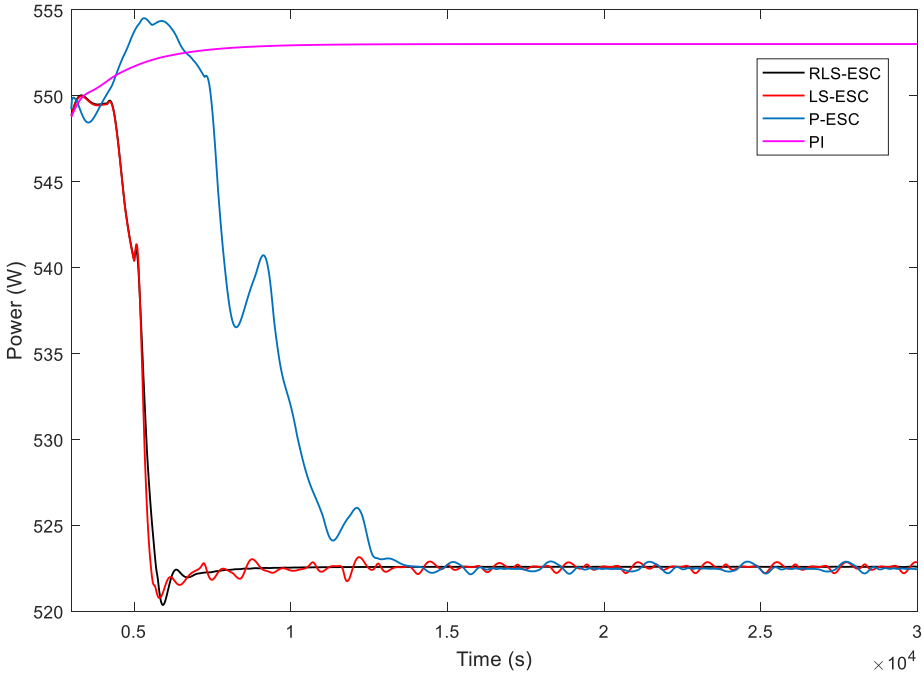

Figure 4.12 The system power consumption over time for the three ESC cases, along with the power consumption of the baseline case. All three ESC algorithms converge to the minimal power consumption of 523W, while the suboptimal case yields a power consumption of 553W.

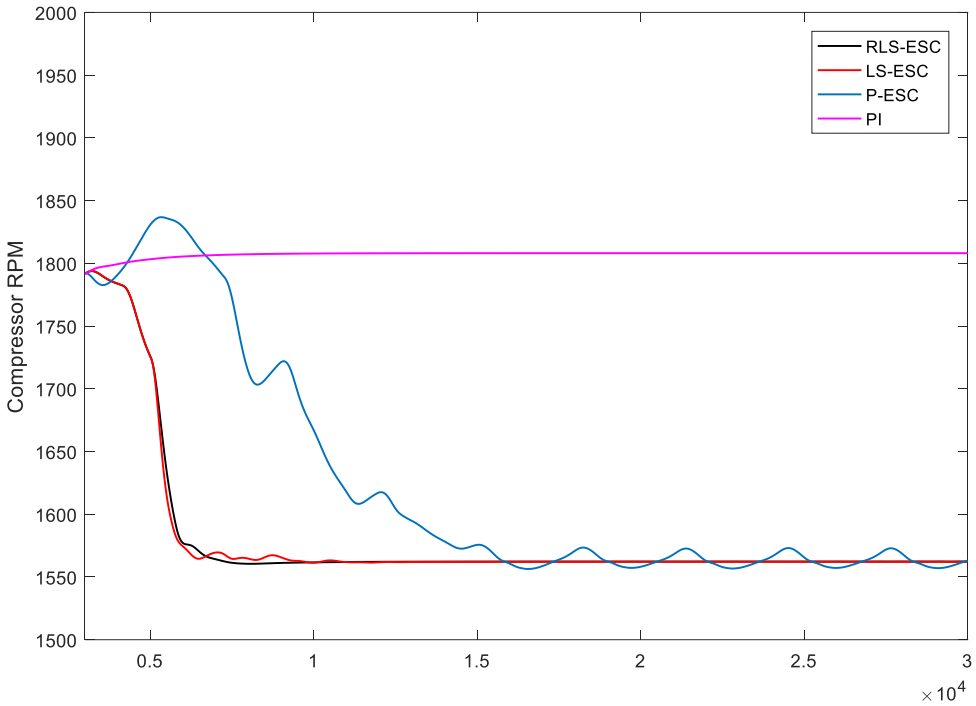

Figure 4.13 The compressor speeds corresponding to the three ESC approaches and the baseline case. All three ESC algorithms converge to the optimal compressor speed.

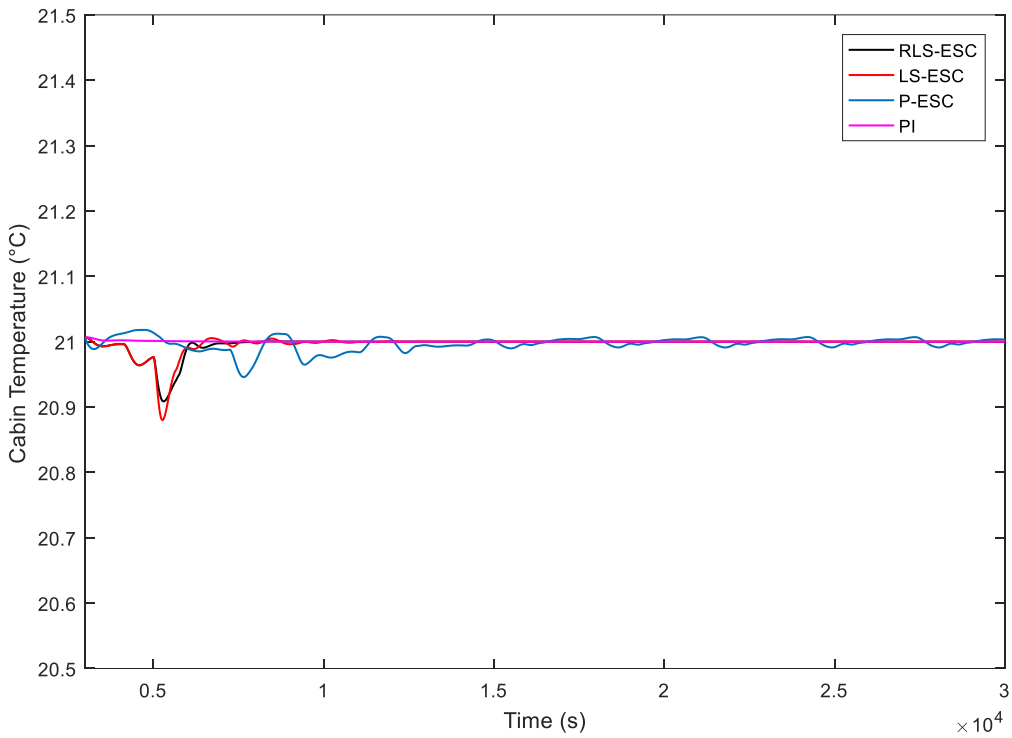

Figure 4.14 The cabin temperature over time once ESC is activated, along with the baseline case. All approaches track the cabin temperature very well with minimal deviations from the temperature setpoint.

The above results demonstrate the successful implementation of ESC. However, the main purpose of ESC implementation in this case is to demonstate battery life extension. Fig. 4.15. shows the battery state of charge over time for the four cases, and Fig. 4.16 summarizes the results by highlighting the battery run time for each of the four cases.

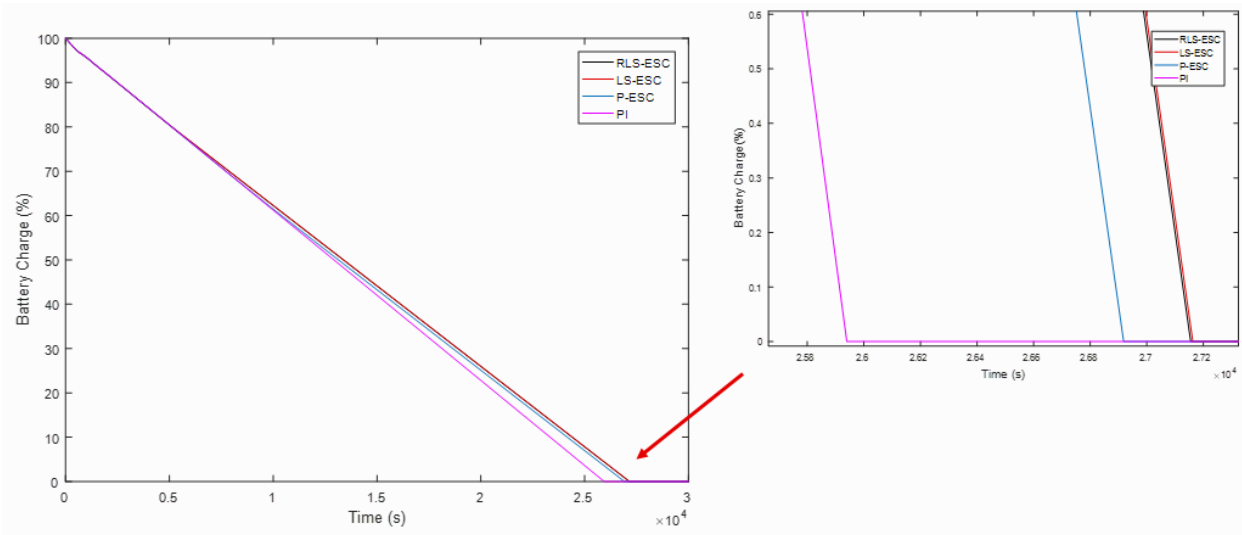

Figure 4.15 The battery state of charge over time for each of the four cases. The time it takes for the battery to drain to 0% charge for the PI, P-ESC, and RLS/LS-ESC cases is 25,940s, 26,919s, and 27,158s resepectively. The P-ESC and RLS/LS-ESC algorithms yield a 3.7% and 4.7% increase in run time respectively.

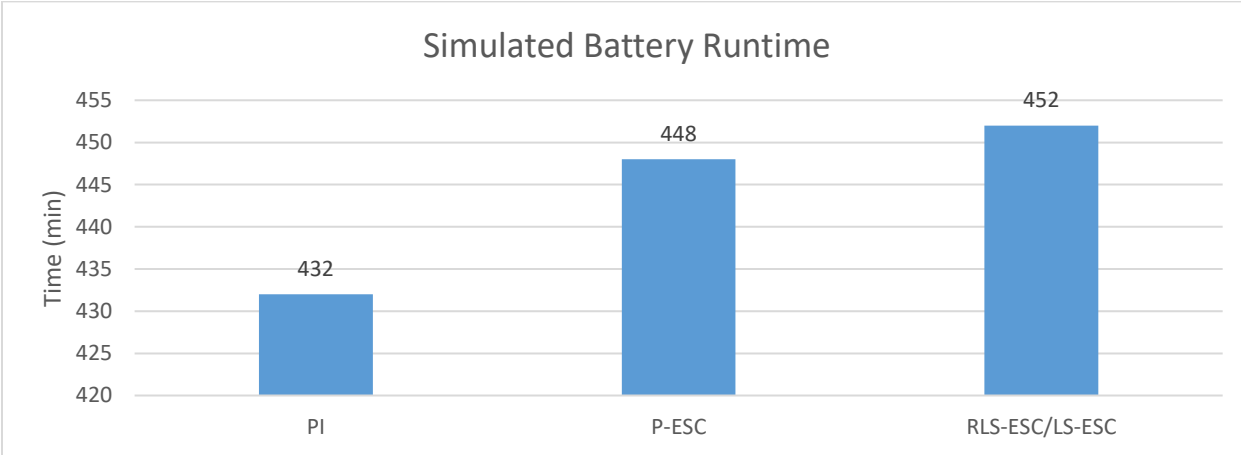

Figure 4.16 A bar plot depicting the runtime of the four cases in minutes. The P-ESC runs for 16 minutes longer than the baseline case, while the RLS/LS-ESC cases run for 20 minutes longer than the baseline case. Again, this is a 3.7% and 4.7% increase in runtime respectively.

4.4 Summary and Next Steps

Thus far, we have demonstrated the use of ESC algorithms to minimize the power consumption of the simulated NITE system while maintaining the vehicle cabin temperature at a constant value. Simulation results show a 4.7% and 3.7% increase in battery life using RLS/LS-ESC and P-ESC respectively. These results validate the use of ESC to optimize VCS and also validate the superior performance of RLS/LS-ESC over P-ESC. While RLS/LS-ESC have virtually the same performance characteristics, RLS-ESC is much less computationally intensive than LS-ESC, and is also generally easier for a user to tune. With these results in hand, we now turn to the experimental system. Will these simulation results mirror ESC performance on the experimental system? How robust are these algorithms to noise and disturbances? What are the challenges of implementing these algorithms in LabVIEW instead of in Simulink? We discuss all of these questions and more in the next section.

Chapter 5

Extremum Seeking Control on the Experimental Integrated NITE System

In the previous chapter, we implemented ESC on the simulated integrated NITE system. Results showed that the use of ESC modestly improved energy efficiency and extended battery life. LS-ESC and RLS-ESC were the most effective at improving run time, with P-ESC also demonstrating some power savings over the baseline case. Naturally, the next step is to investigate whether these results predict ESC performance on the experimental integrated system as well. However, experimental implementation is not as straightforward: there are a number of fundamental differences between the simulated and experimental systems that need to be considered. For instance, the simulated system was developed entirely in MATLAB/Simulink while the experimental system interfaces with LabVIEW. Furthermore, unlike the simulated system, the experimental system is subject to noise and external disturbances which could affect controller performance. And lastly, we need to ensure that industry operators can easily understand and operate the software. Thus, in this chapter, we discuss the implementation of P-ESC, LS-ESC and RLS-ESC on the experimental system using LabVIEW with an emphasis on developing robust and intuitive controllers.

5.1 ESC Development in LabVIEW

ESC algorithm complexity, along with the large number of tunable parameters often poses a significant barrier to adoption. To mitigate these factors, we developed an intuitive front panel user interface so that industry practitioners who do not have prior ESC experience can operate the controller effectively. The front panel can be seen in Fig. 5.1, and a detailed description is given below.

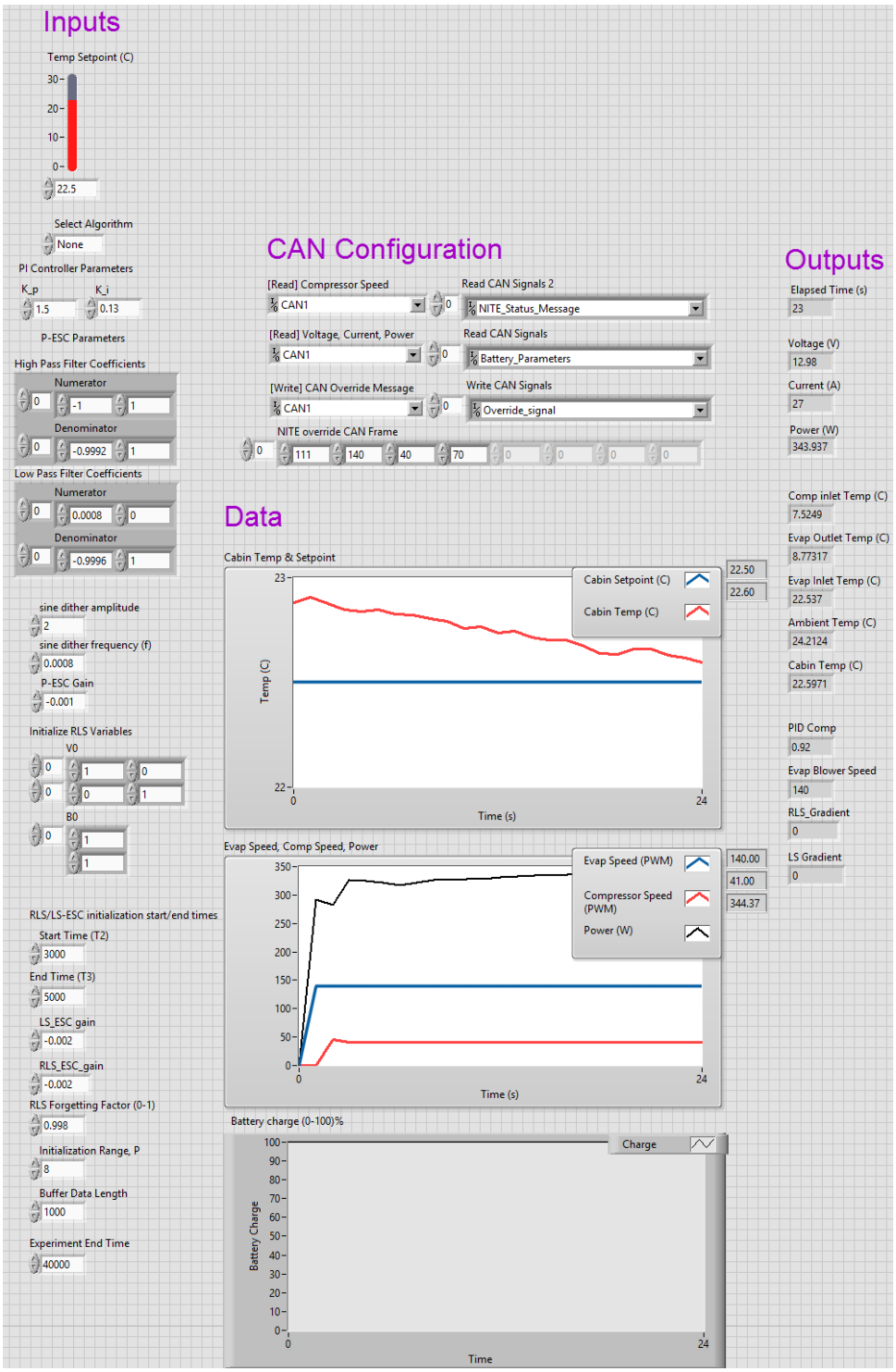

Figure 5.1 The LabVIEW front panel developed for ESC implementation on the physical system.

5.1.1 LabVIEW Front Panel Inputs

The inputs are the values chosen by the operator. Starting from the top, we set the desired cabin temperature set point. Below that, we select the desired control algorithm to use (P-ESC, LS-ESC and RLS-ESC), or lack thereof (None), and also set the PI controller gains used to regulate the cabin temperature. After determining which algorithm to use, we can now input the chosen controller's parameters.

P-ESC: The user determines the filter coefficients along with the dither characteristics and ESC gain. Note that because LabVIEW is normally operated in discrete time, we need to ensure the filter transfer function coefficients are that of a discrete, and not continuous transfer function.

To go from a continuous to discrete function, use the c2d function in MATLAB and specify the time step length along with the conversion method of choice. This thesis utilizes a time step of 1 second along with a zero order hold approximation to obtain the discrete filter coefficients.

LS-ESC: The user chooses the initialization time range length T_i by setting its start and end time, T_2 and T_3 respectively, along with the initialization input range *P*, data buffer length and ESC gain.

RLS-ESC: Like with LS-ESC, the user sets the initialization time and input range. Additionally, the user determines the ESC gain, forgetting factor and also chooses the initial values of V_0 and β_0. Setting V_0 as the identity matrix and β_0 as a vector of ones should be sufficient because, if the initialization time and input range are selected well, then the parameters should rapidly converge to the appropriate values prior to algorithm activation. Finally, we finish by selecting the experiment end time.

5.1.2 LabVIEW Front Panel CAN Configuration

Here, the user selects the module interface through which CAN communication will occur, as well as the specific messages to read or write. As mentioned in chapter 2, we utilize the NI 9862 module to communicate with the NITE. In the figure, CAN1 refers to the module's

interface in the LabVIEW software; select it to communicate with that module specifically. CAN communication with the NITE is established by a document produced by Bergstrom specifying the message identifiers along with their payloads. Here, we read the NITE's compressor speed, power consumption along with voltage and current, and write an override CAN message to set the initial speed of its actuators (evaporator blower, compressor and condenser fan) at the start of the experiment. As seen in the Fig. 5.1, there are a series of numbers in the override CAN frame. According to the NITE CAN communication document, the first number, 111 tells the NITE that this message will override its default component speeds. The numbers 140, 40 and 70 are the user determined PWM speeds of the blower, compressor and condenser fan respectively.

The LabVIEW block diagram takes in all of these signals and manipulates them accordingly. In particular, the PI controller manipulates the compressor speed PWM and the ESC manipulates the evaporator blower speed as was implemented in the previous chapter. More detail on CAN communication and the specific LabVIEW code used can be found in Appendix B.

5.1.3 LabVIEW Front Panel Outputs and Data

As seen in the above figure, the front panel displays a number of pertinent outputs to the user. Starting from the top, we can read the elapsed experiment time, the system voltage, current and power draw, the temperature at a number of different locations as well as the PID adjustment signal, current evaporator blower speed and gradient estimates generated by RLS/LS-ESC. Furthermore, output data is also represented graphically as seen in the above figure, as well as in Fig. 5.2 below. Note that the battery chart seen above represents a "fictitious battery" since the NITE unit is hooked up to a wall power supply. This fictitious battery is based off of the energy capacity of the four Trojan AGM battery bank used by industry practitioners to power the NITE. We calculate the total energy capacity of the batteries assuming each battery supplies 80 amp hours of current at 12V, for a total energy capacity of 13,824,000 Joules. We approximate a power draw by continually subtracting the system power consumption every second from this value until it reaches zero.

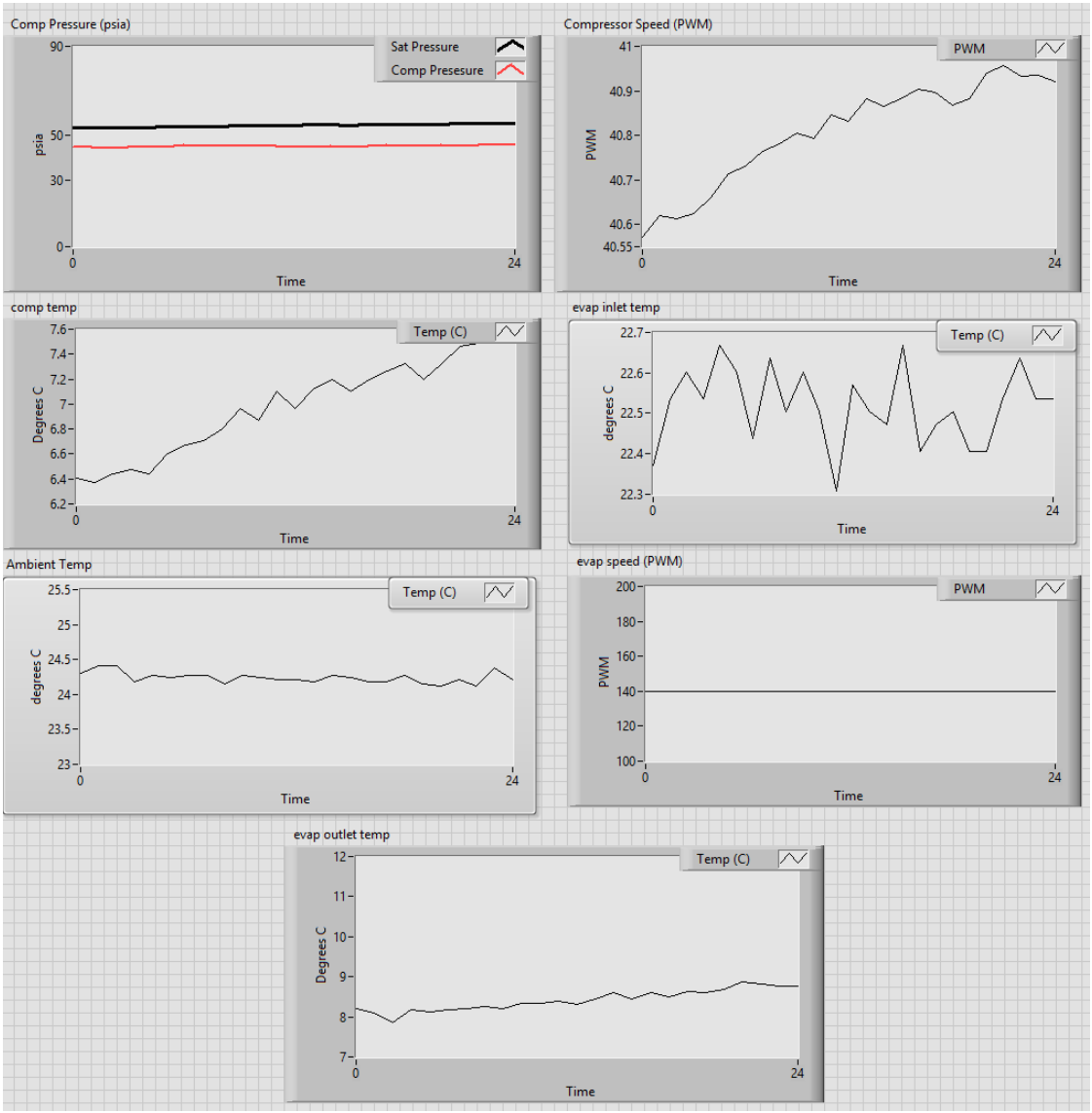

Figure 5.2 Additional graphs showing different temperature, pressure and component states of the integrated NITE system.

5.2 Determining NITE Power Convexity

Similar to the procedure outlined in section 4.1 for the simulated NITE, we wish to determine whether the NITE's power consumption is convex with respect to its inputs for a constant cabin temperature. We choose a similar scenario as before: we wish to minimize power while maintaining a cabin temperature of 22.5°C with a cabin heat load of 500W. From *T = 0 to 3000s*, we only use the PI controller to modulate the compressor speed to converge to the desired temperature and we keep the evaporator fan speed fixed at a suboptimal value of 140 PWM. Once the cabin temperature reaches steady state, at *T=3000s* we start to slowly ramp the blower speed up at a rate of roughly 0.002 PWM/sec until *T=27000s*, as seen in Fig. 5.3. As the blower speed increases, the compressor speed falls precipitously as also depicted in Fig. 5.3. Like in simulation, the power consumption was found to be convex with respect to the inputs. However, as Fig. 5.4. shows, the performance function here is much "steeper"; that is, the power consumption decreases by almost 200W going from the least optimal blower speed (530W at 140-145 PWM) to the most optimal blower speed (350-360W at 170-180 PWM). The simulated system, on the other hand, only showed a drop of roughly 30W between its most optimal and suboptimal input combination. As mentioned in chapter 2, the discrepancy between the simulated and experimental system is due to a lack of full model parameterization, which prevented cross-validation. It is likely that the compressor by and large contributes most to this discrepancy because it consumes the most power and was also not parameterized aside from its volume owing to a lack of data on all of its states. Consequently, battery life extension may be much more significant for the physical unit since there is a lot more room for power savings. However, because VCS performance tends to vary from run to run, some variation in the shape and characteristics of the performance function can be expected. Fig. 5.5 shows the cabin temperature tracking the temperature set point well over the course of the experiment.

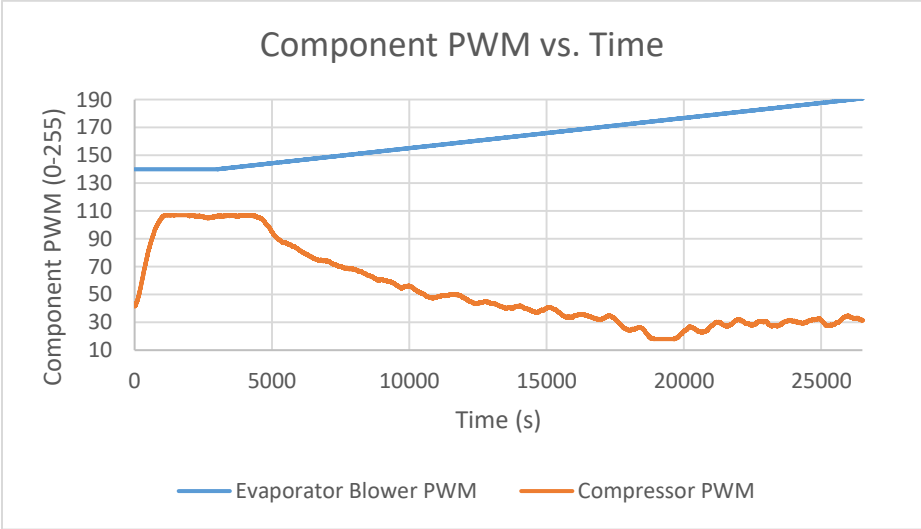

Figure 5.3 Component speeds over time. The compressor speed converges to roughly 25-30 PWM.

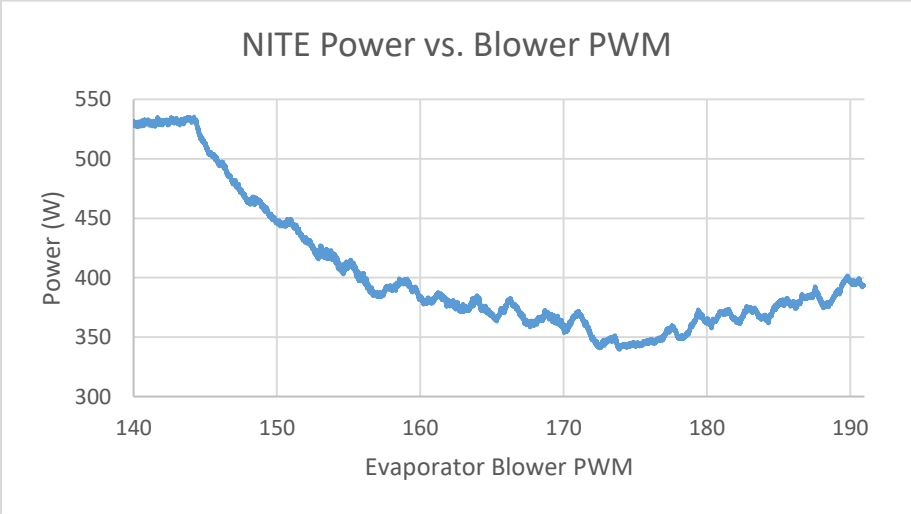

Figure 5.4 The NITE power consumption as a function of the blower speed. The relationship is convex, enabling real time optimization of this system.

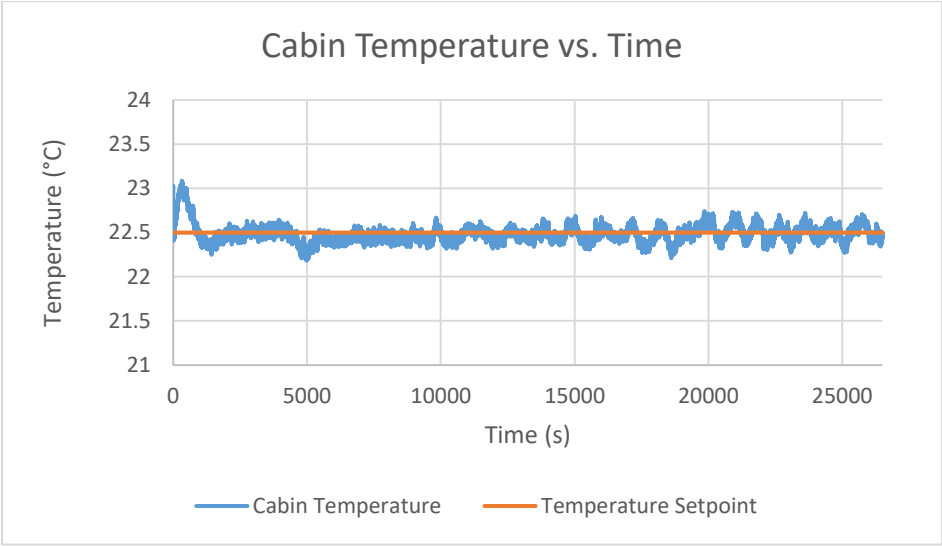

Figure 5.5 Cabin temperature over time. The temperature set point was tracked well through the course of the experiment.

5.3 Implementing ESC on the Experimental System

With power convexity verified, we implement and analyze the performance of P-ESC, LS-ESC and RLS-ESC. We use the same heat load and cabin temperature set-point as before. The evaporator blower speed is again initially set at an energy suboptimal 140 PWM. For P-ESC, we use PI control to regulate the cabin temperature for the first 3000 seconds, and activate the ESC algorithm from 3000 seconds onwards and allow it to determine the optimal combination of inputs. For LS-ESC/RLS-ESC, we follow the same procedure, except at 3000 seconds, we instead perform the initialization procedure for T_i seconds and then activate the ESC.

Table 5.1 lists all simulation parameters used for the three different ESC controllers. These values were chosen based on guidelines listed in the previous chapter along with trial and error to achieve the best possible performance. Note that the parameter values here are somewhat different than in simulation due to different system characteristics and additional factors affecting performance. For example, the PI gains used here are lower than those used in simulation to improve algorithm robustness to noise: high gain values increase controller

reactivity to disturbances in cabin temperature which could then excite system dynamics and interfere with ESC optimization. Note that disturbances and sensor noise induce a persistent excitation in signals, removing the need for an external persistent excitation signal for RLS-ESC.

Table 5.1 ESC Parameters used in Experimental Implementation

ESC METHOD	Parameter	Variable	Value
P-ESC	High Pass Filter Cutoff Frequency	ω_{HP}	0.0008
	Low Pass Filter Cutoff Frequency	ω_{LP}	0.0008
	Integrator Gain	k	-0.002
	Sinusoidal Dither Amplitude	a	2
	Sinusoidal Dither Frequency	ω	0.0008
RLS-ESC	Forgetting Factor	λ	0.9978
	Integrator Gain	k	-0.002
	Initialization Time Range	T_i	2000s
	Initialization Input Range	P	8
	Persistent Excitation Signal Amplitude	a	N/A
	Persistent Excitation Signal Frequency	ω	N/A
LS-ESC	Data Buffer Length	T	1000s
	Integrator Gain	k	-0.0018
	Initialization Time Range	T_i	2000s
	Initialization Input Range	P	8
PI	Proportional Gain	K_p	1.5
	Integral Gain	K_i	0.12

The LabVIEW code used to implement these algorithms can be found in Appendix B. To account for variation in environmental conditions, we ran each ESC algorithm twice.

5.4 Experimental Results

Battery runtime results for the baseline and three ESC cases are shown in Table 5.2. All three ESC algorithms significantly reduced system power consumption through convergence to the optimal range of evaporator blower and compressor speeds. However, like in simulation, RLS-ESC/LS-ESC discovered the optimal blower speed fastest, resulting in larger increases in battery life (29.6%-34.6%) over P-ESC (22.7%-24.8%). RLS-ESC/LS-ESC reached the optimal blower speed in approximately 3000-4000 seconds when including the 2000 seconds needed to initialize the controller. On the other hand, P-ESC took 10000-12000 seconds to reach the optimal blower speed. In terms of minutes of additional runtime, RLS-ESC/LS-ESC added roughly 138-152 minutes of run time, while P-ESC added 103-109 minutes of runtime. Variations in results between runs for each controller were notable but were not very significant. Fig. 5.6 depicts the average battery life increase from the three algorithms graphically, and Fig. 5.7 illustrates the battery runtime for each algorithm for each run.

Table 5.2 Battery runtime and percent increases over baseline for each algorithm.

	P-ESC		**LS-ESC**		**RLS-ESC**	
	Run 1	**Run 2**	**Run 1**	**Run 2**	**Run 1**	**Run 2**
Baseline Runtime (min)	454	439	437	466	444	449
ESC Runtime (min)	557	548	589	604	589	601
Battery run time increase (%)	22.7	24.8	34.6	29.6	32.7	33.8

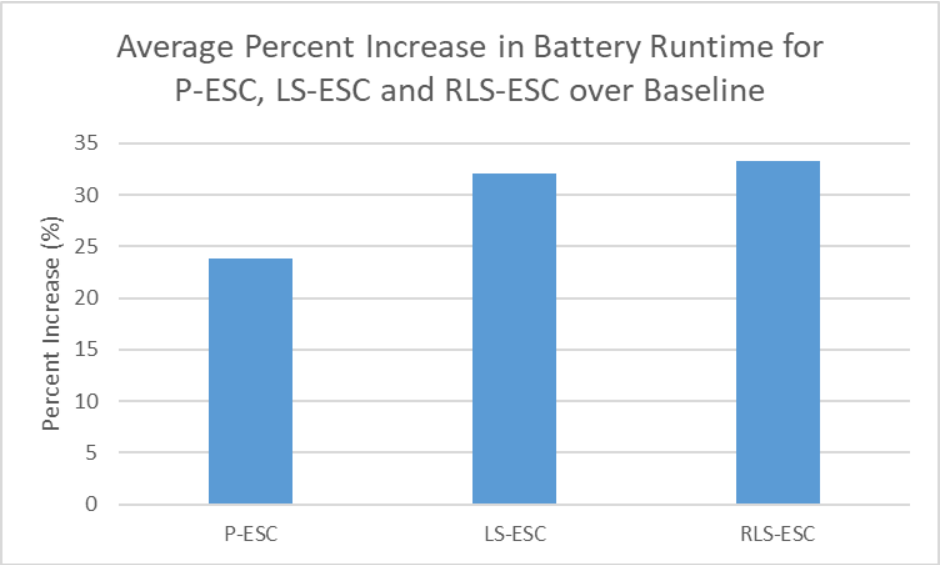

Figure 5.6 Average percent increase in the battery runtime over each of the respective baseline cases.

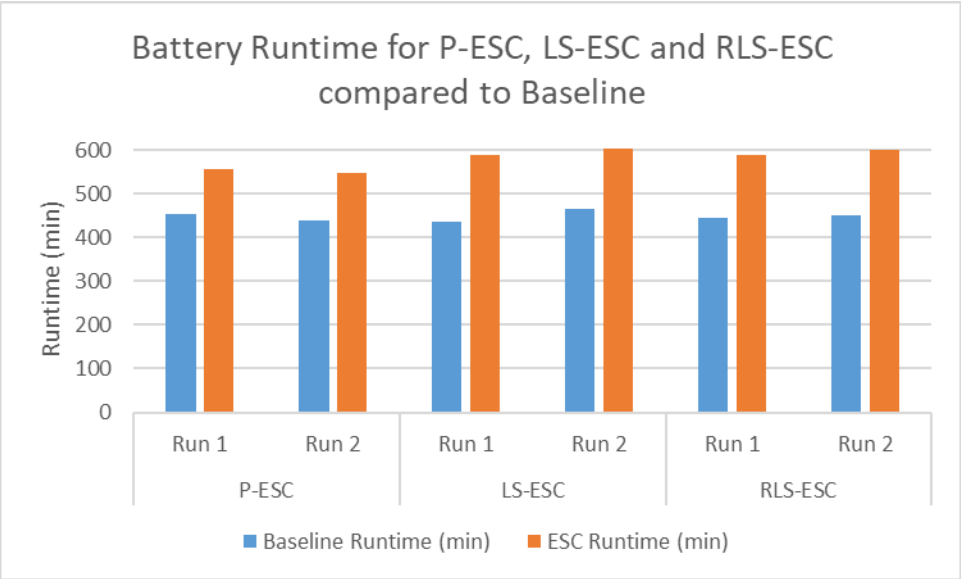

Figure 5.7 Battery runtime for each of the two runs performed for each algorithm, along with the respective baseline runtimes.

5.4.1 Analysis of each Run

Figures 5.8 – 5.12 detail the battery charge, power draw, blower and compressor PWM, cabin temperature and ambient temperature over the course of P-ESC run 1. As seen in the figures below, prior to ESC activation, the initial blower speed is fixed at a suboptimal 140 PWM. PI control raises the compressor PWM to 95 to track the cabin temperature setpoint, resulting in a baseline power draw of 507W. After ESC activation, the evaporator blower speed increases to roughly 160-165 PWM in 10000 seconds, resulting in a corresponding drop in compressor speed to 40-55 PWM and a new power consumption between 370-410W. Note that these component speeds converge slightly outside the optimal range of 170-180 PWM. One possible reason for this is that the slow gradient estimation intrinsic to P-ESC may make it more likely for the controller to "get stuck" once getting closer to the optimal region, where the gradient is "flatter". The cabin temperature set point is tracked very well, with small fluctuations no greater than $\pm 0.25°C$. The ambient temperature in the room fluctuated between $24°C - 24.5°C$.

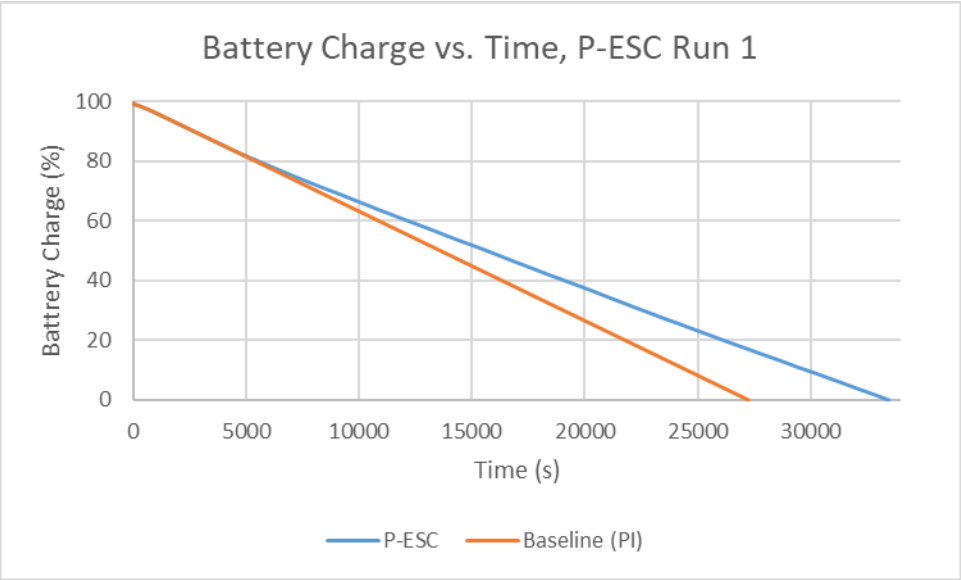

Figure 5.8 Battery charge vs. time for the first P-ESC run and its baseline case.

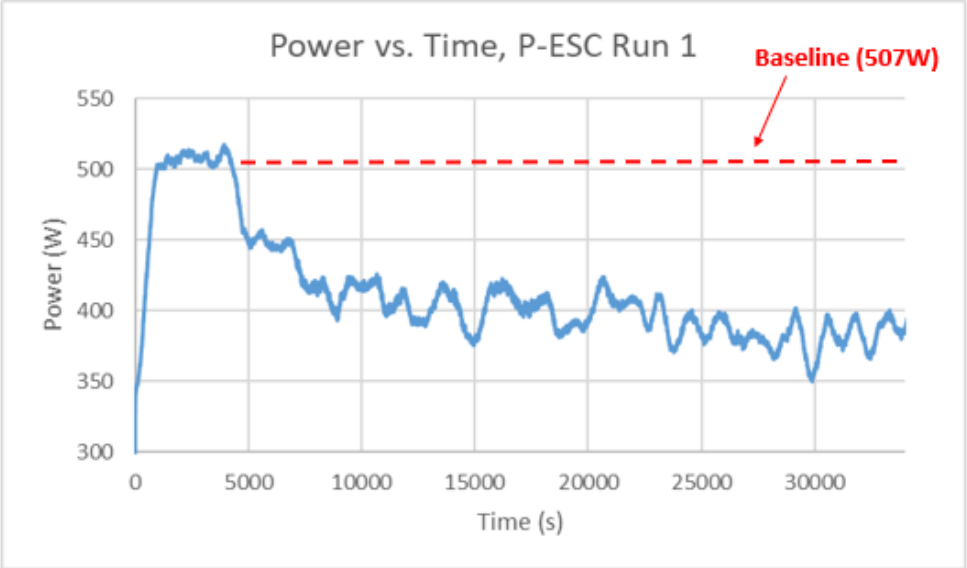

Figure 5.9 Power vs. time for the first P-ESC run.

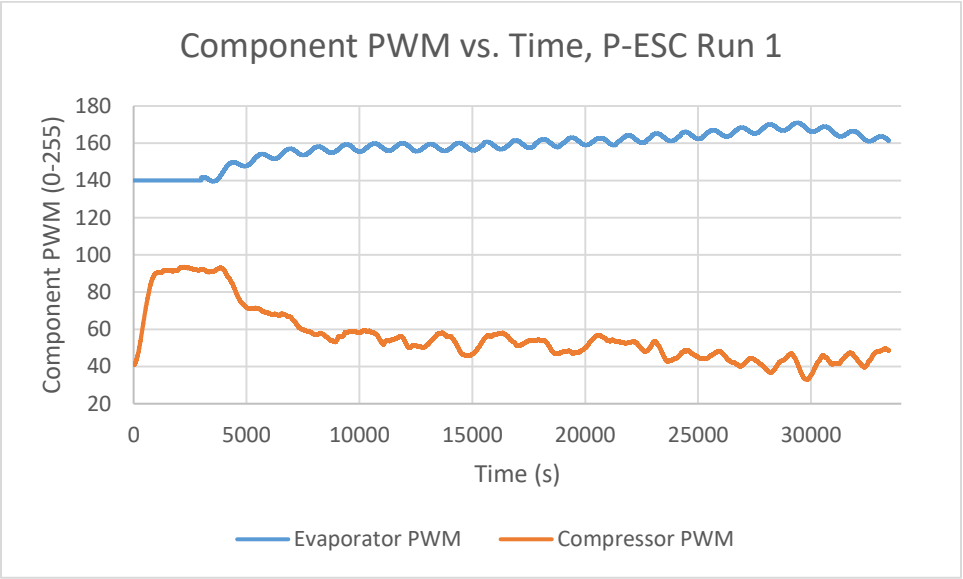

Figure 5.10 Component PWM vs. time for the first P-ESC run.

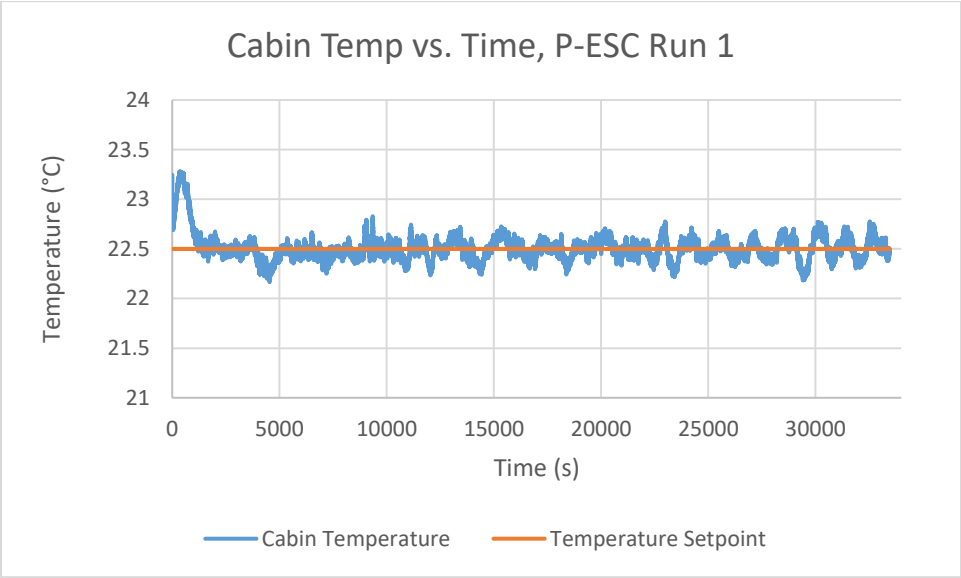

Figure 5.11 Cabin temperature vs. time for the first P-ESC run.

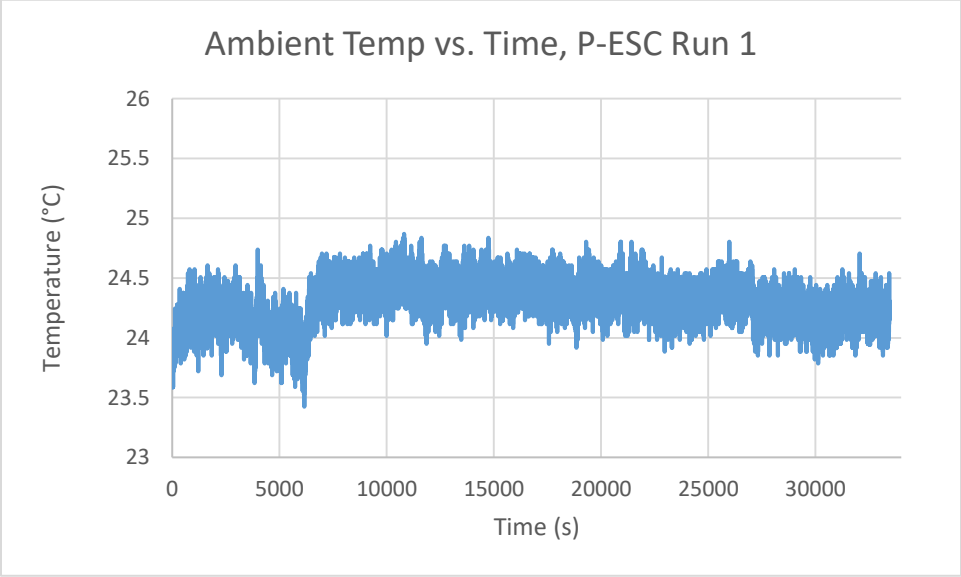

Figure 5.12 Ambient temperature vs. time for the first P-ESC run.

Figures 5.13 – 5.17 detail the battery charge, power draw, blower and compressor PWM, cabin temperature and ambient temperature over the course of the second P-ESC run. Like before, prior to ESC activation, the initial blower speed is fixed at a suboptimal 140 PWM. PI control raises the compressor PWM to 100 to track the cabin temperature set point, resulting

in a baseline power draw of 525W. After the ESC is activated, the evaporator blower speed increases to roughly 160-165 PWM in 10000 seconds, and the compressor speed decreases to 40-60 PWM resulting in a new power consumption between 380-420W. Again, the controller showed some difficulty reaching the most optimal region of the performance function, and even moved outside of the optimal region towards the end. This may be due to the reasons listed previously, along with errors estimating the gradient due to disturbances. Nevertheless, the cabin temperature set point is tracked very well, with small fluctuations no greater than $\pm 0.25°C$. The ambient temperature in the room fluctuated between $24°C - 24.5°C$.

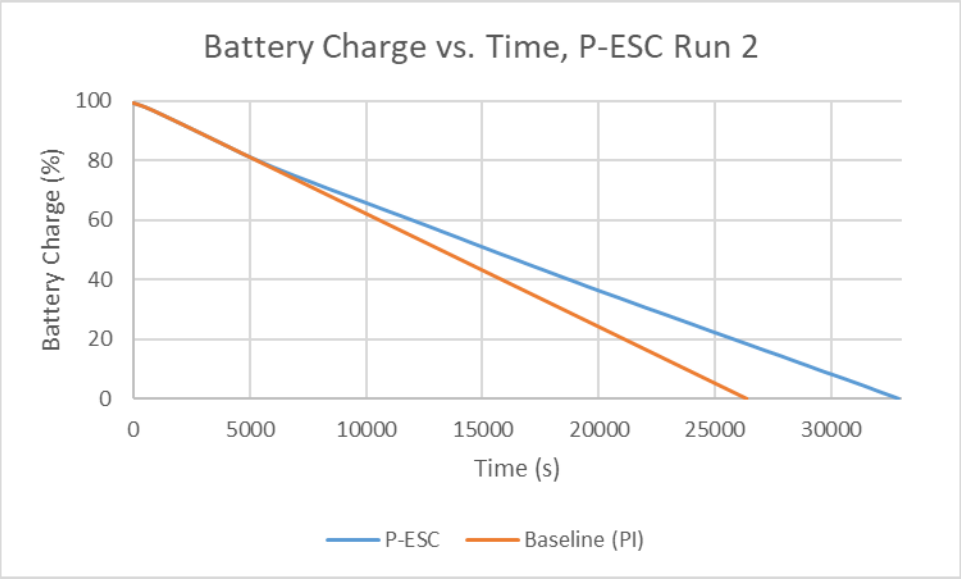

Figure 5.13 Battery charge vs. time for the second P-ESC run and its baseline case.

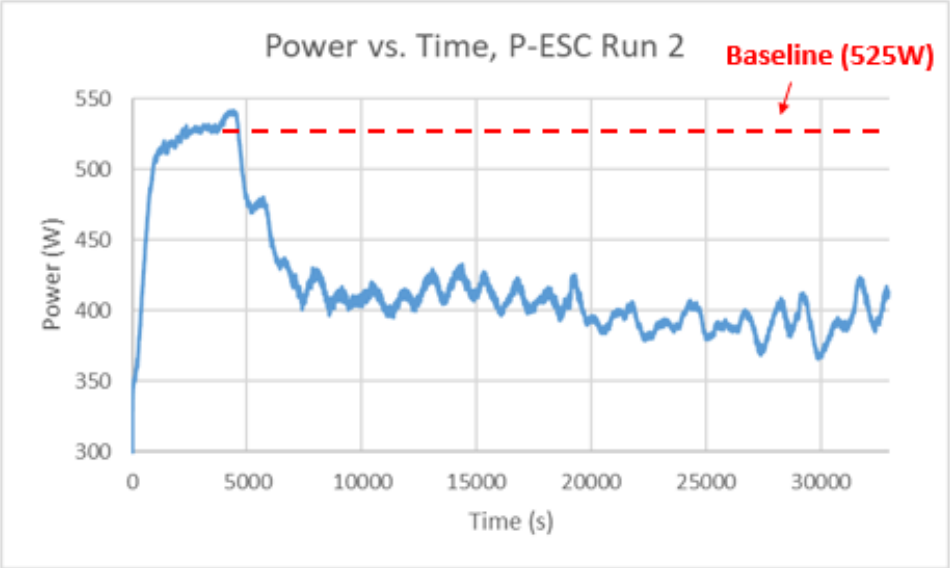

Figure 5.14 Power vs. time for the second P-ESC run.

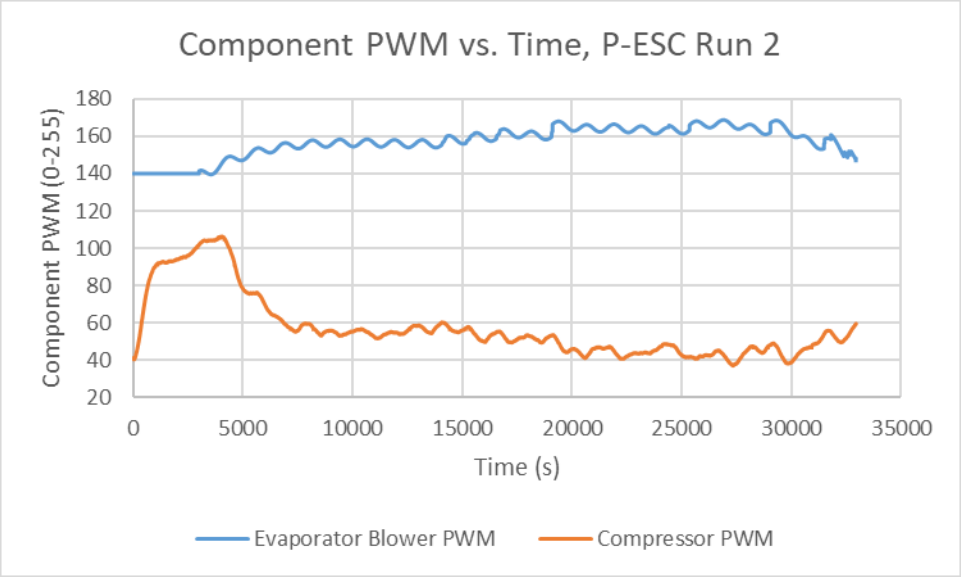

Figure 5.15 Component PWM vs. time for the second P-ESC run.

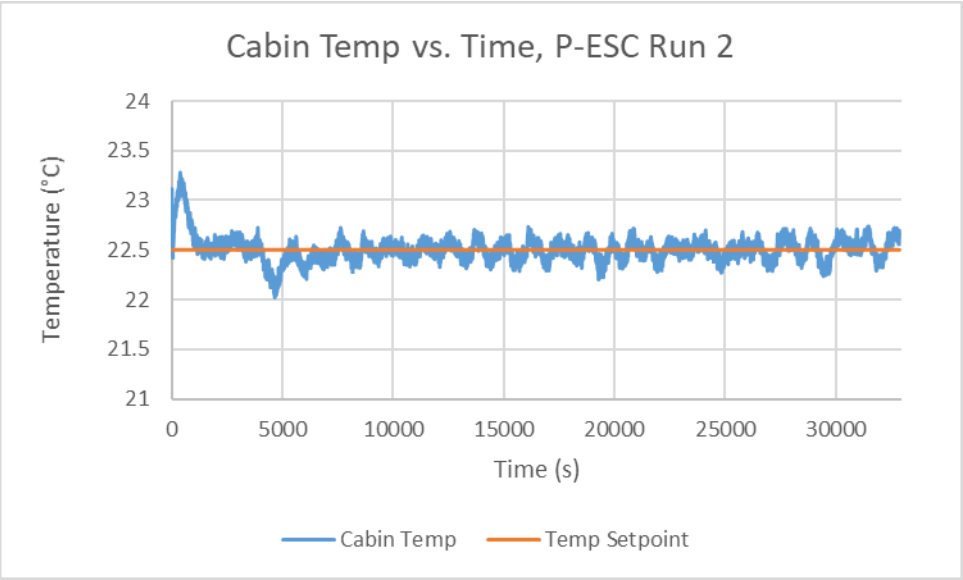

Figure 5.16 Cabin temperature vs. time for the second P-ESC run.

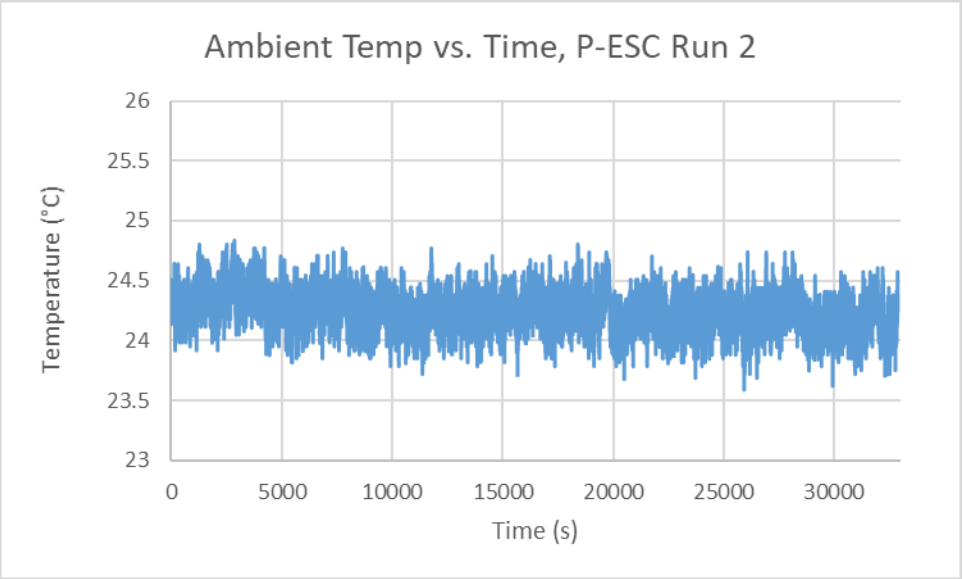

Figure 5.17 Ambient temperature vs. time for the second P-ESC run.

Figures 5.18 – 5.22 show the battery charge, power draw, blower and compressor PWM, cabin temperature and ambient temperature over the course of the first LS-ESC run. As before, the initial blower speed is fixed at a suboptimal 140 PWM for 3000 seconds. PI control raises the compressor PWM to 107 to track the cabin temperature set point, resulting in a baseline power draw of 527W. From 3000 to 5000 seconds, the blower speed is slowly increased to initialize the controller with performance data, resulting in a decrease in compressor speed and power. After the ESC is activated at 5000 seconds, the evaporator blower speed increases to an energy optimal 165-180 PWM in 3000 seconds, resulting in a drop in compressor speed to 25-35 PWM and a new power consumption between 350-370W. Due to faster gradient estimation, the LS-ESC was able to converge to the true function minimum more rapidly than P-ESC. One interesting feature, however, was the increase in fluctuations in the compressor speed, blower speed and the cabin temperature especially, which increased to $\pm 0.5°C$. This may be a result of the large data buffer length used, which may result in the controller having more "inertia", causing more oscillatory behavior around the performance function minimum. Decreasing the data buffer length increases susceptibility to noise and disturbances however, so this is a tradeoff to be balanced.

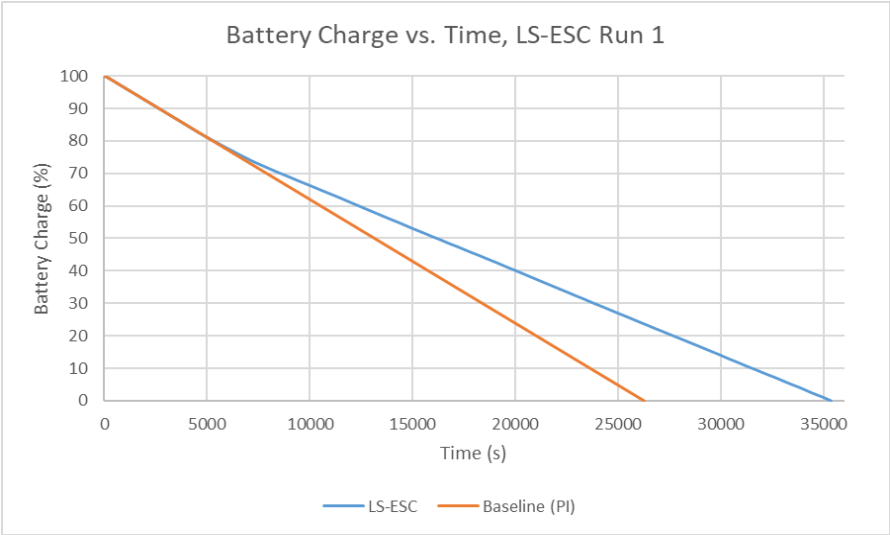

Figure 5.18 Battery charge vs. time for the first LS-ESC run and its baseline case.

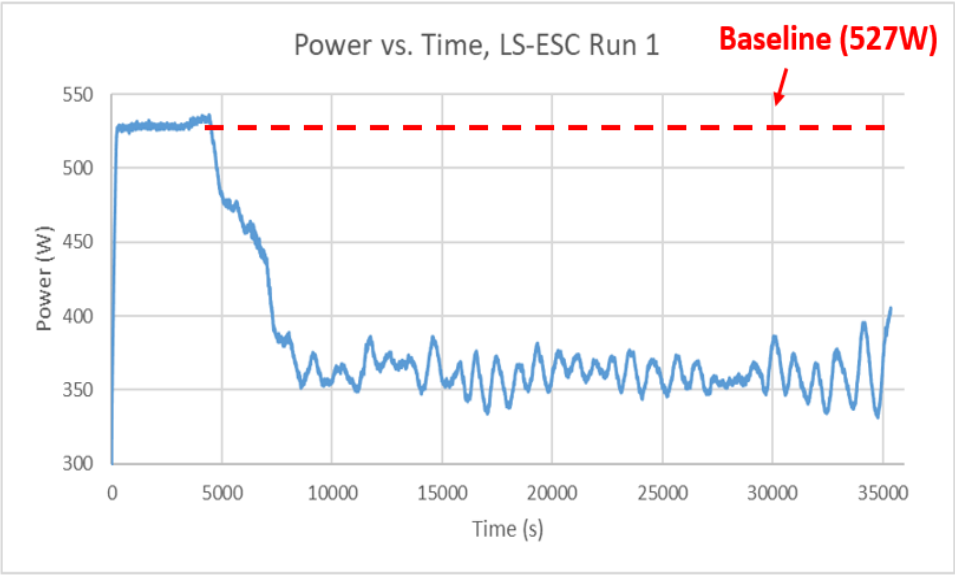

Figure 5.19 Power vs. time for the first LS-ESC run.

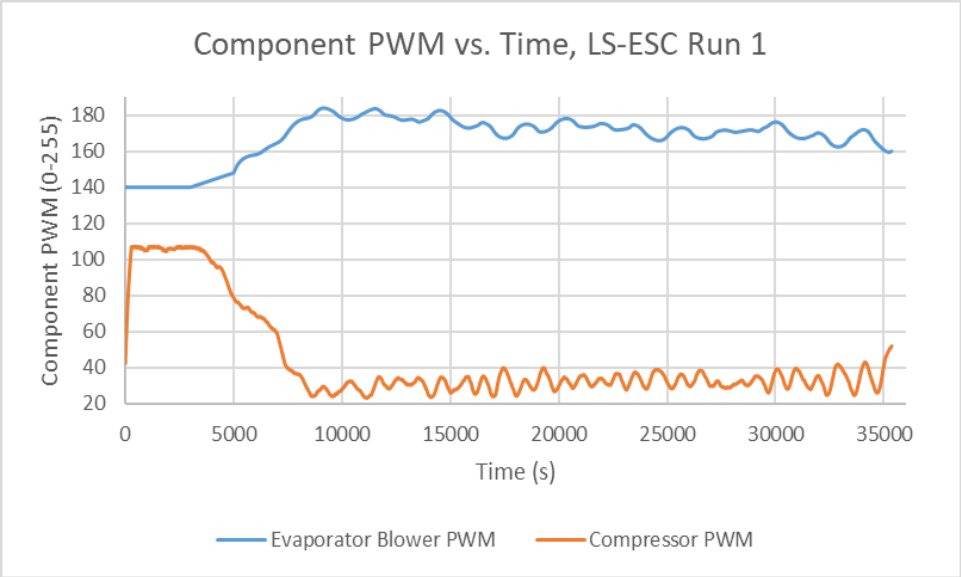

Figure 5.20 Component PWM vs. time for the first LS-ESC run.

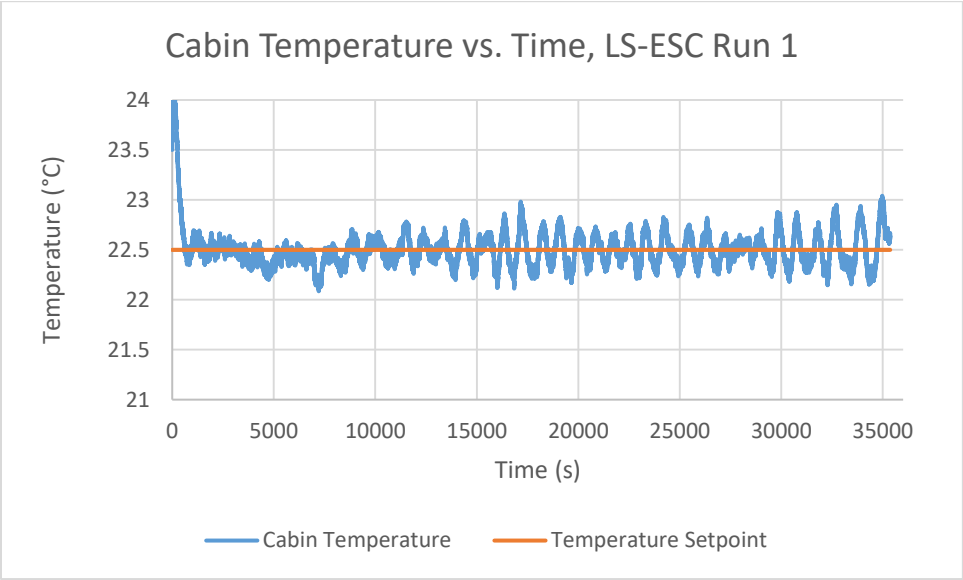

Figure 5.21 Cabin temperature vs. time for the first LS-ESC run.

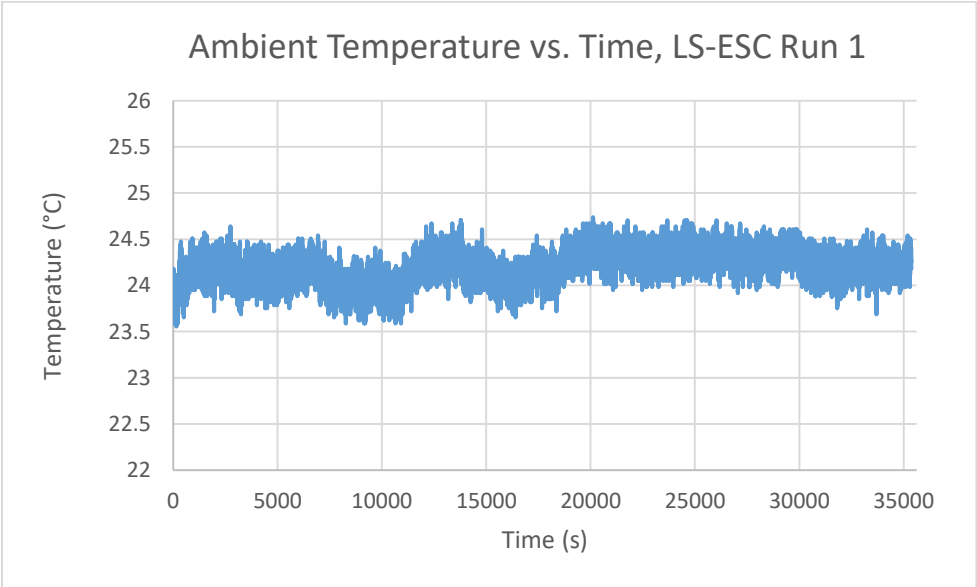

Figure 5.22 Ambient temperature vs. time for the first LS-ESC run.

Figures 5.23 – 5.27 depict the battery charge, power draw, blower and compressor PWM, cabin temperature and ambient temperature for LS-ESC run 2. This time, the baseline power consumption was 497W with a compressor speed between 80-90 PWM. Like before, after the ESC is activated, the evaporator blower speed increases to 160-175 PWM in 3000 seconds, resulting in a drop in compressor speed to 25-40 PWM and a new power consumption between 350-370W. Interestingly, there were less oscillations this run. This may indicate that these oscillations are triggered by external disturbances or other transient factors. Furthermore, there were some disturbances in the ambient temperature initially, flattening out after a few thousand seconds.

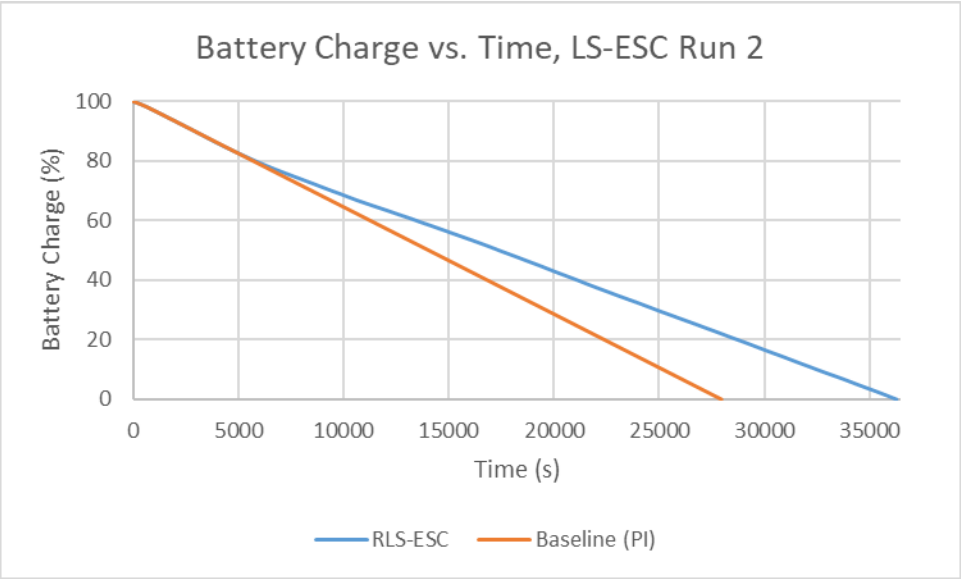

Figure 5.23 Battery charge vs. time for the second LS-ESC run and its baseline case.

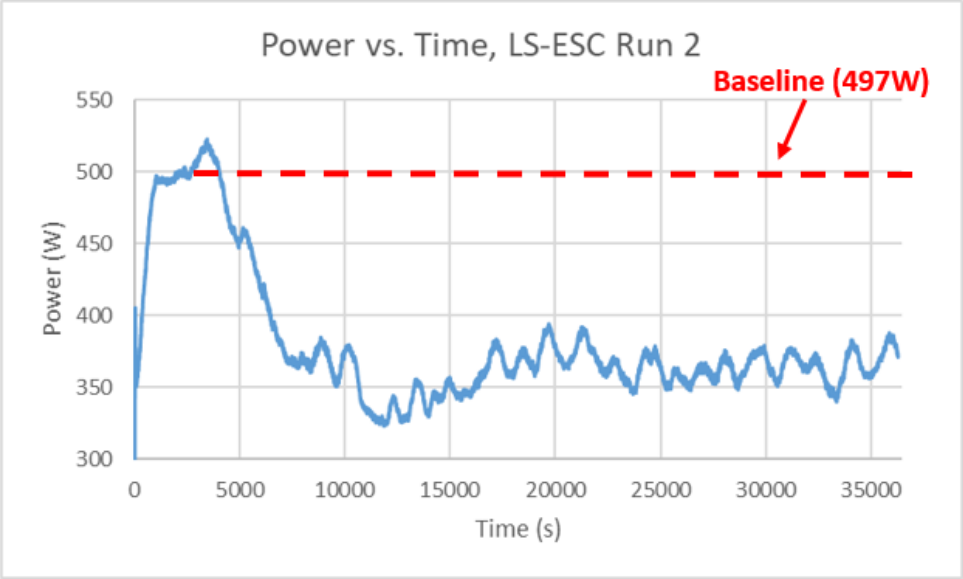

Figure 5.24 Power vs. time for the second LS-ESC run.

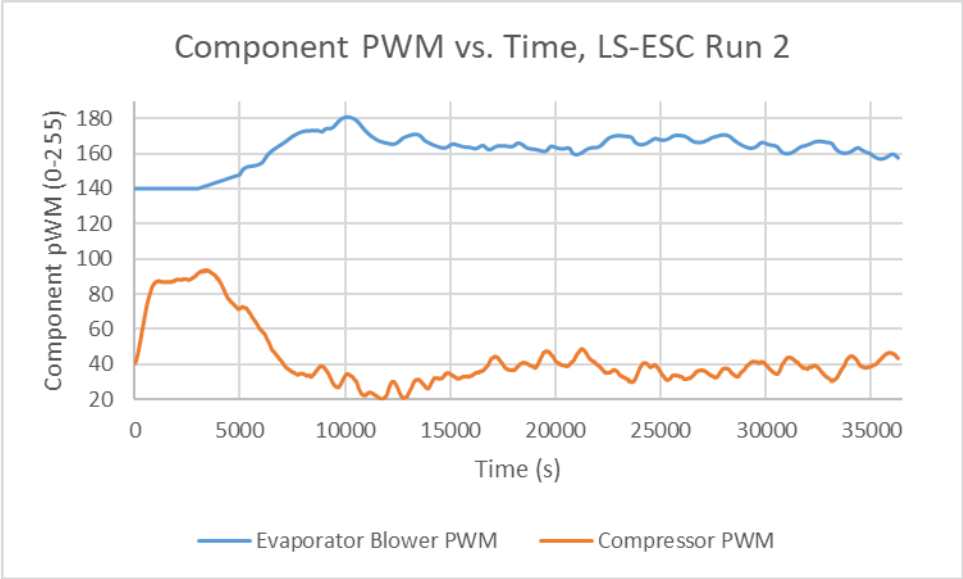

Figure 5.25 Component PWM vs. time for the second LS-ESC run.

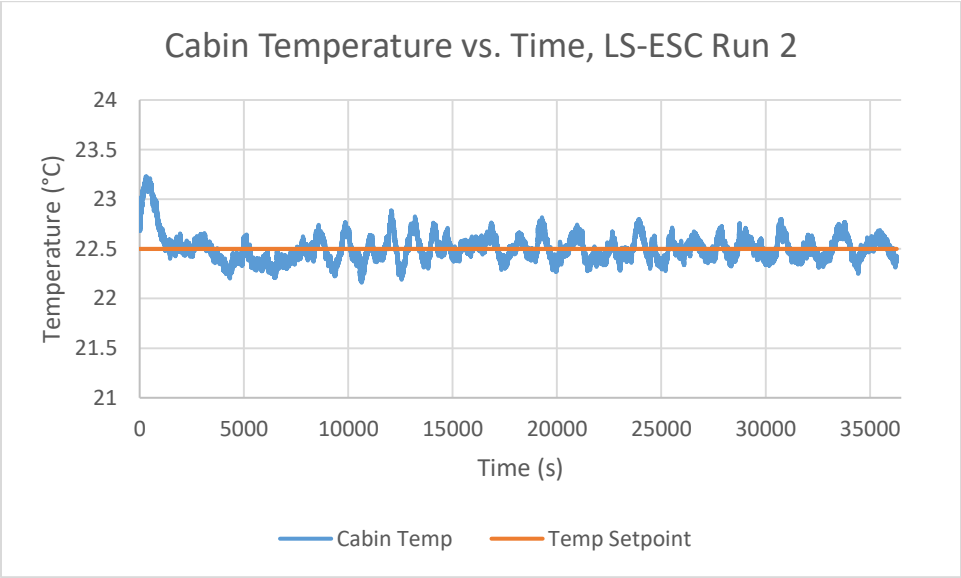

Figure 5.26 Cabin temperature vs. time for the second LS-ESC run.

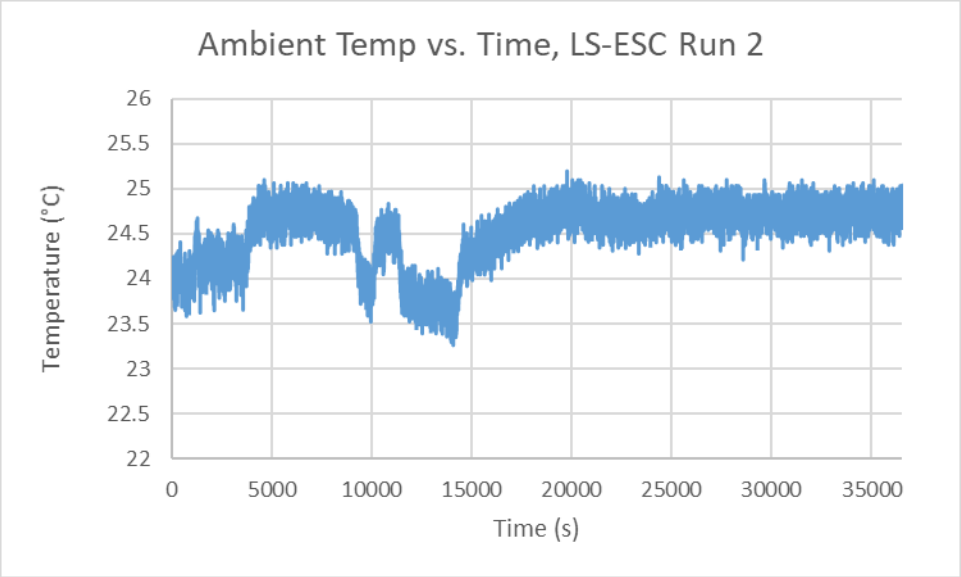

Figure 5.27 Ambient temperature vs. time for the second LS-ESC run.

Figures 5.28 – 5.32 show the battery charge, power draw, blower and compressor PWM, cabin temperature and ambient temperature for RLS-ESC run 1. The baseline power consumption was 525W with a compressor speed around 105 PWM. Like for LS-ESC, we ramp the evaporator blower speed from 3000 to 5000 seconds and activate the ESC algorithm after. The evaporator blower speed quickly increases to roughly 180-190 PWM before settling around 175 PWM. This slight overshoot is a result of a large forgetting factor which improves sensitivity to noise at the expense of a slightly less accurate gradient estimate. The power consumption drops to 380W in roughly 3000 seconds and eventually settles around 350W, while the compressor speed drops to 35 PWM. Note that the plots generally look "smoother" and less oscillatory than LS-ESC. This is because RLS-ESC applies an exponentially decaying filter to all past performance data as opposed to LS-ESC which uses a fixed moving window of T seconds of data. The only exceptions are some oscillations evident in the cabin temperature and component speeds around 10500 and 27000 seconds. This may be the result of some disturbances. Other than that, the cabin temperature and ambient temperature behave well.

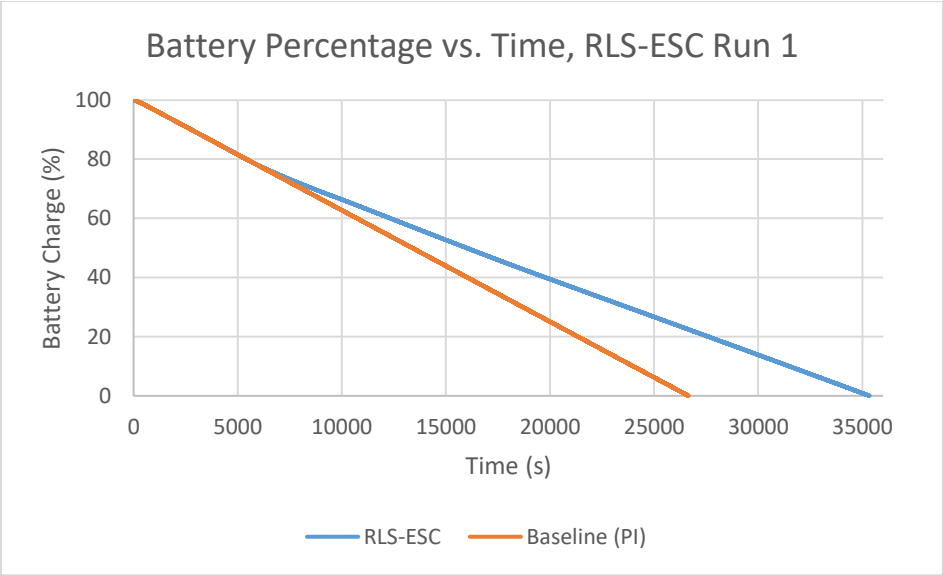

Figure 5.28 Battery charge vs. time for the first RLS-ESC run and its baseline case.

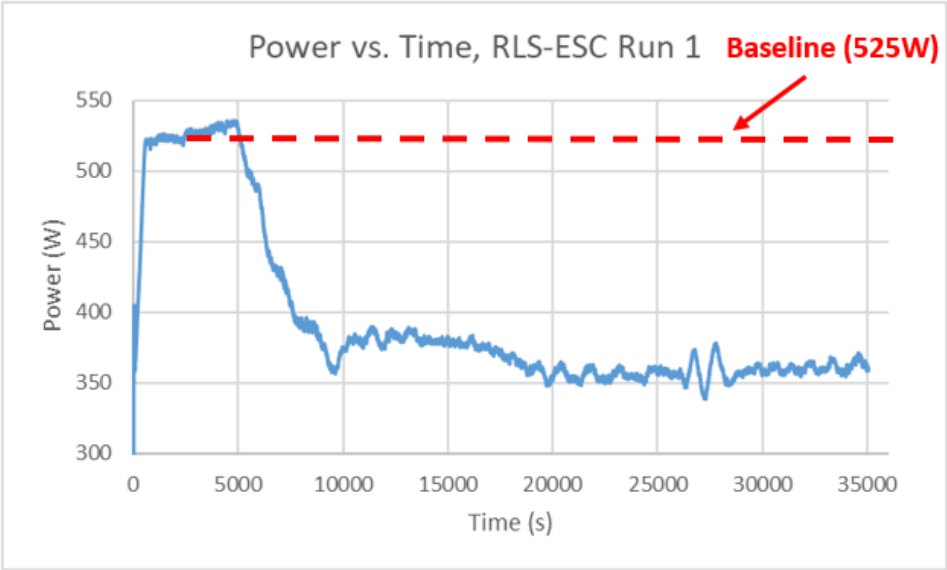

Figure 5.29 Power vs. time for the first RLS-ESC run.

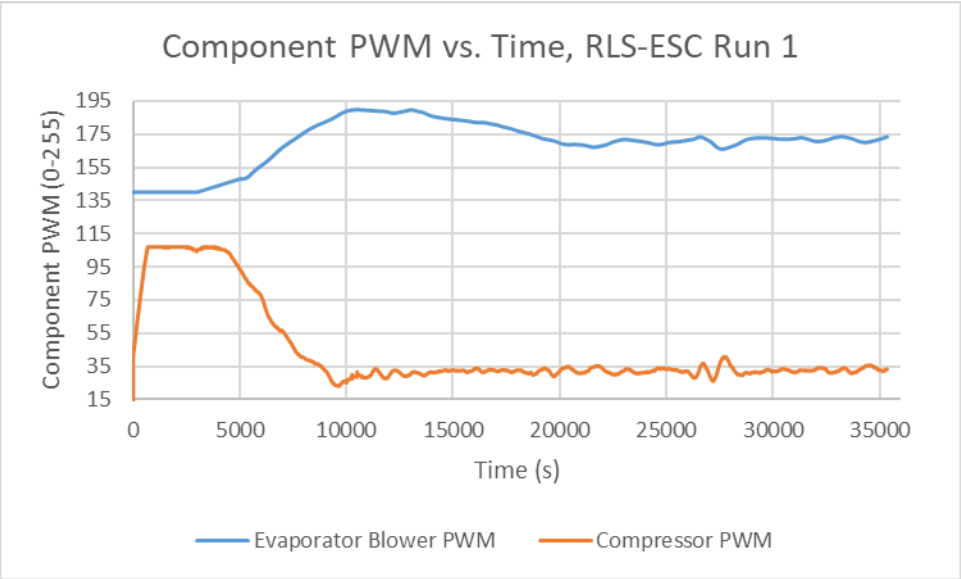

Figure 5.30 Component PWM vs. time for the first RLS-ESC run.

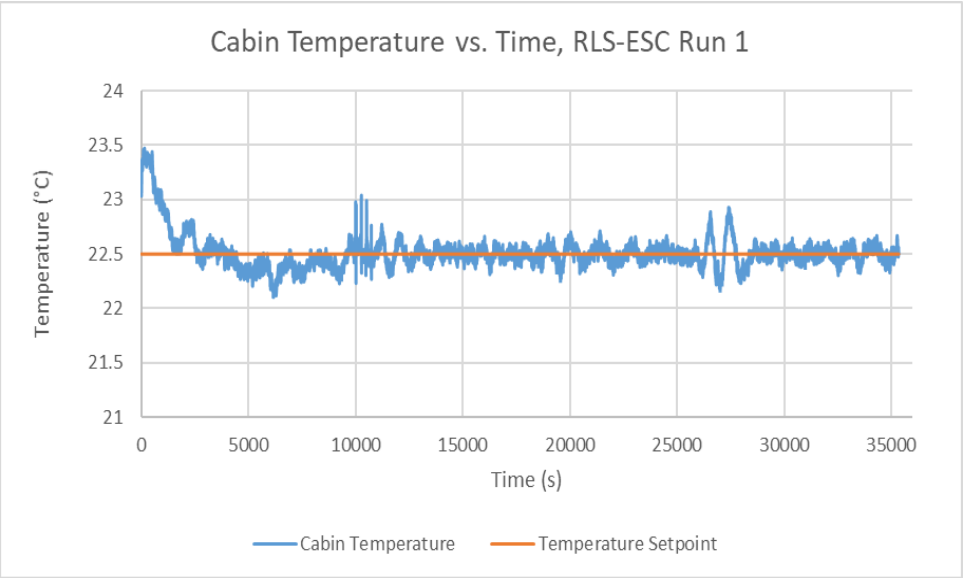

Figure 5.31 Cabin temperature vs. time for the first RLS-ESC run.

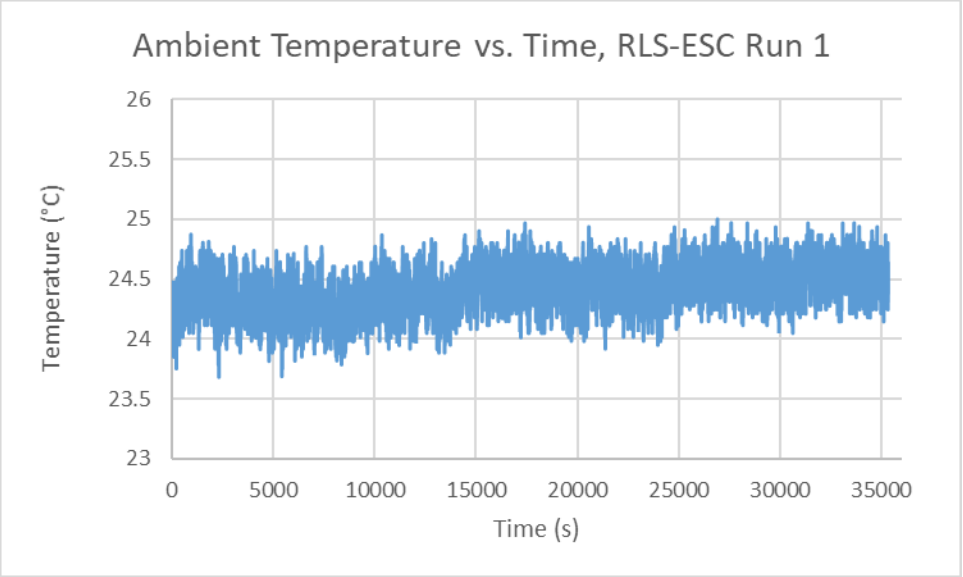

Figure 5.32 Ambient temperature vs. time for the first RLS-ESC run.

Figures 5.33 – 5.37 show the battery charge, power draw, blower and compressor PWM, cabin temperature and ambient temperature for RLS-ESC run 2. This time, the baseline power consumption was 515W with a compressor speed between 90-95 PWM. Like before, after ESC is activated, the evaporator blower speed increases quickly to 180-185 PWM, before settling to a speed around 160-165 PWM. The compressor speed drops to around 30 PWM and the power consumption drops to 330-350W over time. This time though, the cabin temperature showed more oscillatory behavior, although not to the extent seen in LS-ESC. The ambient temperature remained relatively constant through the duration of the experiment.

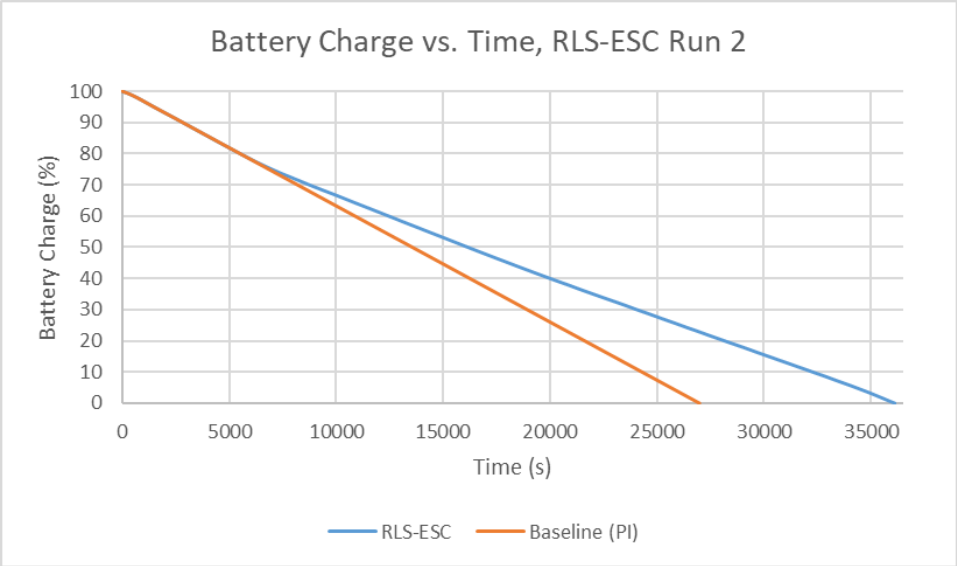

Figure 5.33 Battery charge vs. time for the second RLS-ESC run and its baseline case.

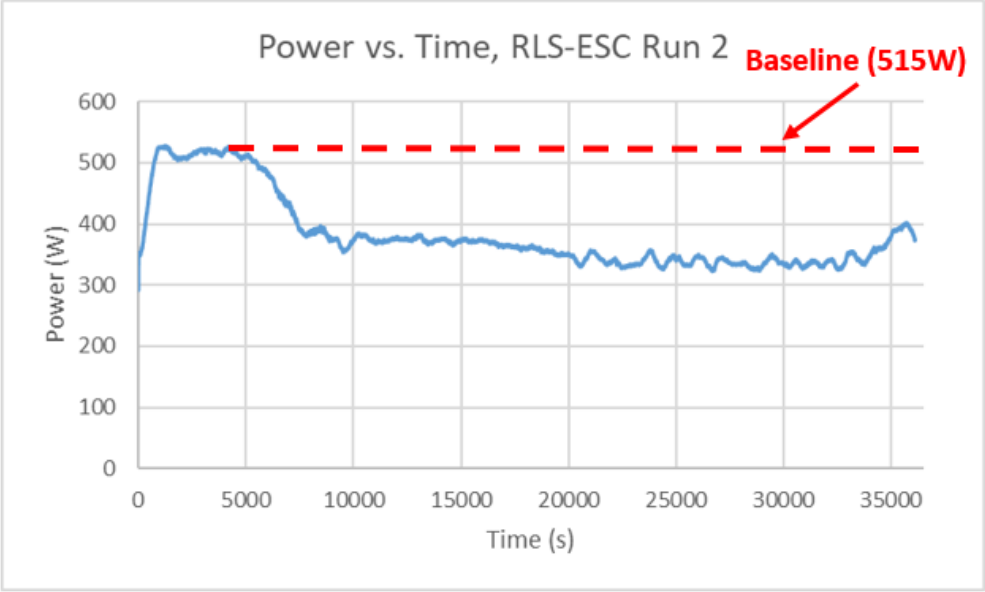

Figure 5.34 Power vs. time for the second RLS-ESC run.

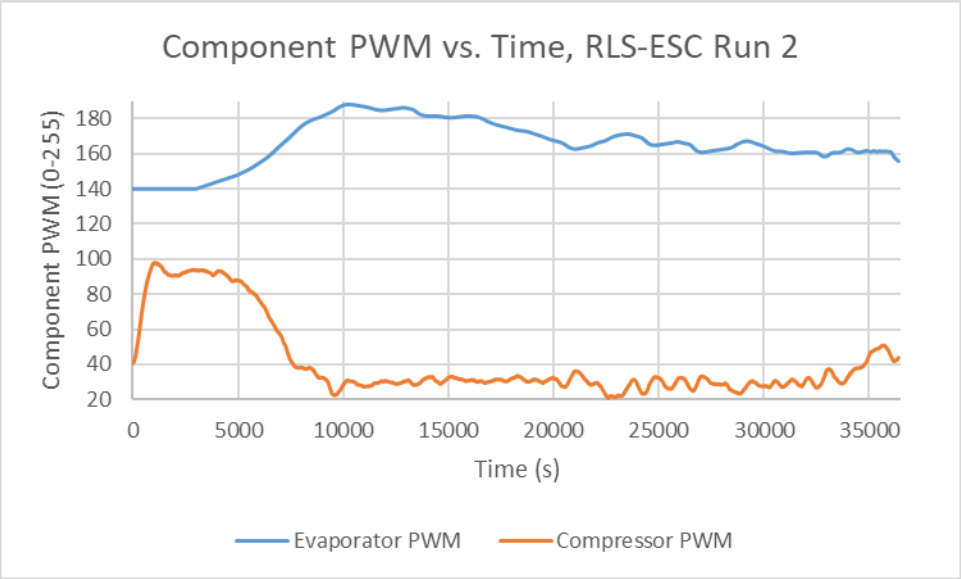

Figure 5.35 Component PWM vs. time for the second RLS-ESC run.

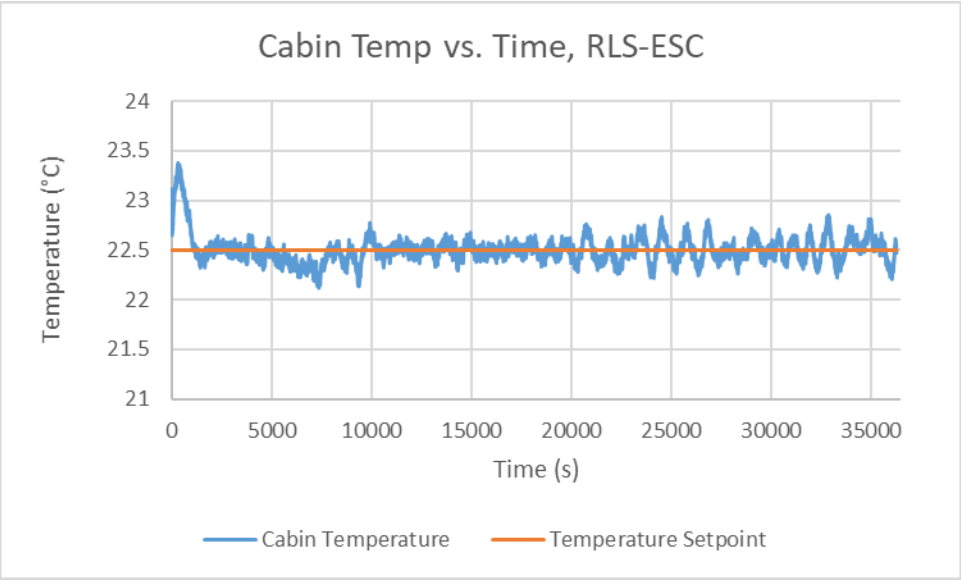

Figure 5.36 Cabin temperature vs. time for the second RLS-ESC run.

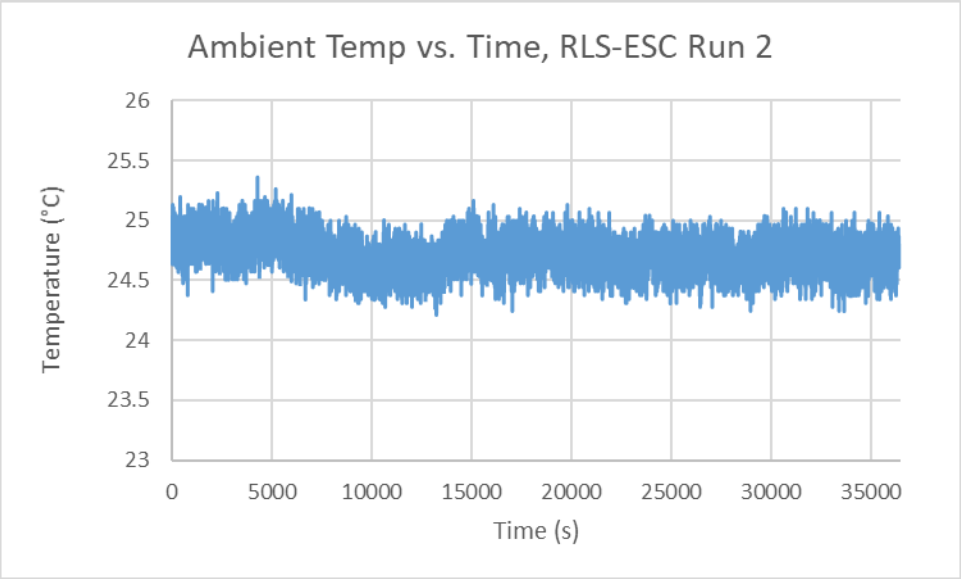

Figure 5.37 Ambient temperature vs. time for the second RLS-ESC run.

5.5 Concluding Remarks

By applying ESC to the experimental system using LabVIEW, we achieved significant reductions in power corresponding with a substantial increase in battery life. All three ESC algorithms were successful in minimizing system power while meeting temperature objectives, with LS-ESC and RLS-ESC demonstrating superior performance over P-ESC. When comparing the performance between these two algorithms, both algorithms extended the battery life by similar amounts amount; however, RLS-ESC tends to induce less oscillatory behavior in the actuator speeds. Therefore, we recommend the use of RLS-ESC when operating the NITE. The LabVIEW interface presented in this chapter is intuitive and easy to read, and tuning parameters is relatively straightforward for an industry practitioner.

Chapter 6
Conclusion

6.1 Thesis Summary and Contributions

This thesis examined the use of ESC, a real time model free optimization method, to minimize the power consumption of a battery powered vehicle VCS. To that end, we developed a model of an integrated VCS in MATLAB/Simulink consisting of a VCS, battery pack, auxiliary fans and a vehicle cabin. In particular, we present a detailed derivation of the vehicle cabin model as it was specifically developed for this thesis. We then presented an open-loop validation of the cabin model based on energy conservation principles along with some limited validation against available experimental data. Next, we detailed the development of the integrated experimental system centered around the NITE, a no-idle battery powered VCS unit developed by Bergstrom. Basic closed loop validation and PI control on the experimental and simulated system was successfully performed. The latter half of this thesis focused on the theory behind ESC and its applications to VCS. We presented a thorough derivation and analysis of three different ESC algorithms and examined their respective tradeoffs. We then implemented ESC on the simulated and experimental integrated unit and achieved significant improvements in battery life.

This thesis yields two main contributions. First, to the author's knowledge, this thesis represents the first application of ESC on battery powered VCS. Second, we developed an intuitive user interface for industry operators to easy apply these algorithms to their systems.

6.2 Future Work

There are a number of opportunities for future work. First, there is much room to improve the models used in this thesis. For the cabin model, a number of parameters such as the vehicle wall thicknesses and material properties are currently hardcoded into the model and

could be parameterized. Secondly, it would be nice to have access to a physical vehicle cabin as part of an integrated experimental system instead of a generic enclosed space which was built for this thesis. Having access to a cabin would also be useful when performing experimental validation of the cabin model in order to ensure vehicle dimensions and material properties are as accurate as possible. Secondly, there was a lack of cross-validation between the simulated and physical integrated NITE system which was a result of the difficulties parameterizing simulated VCS components. Future work should address this gap in order to validate the efficacy of the models and have better predictions of ESC performance.

There is also room for improvement when it comes to ESC implementation. This thesis only considers single-variable extremum seeking, modulating only the evaporator blower speed. However, the condenser fan speed, an important actuator, was left untouched. This means we may not be optimizing the system as well as it could have been. Future work should therefore examine the use of multivariable extremum seeking schemes.

Lastly, all experimental testing was done in a room with mildly varying ambient conditions. In practical application, the NITE and cabin experience more rapidly changing environmental loads and disturbances. In particular, the NITE condenser fan is normally exposed to highly variable ambient conditions due to it being housed outside, whereas we simply shone a heat lamp on it to approximate an ambient heat load. Because ESC is a quasi-steady state optimization algorithm, it could be significantly impacted by these variations and disturbances. Future work should examine ESC robustness and effectiveness with the NITE and cabin being subjected to more variable, real-world conditions.

References

[1] Blackburn Energy, “Could your Truck be Idling Illegally?” [Online]. Available: https://blackburnenergy.com/idling_against_the_law

[2] D. J. Burns and C. Laughman, “Extremum Seeking Control for Energy Optimization of Vapor Compression Systems,” pp. 1–7, 2012.

[3] M Leblanc. Sur l'*é*lectrification des chemins de fer au moyen de courants alternatifs de fr*é*quence *é*lev*é*e. Revue G*é*n*é*rale de l'Electricité, 1922.

[4] M. Krstić and H.-H. Wang, “Stability of extremum seeking feedback for general nonlinear dynamic systems,” *Automatica*, vol. 36, no. 4, pp. 595–601, 2000.

[5] R. Leyva, C. Alonso, I. Queinnec, a Cid-Pastor, D. Lagrange, and L. Martinez-Salamero, “MPPT of photovoltaic systems using extremum-seeking control,” *IEEE Trans. Aerosp. Electron. Syst.*, vol. 42, no. 1, pp. 249–258, 2006.

[6] M. Guay, D. Dochain, and M. Perrier, “Adaptive extremum seeking control of continuous stirred tank bioreactors with unknown growth kinetics,” *Automatica*, vol. 40, no. 5, pp. 881–888, 2004.

[7] J. Creaby, Y. Li, and J. E. Seem, “Maximizing Wind Turbine Energy Capture using Multivariable Extremum Seeking Control,” *Wind Eng.*, vol. 33, pp. 361–387, 2009.

[8] Trojan Battery, “Battery Specifications.” [Online]. Available: http://www.trojanbattery.com/product/31-agm/

[9] D. J. Burns, W. K. Weiss, and M. Guay, “Realtime setpoint optimization with time-varying extremum seeking for vapor compression systems,” *Proc. Am. Control Conf.*, vol. 2015–July, pp. 974–979, 2015.

[10] B. G. B. Hunnekens, M. A. M. Haring, N. Van De Wouw, and H. Nijmeijer, “A dither-free extremum-seeking control approach using 1st-order least-squares fits for gradient estimation,” *Proc. IEEE Conf. Decis. Control*, vol. 2015–February, no. February, pp. 2679–2684, 2014.

[11] Bergstrom Inc., “NITE SSI.” [Online]. Available: https://us.bergstrominc.com/nite-ssi/

[12] A. Alleyne, “THERMOSYS 4 Toolbox,” University of Illinois at Urbana-Champaign, no. March, 2012. [Online]. Available: http://arg.mechse.illinois.edu/thermosys

[13] B. P. Rasmussen and A. G. Alleyne, “Dynamic modeling and advanced control of air conditioning and refrigeration systems,” Ph.D. dissertation, 2006.

[14] D. Marcos, F. J. Pino, C. Bordons, and J. J. Guerra, “The development and validation of a thermal model for the cabin of a vehicle,” *Appl. Therm. Eng.*, vol. 66, no. 1–2, pp. 646–656, 2014.

[15] B. Fletcher and C. Saunders, “Air change in stationary and moving motor vehicles,” *J. Hazard. Mater.*, vol. 38, no. 2, pp. 243–256, 1994.

[16] F. P. Incropera, T. L. Bergman, A. S. Lavine, and D. P. DeWitt, *Fundamentals of Heat and Mass Transfer*. 2011.

[17] B. D. Keating, “Model-Free Real-Time Optimization for Vapor Compression Systems,” 2017.

[18] I. Bayraktar, “Computational simulation methods for vehicle thermal management,” *Appl. Therm. Eng.*, vol. 36, no. 1, pp. 325–329, 2012.

[19] S. Aigarni and D. Nutter, “Survey of sky effective temperature models applicable to building envelope radiant heat transfer,” *ASHRAE Trans.*, vol. 121, no. October, pp. 351–363, 2015.

[20] M. A. Fayazbakhsh and M. Bahrami, “Comprehensive Modeling of Vehicle Air Conditioning Loads Using Heat Balance Method,” no. x, 2013.

[21] AVMA, “Pets in Vehicles.” [Online]. https://www.avma.org/public/PetCare/Pages/pets-in-vehicles.aspx

[22] J. P. Koeln and A. G. Alleyne, “Optimal subcooling in vapor compression systems via extremum seeking control: Theory and experiments,” *Int. J. Refrig.*, vol. 43, pp. 14–25, 2014.

[23] Otexts, “7.3.1 Recursive Least Squares.” [Online]. Available: https://www.otexts.org/1582

[24] B. Hu, Y. Li, F. Cao, and Z. Xing, “Extremum seeking control of COP optimization for air-source transcritical CO2heat pump water heater system,” *Appl. Energy*, vol. 147, pp. 361–372, 2015.

[25] X. Li, Y. Li, J. E. Seem, and P. Li, “Dynamic modeling and self-optimizing operation of chilled water systems using extremum seeking control,” *Energy Build.*, vol. 58, pp. 172–182, 2013.

[26] L. Dong, Y. Li, B. Mu, and Y. Xiao, “Self-optimizing control of air-source heat pump with multivariable extremum seeking,” *Appl. Therm. Eng.*, vol. 84, pp. 180–195, 2015.

[27] Y. Xiao, Y. Li, and J. E. Seem, “Multi-variable Extremum Seeking Control for Mini-split Air-conditioning System,” *Int. Refrig. Air Cond. Conf.*, 2014.

[28] H. Pangborn, “Dynamic Modeling, Validation, and Control for Vapor Compression Systems,” 2015.

[29] Mathworks, “Battery.” [Online]. Available: https://www.mathworks.com/help/physmod/sps/powersys/ref/battery.html

[30] International Trucks, “Prostar.” [Online]. Available: https://www.internationaltrucks.com/trucks/prostar

[31] G. Gelbert, J. P. Moeck, C. O. Paschereit, and R. King, “Advanced algorithms for gradient estimation in one- and two-parameter extremum seeking controllers,” *J. Process Control*, vol. 22, no. 4, pp. 700–709, 2012.

[32] M. Guay and D. Dochain, “A time-varying extremum-seeking control approach,” *Automatica*, vol. 51, pp. 356–363, 2015.

Appendix A

Simulink Diagrams and Code

A.1 Cabin Model

A.1.1 Cabin Model Diagrams

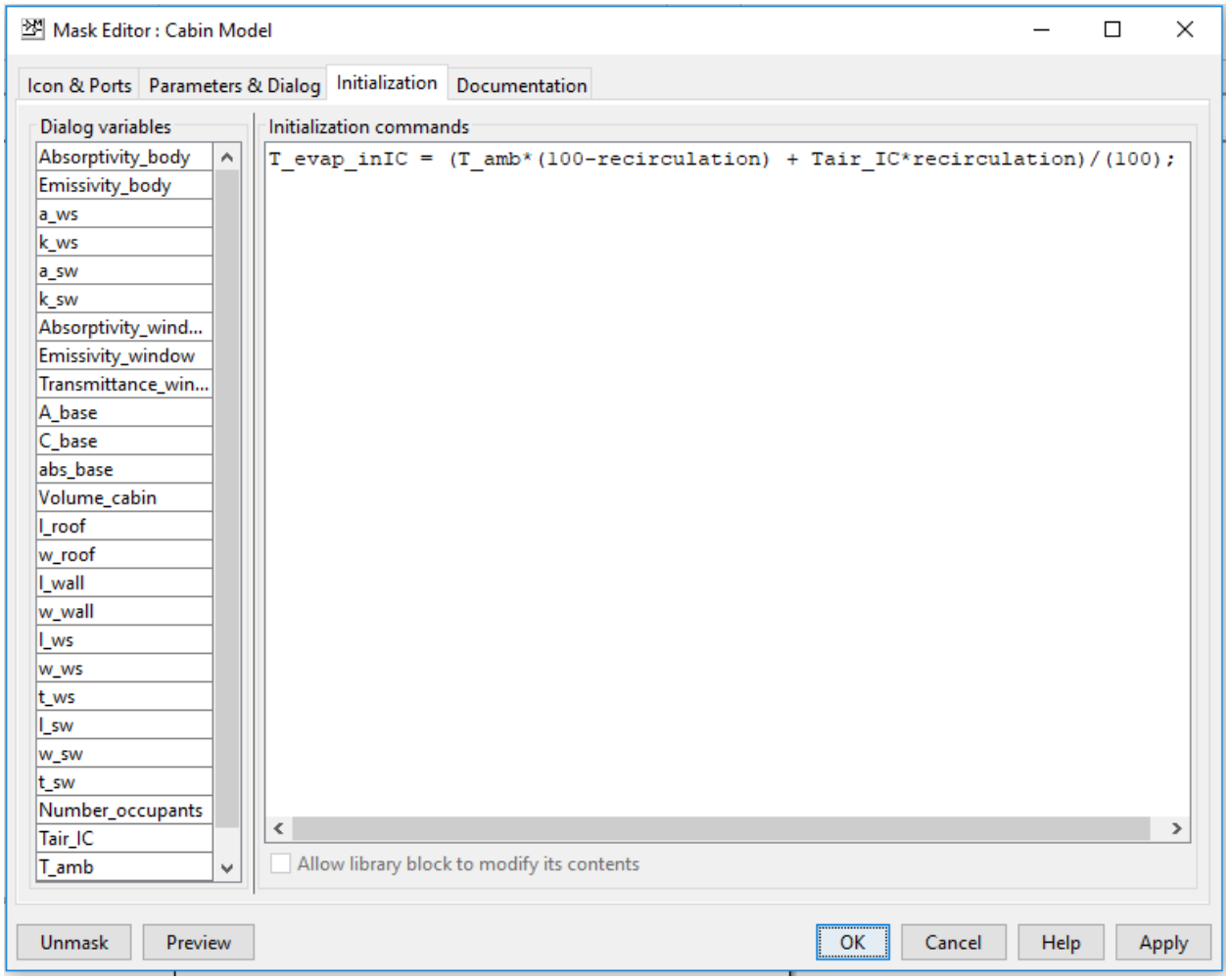

Figure A.1 Initialization of cabin model parameters

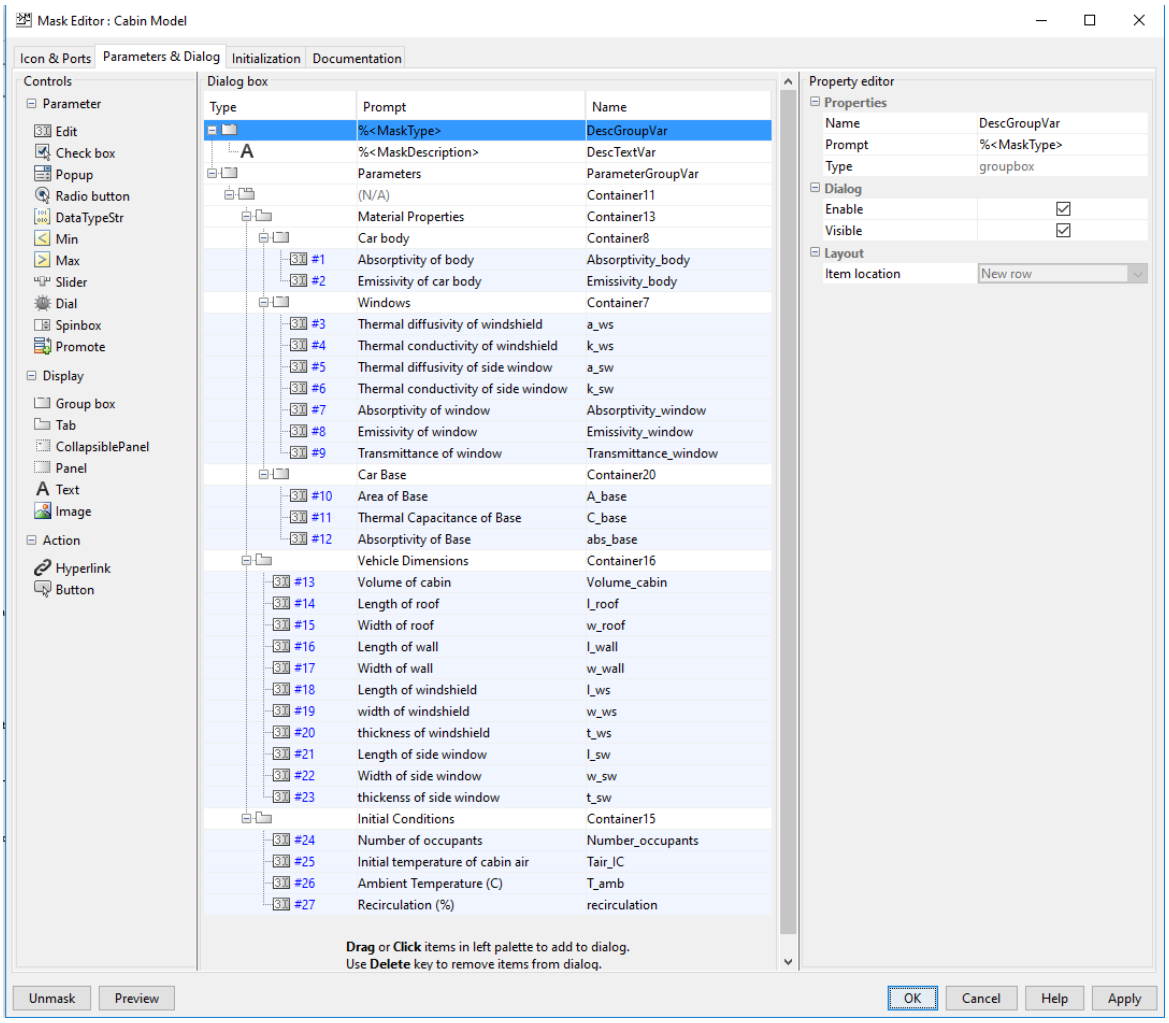

Figure A.2 Cabin Model mask parameters

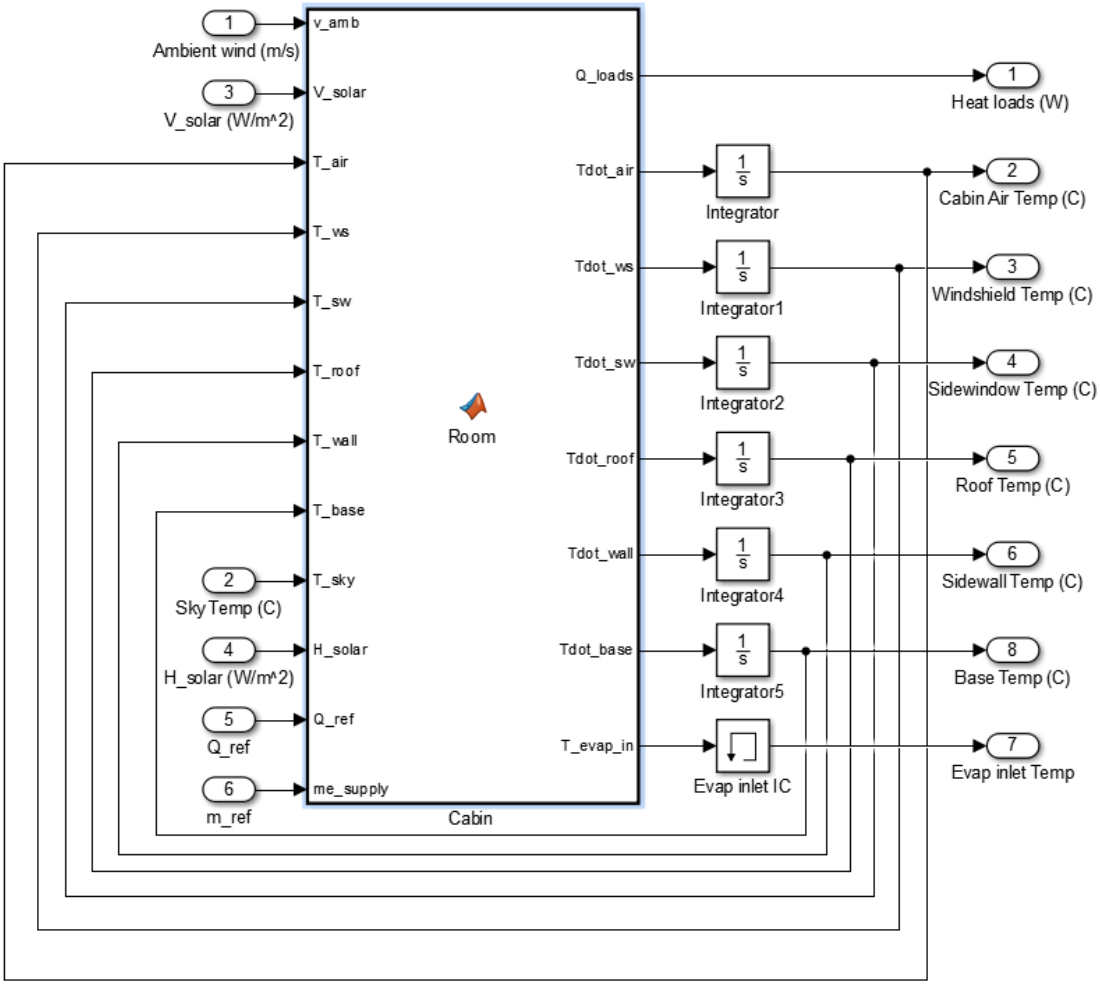

Figure A.3 Underlying Simulink structure underneath cabin model mask.

A.1.2 Cabin Model Code

```
function
[Q_loads,Tdot_air,Tdot_ws,Tdot_sw,Tdot_roof,Tdot_wall,Tdot_base,T_evap_in]
= Room(v_amb,V_solar,T_air,T_ws,T_sw,...
T_roof, T_wall,T_base,T_sky,H_solar,Number_occupants, Emissivity_body,
Emissivity_window,...
Transmittance_window,Volume_cabin, l_ws, w_ws, t_ws, k_ws, a_ws, l_sw,
w_sw, t_sw, k_sw, a_sw,l_roof,w_roof,...
l_wall, w_wall,Q_ref,me_supply, Absorptivity_body, Absorptivity_window,
T_amb,recirculation,A_base, C_base,abs_base)
```

```
%%%%%%%%%%%%%%%%AIR PROPERTIES %%%%%%%%%%%%%%%%%%%%%
AirProp_T = [-40,-20,0,20,40,60];
AirProp_ka = [-73.15 -23.15 26.85 76.85 126.85];
AirProp_rho = [1.516,1.395,1.293,1.204,1.127,1.059];
AirProp_v= [7.59,11.44,15.89,20.92,26.41]*(1e-6);
AirProp_k = [18.1 22.3 26.3 30 33.8]*(1e-3);
AirProp_a = [10.3 15.9 22.5 29.9 38.3]*(1e-6);
AirProp_mu = [132.5 159.6 184.6 208.2 230.1]*(1e-7);
rho_air = interp1(AirProp_T, AirProp_rho, T_air);
a = interp1(AirProp_ka, AirProp_a, T_air);
k_air = interp1(AirProp_ka, AirProp_k, T_air) ;
v = interp1(AirProp_ka, AirProp_v, T_air);
mu = interp1(AirProp_ka,AirProp_mu, T_amb) ;

%internal cabin air properties
Volume_air = Volume_cabin - Number_occupants*0.071;%0.071 = volume of 160
pound human (m^3)
m_air = Volume_air*rho_air; %mass of the air
cp_air = 1007; %J/kg*K
Pr = 0.707; %Prandtl Number

%vehicle dimensions
A_roof = l_roof*w_roof;
A_sw = l_sw*w_sw;
A_ws = l_ws*w_ws;
A_wall = l_wall*w_wall;
sigma = 5.67*10^-8; %stefan-boltzmann constant

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate Conductive Windshield Heat Load%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dx = t_ws/4; %discretize windshield width into 4 nodes.
q_ws = H_solar*Absorptivity_window; %incident absorbed solar radiation
Re_wse = rho_air*v_amb*l_ws/mu;  %Reynolds number of windshield
Tf_wse = (T_amb + T_ws(1))/2 +273.15; %exterior film temperature
l_c_ws = A_ws/(2*l_ws + 2*w_ws); %characteristic length A/P
Ra_wse = abs(9.81*(1/Tf_wse)*(T_amb - T_ws(1))*(l_c_ws).^3/(v*a));
%rayleigh number

%calculate external heat transfer coefficient
if Re_wse == 0

    Nu_wse = (0.825 + (0.387*Ra_wse.^(1/6))/(1 +
(0.492/Pr)^(9/16)).^(8/27)).^2;
    h_wse = Nu_wse*(k_air/l_ws); %(no wind) external free hxfr coefficient


elseif Re_wse < 5*10^5

    Nu_wse = 0.664*Re_wse.^(1/2)*Pr.^(1/3);
```

```
    h_wse = Nu_wse*(k_air/l_ws); %external laminar hxfr coefficient

else

    Nu_wse = (0.037*Re_wse.^(4/5) - 871)*Pr.^(1/3);
    h_wse = Nu_wse*(k_air/l_ws); %external turbulent/transition hxfr
coefficient

end

%Calculate inside windshield heat transfer coefficient
Tf_ws = (T_ws(5) + T_air)/2 + 273.15; %Kelvin
Ra_ws = abs(9.81*(1/Tf_ws)*(T_ws(5) - T_air)*(l_c_ws).^3/(v*a)) ;%interior
rayleigh number
Nu_ws = (0.825 + (0.387*Ra_ws.^(1/6))/(1 + (0.492/Pr)^(9/16)).^(8/27)).^2;
h_ws = Nu_ws*(k_air/l_ws) ;%free convection hxfr coefficient for
windshield interior

%Finite Element Discretization of Windshield Thickness
Tdot_ws = zeros(1,5);
Tdot_ws(1) = (2*q_ws*a_ws)/(k_ws*dx) + (2*a_ws*h_wse/(k_ws*dx))*(T_amb-
T_ws(1)) + (2*a_ws/dx)*(T_ws(2)-T_ws(1)) +...
(2*a_ws*Emissivity_window*sigma/(k_ws*dx))*((T_sky+273.15)^4-
(T_ws(1)+273.15)^4);
Tdot_ws(2) = (a_ws/(dx^2))*(T_ws(1)-T_ws(2)) + (a_ws/(dx^2))*(T_ws(3)-
T_ws(2));
Tdot_ws(3) = (a_ws/(dx^2))*(T_ws(2)-T_ws(3)) + (a_ws/(dx^2))*(T_ws(4)-
T_ws(3));
Tdot_ws(4)= (a_ws/(dx^2))*(T_ws(3)-T_ws(4)) + (a_ws/(dx^2))*(T_ws(5)-
T_ws(4));
Tdot_ws(5) = (2*a_ws*h_ws/(k_ws*dx))*(T_air-T_ws(5)) +
(2*a_ws)/(dx^2)*(T_ws(4)-T_ws(5));
Q_ws = h_ws*A_ws*(T_ws(5) - T_air); %conductive heat transfer through
windshield


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate Conductive Side Window Heat Load%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dx = t_sw/4;
q_sw = H_solar*Absorptivity_window;
Re_swe = rho_air*v_amb*l_sw/mu;
l_c_sw = A_sw/(2*l_sw + 2*w_sw); %characteristic length A/P

if Re_swe == 0

    Tf_swe = (T_sw(1)+T_amb)/2 + 273.15; %film temperature
    Ra_swe = abs(9.81*(1/Tf_swe)*(T_sw(1) - T_amb)*(l_c_sw).^3/(v*a));
%rayleigh number
```

```
    Nu_swe = (0.825 + (0.387*Ra_swe.^(1/6))/(1 +
(0.492/Pr)^(9/16)).^(8/27)).^2;
    h_swe = Nu_swe*(k_air/l_sw);

elseif Re_swe < 5*10^5

    Nu_swe = 0.664*Re_swe.^(1/2)*Pr.^(1/3);
    h_swe = Nu_swe*(k_air/l_sw) ;%external laminar hxfr coefficient

else
    Nu_swe = (0.037*Re_swe.^(4/5) - 871)*Pr.^(1/3);
    h_swe = Nu_swe*(k_air/l_sw) ;%external turbulent/transition hxfr
coefficient

end

%include inner side window condition
Tf_sw = (T_sw(5) + T_air)/2 + 273.15; %Kelvin
Ra_sw = abs(9.81*(1/Tf_sw)*(T_sw(5) - T_air)*(l_c_sw).^3/(v*a)) ;%interior
rayleigh number
Nu_sw = (0.825 + (0.387*Ra_sw.^(1/6))/(1 + (0.492/Pr)^(9/16)).^(8/27)).^2;
h_sw = Nu_sw*(k_air/w_sw) ;%free convection hxfr coefficient for
windshield interior

Tdot_sw = zeros(1,5);
Tdot_sw(1) = (2*q_sw*a_sw)/(k_sw*dx)+(2*a_sw*h_swe/(k_sw*dx))*(T_amb-
T_sw(1)) + (2*a_sw/dx)*(T_sw(2)-T_sw(1)) +
(2*a_sw*Emissivity_window*sigma/(k_sw*dx))*((T_sky+273.15)^4-
(T_sw(1)+273.15)^4);
Tdot_sw(2) = (a_sw/(dx^2))*(T_sw(1)-T_sw(2)) + (a_sw/(dx^2))*(T_sw(3)-
T_sw(2));
Tdot_sw(3) = (a_sw/(dx^2))*(T_sw(2)-T_sw(3)) + (a_sw/(dx^2))*(T_sw(4)-
T_sw(3));
Tdot_sw(4)= (a_sw/(dx^2))*(T_sw(3)-T_sw(4)) + (a_sw/(dx^2))*(T_sw(5)-
T_sw(4));
Tdot_sw(5) = (2*a_sw*h_sw/(k_sw*dx))*(T_air-T_sw(5)) +
(2*a_sw)/(dx^2)*(T_sw(4)-T_sw(5));
Q_sw = h_sw*A_sw*(T_sw(5) - T_air); %conductive heat transfer through
windshield

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate Conductive Roof Heat Load%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

q_roof = V_solar*Absorptivity_body;
l_c_roof = A_roof/(2*l_roof + 2*w_roof); %characteristic length A/P
Tf_ceil = (T_air + T_roof(1))/2 + 273.15; %ceiling film temp (Kelvin)
Re_roof = rho_air*v_amb*l_roof/mu; %reynolds number
Ra_ceil = abs((9.81*(1/Tf_ceil)*(T_roof(5) - T_air)*(l_c_roof).^3)/(v*a));

if Re_roof == 0
```

```
    Tf_roof = (T_amb + T_roof(1))/2 + 273.15; %roof film temperature
(Kelvin)
    Ra_roof = abs((9.81*(1/Tf_roof)*(T_amb -
T_roof(1))*(l_c_roof).^3)/(v*a)); %rayleigh number for roof

    if T_roof(1) > T_amb

        if Ra_roof < 10^7

            Nu_roof = 0.54*Ra_roof^(1/4);
            h_roof = Nu_roof*(k_air/l_c_roof); %free convection hxfr
coefficient (laminar)

        else

            Nu_roof = 0.15*Ra_roof^(1/3);
            h_roof = Nu_roof*(k_air/l_c_roof); %free convection hxfr
coefficient (turbulent)

        end

    else

        Nu_roof = 0.52*Ra_roof^(1/5);
        h_roof = Nu_roof*(k_air/l_c_roof); %free convection hxfr
coefficient

    end

elseif Re_roof < 5*10^5

    Nu_roof = 0.664*Re_roof.^(1/2)*Pr.^(1/3);
    h_roof = Nu_roof*(k_air/l_roof); %external laminar hxfr coefficient

else

    Nu_roof = (0.037*Re_roof.^(4/5) - 871)*Pr.^(1/3);
    h_roof = Nu_roof*(k_air/l_roof); %external turbulent/transition hxfr
coefficient

end
%inner ceiling heat transfer coefficient

if T_roof(5) > T_air

        Nu_ceil = 0.52*Ra_ceil^(1/5);
        h_ceil = Nu_ceil*(k_air/l_c_roof) ;

    else
```

```
        if Ra_ceil < 10^7

            Nu_ceil = 0.54*Ra_ceil^(1/4);
            h_ceil = Nu_ceil*(k_air/l_c_roof) ;

        else

            Nu_ceil = 0.15*Ra_ceil^(1/3);
            h_ceil = Nu_ceil*(k_air/l_c_roof) ;

        end

end


a_steel = 3.954e-6;
k_steel = 14.9;
a_cotton = 2.76e-8;
k_cotton = 0.06;
dx = 5e-4/1.5;
dx1 = 1e-4;
dx2 = 5e-3/1.5;

Tdot_roof = zeros(1,5);
Tdot_roof(1) =
(2*q_roof*a_steel)/(k_steel*dx)+(2*a_steel*h_roof/(k_steel*dx))*(T_amb-
T_roof(1)) + (2*a_steel/dx)*(T_roof(2)-T_roof(1)) +
(2*a_steel*Emissivity_body*sigma/(k_steel*dx))*((T_sky+273.15)^4-
(T_roof(1)+273.15)^4);
Tdot_roof(2) = (a_steel/(dx^2))*(T_roof(1)-T_roof(2)) +
(a_steel/(dx^2))*(T_roof(3)-T_roof(2));
Tdot_roof(3) = (a/(dx1^2))*(T_roof(2)-T_roof(3)) + (a/(dx1^2))*(T_roof(4)-
T_roof(3));
Tdot_roof(4)= (a_cotton/(dx2^2))*(T_roof(3)-T_roof(4)) +
(a_cotton/(dx2^2))*(T_roof(5)-T_roof(4));
Tdot_roof(5) = (2*a_cotton*h_ceil/(k_cotton*dx2))*(T_air-T_roof(5)) +
(2*a_cotton)/(dx2^2)*(T_roof(4)-T_roof(5));

Q_roof = h_ceil*A_roof*(T_roof(5)-T_air);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate Conductive Vehicle Side Walls Heat Load%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

q_wall = H_solar*Absorptivity_body;
Re_walle = rho_air*v_amb*l_wall/mu;  %Reynolds number of sides
Tf_walle = (T_amb + T_wall(1))/2 +273.15; %exterior film temperature
l_c_wall = A_wall/(2*l_wall + 2*w_wall); %characteristic length A/P
Ra_walle = abs(9.81*(1/Tf_walle)*(T_amb - T_wall(1))*(l_c_wall).^3/(v*a));
%rayleigh number
```

```
%Calculate outer side wall condition

if Re_walle == 0 %free convection

    Nu_walle = (0.825 + (0.387*Ra_walle.^(1/6))/(1 +
(0.492/Pr)^(9/16)).^(8/27)).^2;
    h_walle = Nu_walle*(k_air/l_wall);


elseif Re_walle < 5*10^5 %laminar flow

    Nu_walle = 0.664*Re_walle.^(1/2)*Pr.^(1/3);
    h_walle = Nu_walle*(k_air/l_wall); %external laminar hxfr coefficient

else %turbulent flow

    Nu_walle = (0.037*Re_walle.^(4/5) - 871)*Pr.^(1/3);
    h_walle = Nu_walle*(k_air/l_wall); %external turbulent/transition hxfr
coefficient

end

%include inner side wall condition
Tf_wall = (T_wall(5) + T_air)/2 + 273.15; %Kelvin
Ra_wall = abs(9.81*(1/Tf_wall)*(T_wall(5) - T_air)*(l_c_wall).^3/(v*a))
;%interior rayleigh number
Nu_wall = (0.825 + (0.387*Ra_wall.^(1/6))/(1 +
(0.492/Pr)^(9/16)).^(8/27)).^2;
h_wall = Nu_wall*(k_air/w_wall) ;%free convection hxfr coefficient for
side wall interior

%Finite element Method: Outer surface=1, Inner surface=5
Tdot_wall = zeros(1,5);
Tdot_wall(1) =
(2*q_wall*a_steel)/(k_steel*dx)+(2*a_steel*h_walle/(k_steel*dx))*(T_amb-
T_wall(1)) + ...
(2*a_steel/dx)*(T_wall(2)-T_wall(1)) +
(2*a_steel*Emissivity_body*sigma/(k_steel*dx))*((T_sky+273.15)^4-
(T_wall(1)+273.15)^4);
Tdot_wall(2) = (a_steel/(dx^2))*(T_wall(1)-T_wall(2)) +
(a_steel/(dx^2))*(T_wall(3)-T_wall(2));
Tdot_wall(3) = (a/(dx1^2))*(T_wall(2)-T_wall(3)) + (a/(dx1^2))*(T_wall(4)-
T_wall(3));
Tdot_wall(4)= (a_cotton/(dx2^2))*(T_wall(3)-T_wall(4)) +
(a_cotton/(dx2^2))*(T_wall(5)-T_wall(4));
Tdot_wall(5) = (2*a_cotton*h_wall/(k_cotton*dx2))*(T_air-T_wall(5)) +
(2*a_cotton)/(dx2^2)*(T_wall(4)-T_wall(5));

Q_wall = h_wall*A_wall*(T_wall(5)-T_air);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Air flow to the evaporator inlet %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Assuming Mass flow is set by evaporator block and not initialized here.
mdot_return = (recirculation/100)*me_supply; %recirculated room air
through the evaporator. (kg/s)
mdot_amb = (100-recirculation)/100*me_supply; %fresh air pulled through
evaporator.
T_evap_in = (T_amb*mdot_amb + T_air*mdot_return)/(mdot_amb + mdot_return);
%Temperature of air @evaporator inlet.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Vehicle Base Heat load %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    l_c_base = sqrt(A_base); %assuming the base is a square.
    Tf_base = (T_air + T_base)/2 + 273.15; %roof film temperature (Kelvin)
    Ra_base = abs((9.81*(1/Tf_base)*(T_air -
T_base)*(l_c_base).^3)/(v*a)); %rayleigh number for base

    if T_base > T_air

        if Ra_base < 10^7

            Nu_base = 0.54*Ra_base^(1/4);
            h_base = Nu_base*(k_air/l_c_base); %free convection hxfr
coefficient (Rayleigh laminar)

        else

            Nu_base = 0.15*Ra_base^(1/3);
            h_base = Nu_base*(k_air/l_c_base); %free convection hxfr
coefficient (Rayleigh turbulent)

        end

    else

        Nu_base = 0.52*Ra_base^(1/5);
        h_base = Nu_base*(k_air/l_c_base); %free convection hxfr
coefficient

    end

Q_incident = Transmittance_window*(H_solar)*(2*A_sw + A_ws); %Transmitted
Solar flux into Vehicle.

Q_base = h_base*A_base*(T_base-T_air); %heat transfer b/w base and air.
```

```
Tdot_base = (1/(C_base))*(abs_base*Q_incident - Q_base);

%%%%%%%%%%%%%%%%%%%%%%%%
%%% Other Heat Loads %%%
%%%%%%%%%%%%%%%%%%%%%%%%

ACH = 0.8*v_amb; %rough ACH as a result of ambient air infiltration into
the vehicle. See Flectcher and Saunders (1994) for more information.
Q_infil = (ACH/3600)*(Volume_air*rho_air*cp_air)*(T_amb - T_air); %heat
load as a result of infiltrating air
Q_human = Number_occupants*108; %human heat load
Tdot_air = (1/(m_air*cp_air))*(Q_base + Q_ws + 2*Q_sw + Q_roof + Q_human+
Q_infil + Q_ref +2*Q_wall);
Q_loads = [Q_ws; 2*Q_sw; Q_roof; Q_infil; Q_human; Q_ref; 2*Q_wall;
Q_base];
```

A.2 Auxiliary Models Code

A.2.1 Condenser Fan Code

```
function [mdot_cond,power_cond] = cond(fanspeed)

fanspeeds = [120.0000
100.0000
85.0000
70.0000
60.0000
50.0000
40.0000
20.0000]; %0-255 PWM

flowrates = [0.304631598
0.371971406
0.422476112
0.472980818
0.506650622
0.540320425
0.573990229
0.64132968]; %mass flowrates (kg/s)

powers = [28
40
50
64.5
79
89
107
134]; %Watts
```

```
mdot_cond = interp1(fanspeeds,flowrates,fanspeed,'linear');

power_cond = interp1(fanspeeds,powers,fanspeed,'spline');

return
```

A.2.2 Blower Fan Code

```
function [mdot_evap,power_evap] = evap(fanspeed)


fanspeeds = 195.0000
190.0000
180.0000
170.0000
160.0000
150.0000
140.0000
130.0000
120.0000
110.0000
100.0000]; %0-255 PWM

flowrates = [0.108345398
0.106956354
0.100011136
0.092510301
0.0861207
0.0797311
0.073619308
0.068340943
0.063062578
0.05191422
0.0495544
]; %mass flowrates (kg/s)

powers = [147, 139, 119, 100, 86, 73, 59, 49, 41 35 28]; %Watts

%mass flowrates (kg/s)
mdot_evap = interp1(fanspeeds,flowrates,fanspeed,'linear');
power_evap = interp1(fanspeeds,powers,fanspeed,'spline');

return
```

A.3 Control Algorithm Code

A.3.1 RLS Algorithm Code

```
function [V1,B1,gradient] = fcn(evap,power,V0,B0,lambda)
Xn = [evap 1]; %New evap fan data
yn = power; %new power data
V1 = (1/lambda)*(V0 - V0*transpose(Xn)*Xn*V0/(1+Xn*V0*transpose(Xn)));
gamma_1 = V1*transpose(Xn);
e = yn - Xn*B0; %new error
B1= B0 + gamma_1*e; %new linear approx. coefficients
gradient = B1(1); %obtain slope value
return
```

A.3.2 LS-ESC Code

```
function gradient = fcn(power,evap)
a = max(evap); %maximum evap value in data buffer
b = min(evap); %mass flowrates (kg/s)

coeff = polyfit(evap,power,1); %perform linear fit on data & obtain
coefficients
gradient = coeff(1); %obtain slope value

if abs(a-b) ==0; %turn off gradient calculation if data is not rich enough
    gradient=0;
end
return
```

Appendix B
Experimental System Hardware and Software

This section details the specific hardware and software configuration used in the development of the integrated experimental system.

B.1 Experimental Hardware Setup

The experimental system is outfitted with transducers, circuitry and data acquisition hardware to obtain relevant data such as temperature and pressure, as well as to facilitate communication between the NITE system and the computer. The National Instruments cRIO-9035 is the central piece of hardware that performs all of the above tasks and more. This data acquisition controller comes with a chassis which allows the user to add various I/O modules to send and receive different types of data. As discussed briefly in chapter 2 (see Fig. 2.32), this thesis uses the NI-9205 analog input module to read analog sensor data, and the NI-9862 CAN module to send and receive CAN signals between the computer and the NITE system. We discuss each module's setup and configuration below.

B.1.1 NI-9205 Module Setup and Configuration

The first step is to download all requisite software onto the NI cRIO. This is done using NI Max (Measurement and Automation Explorer), software which can be used to easily interface with NI hardware. Download and setup NI Max, and then browse to the module specific page on the National Instruments website to find the software needed. Insert the module into an empty chassis slot. Next, open up a new LabVIEW project file. In the project tree, right click on the project file (ends with .lvproj) and select new targets and devices. Under the Real-Time CompactRIO folder, the NI-cRIO in use should be listed there. Once adding it to the project tree, expand the Chassis sub-tree to view the NI 9205 module.

The NI 9205 Module has 32 analog input channels with high degrees of accuracy and protection from overvoltage. To read a signal, one simply needs to wire a sensor's output signal to any of the 32 input channels (AI0-AI31) and wire the sensor ground to the module's ground. To read this signal in the LabVIEW workspace, simply open the NI 9205 module sub-tree and drag the respective input channel into your VI file. Fig. B.1 illustrates this procedure.

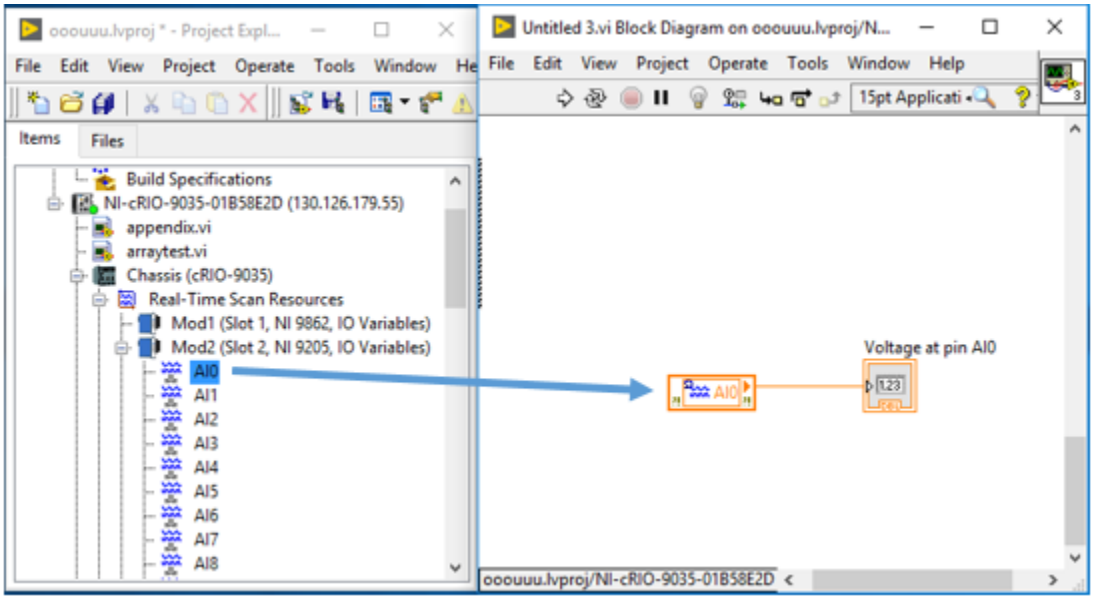

Figure B.1 Reading an analog input pin in LabVIEW by dragging it into the VI.

B.1.2 Sensor Wiring Diagram

The LM35 analog temperature sensor, along with the pressure sensor, are highly susceptible to noise and interference. Therefore, prior to sending the sensor's output signal to the NI 9205, we pass them through a signal conditioning breadboard seen previously in Chapter 2. This breadboard provides 5V power and a reference ground to the sensor's power and ground pins respectively via a connection to an external 5V power supply, and also contains seven low pass filters through which each of the seven signals passes through to attenuate noise. The cutoff frequency of the low pass filters is approximately 7 Hz, which is low enough to attenuate most noise induced by power supplies and other external disturbances. Note that the pressure transducer is powered separately by the NITE's 12V supply. Fig B.2 shows the wiring diagram of the signal conditioning breadboard.

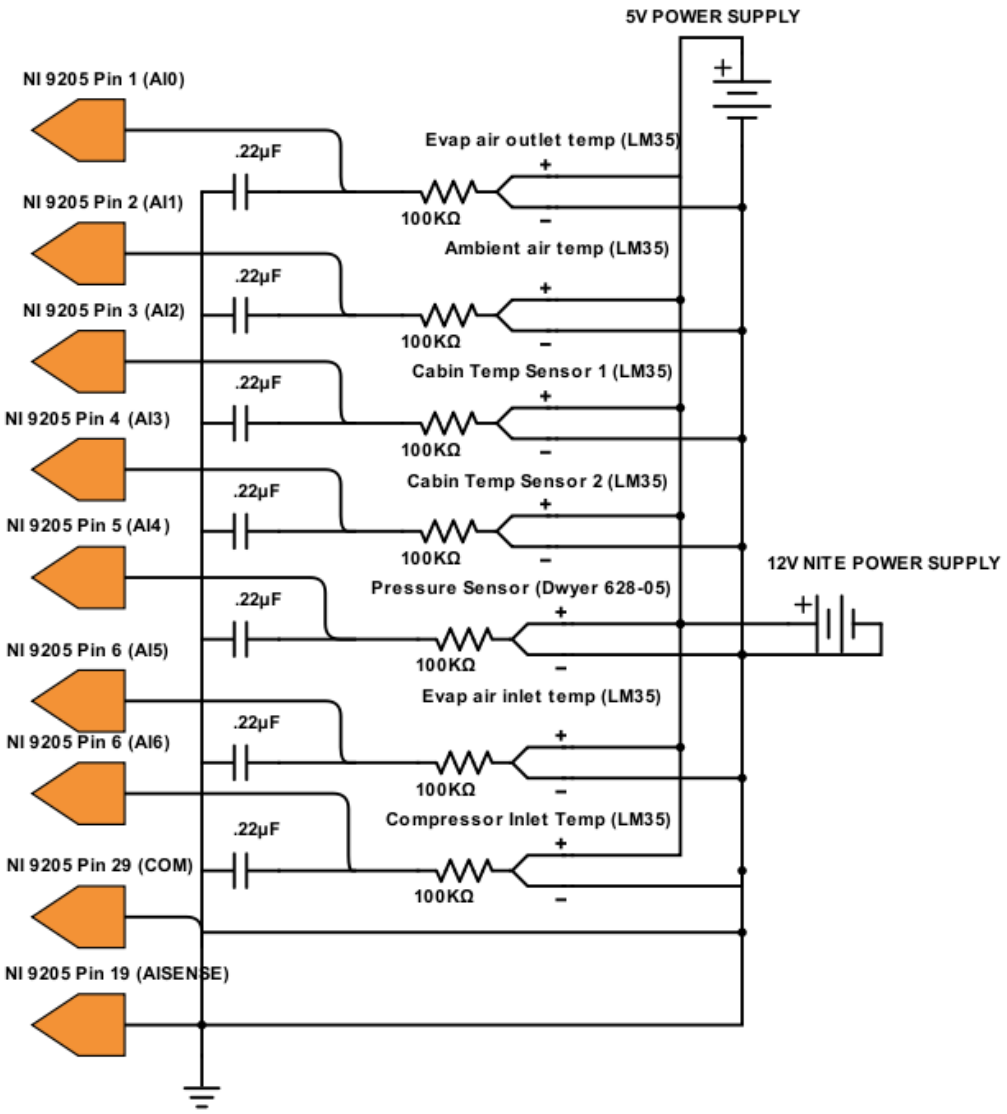

Figure B.2 Wiring schematic of the signal conditioning breadboard

B.1.3 NI-9862 Module Setup and Configuration

Setting up and configuring the NI-9862 to send and receive CAN signals is slightly more complicated than setting up the NI-9205. Thankfully, the procedure is well detailed in

the NI-XNET manual available online. Before summarizing the procedure, we first discuss the fundamentals of the CAN bus protocol.

B.1.3.1 CAN Bus

CAN bus is a communication protocol standard developed to facilitate communication between microcontrollers and other devices within a vehicle without the need for a supervisory, central host computer. CAN was first developed by Bosch in the 1980s and has since been ubiquitous in vehicle systems. The CAN bus protocol is standardized by ISO-11898.

CAN utilizes a multi-master serial bus structure, where all vehicle microcontrollers and devices are all connected to a two wire bus. The wires are called CANH and CANL, the high and low voltage lines respectively. To send messages on the bus, microcontrollers modulate the high and low voltage lines accordingly which correspond to sending speicifc bits of data. The first part of a CAN message is referred to as the “identifier” which identifies the message source and also establishes its priority on the bus. The specific payload data then follows this identifier. Identifiers are important in order for microcontrollers to distinguish which messages it needs to pay attention or respond to since all data communication is visible to all devices on the CAN bus.

Because the NITE system is used in vehicle systems, it too adheres to the CAN standard. For this thesis, Bergstrom provided a document listing all CAN messages sent and received by the NITE, as well as details on how often messages are sent and at what baud rate. There are three main messages of interest: the battery parameters message, the system broadcast message and the overriding command message. The battery parameters message tells us the NITE's voltage and current draw, the system broadcast message includes a variety of information such as component speed and the override command message allows the user to set the speed of the condenser fan, evaporator blower and compressor actuators.

B.1.3.2 NI-9862 Setup

The NI-9862 is a single port high speed CAN transceiver module with the ability to send and receive CAN signals. To connect the module to the existing CAN bus simply splice and solder the CAN-H and CAN-L lines from your system to the appropriate pins on the NI

9862 9-pin D-sub port. The NI-9862 requires a 9-30V power supply to operate, so solder the power and ground wires to the module's pins accordingly. Detailed diagrams and instructions can be found online in the getting started guide for the NI-9862 available on the NI website.

Once the module is set up, configure the NI-9862 by following the procedure outlined in Chapter 2 of the NI-XNET manual. Specifically, closely follow the instructions under the "Getting started with CompactRIO" section to add the module to your LabVIEW project as done for the NI-9205.

B.1.3.3 Reading, Writing and Manipulating CAN Frames in LabVIEW

With the hardware set up, the next step is to use LabVIEW to read, write and manipulate CAN messages (also known as CAN frames). As discussed in Chapter 5, we wish to read the NITE's power consumption and compressor speed, and also wish to write overriding messages to the NITE to control its actuators (the condenser fan, evaporator blower and compressor speeds). Given that we know what the messages are, how can we use LabVIEW to read and write such messages? A brief summary is presented below, but more detailed information is provided in Chapter 4 of the NI-XNET manual.

First, make sure you have added the NI-XNET toolbox to your LabVIEW. Right click on the LabVIEW block diagram, select Measurement I/O, select XNET and then select the "Create Session" VI. This VI initializes a CAN session and also determines whether to read or write CAN frames. In most instances, one would select "Frame In Single Point" to read CAN frames, and "Frame Out Single Point" to write CAN frames. Having done that, the next step is to give the Create Session VI a list of frames to read or write and also select the CAN interface through which the signals will be sent. The name of the CAN interface can be found in NI-MAX. To specify the frames to read/write, right click on the frame list control in the front panel and select "New XNET Database". Select "New Cluster". Select the desired protocol (CAN) and press ok. Now, specify the message's name and baud rate (this is determined by the specific system one is interfacing with. The NITE's baud rate is 250 kBaud). Now, one can add frames to read and write by right clicking on the cluster and selecting "Create Frame". Some examples of this can be seen in B.3 – B.6 below.

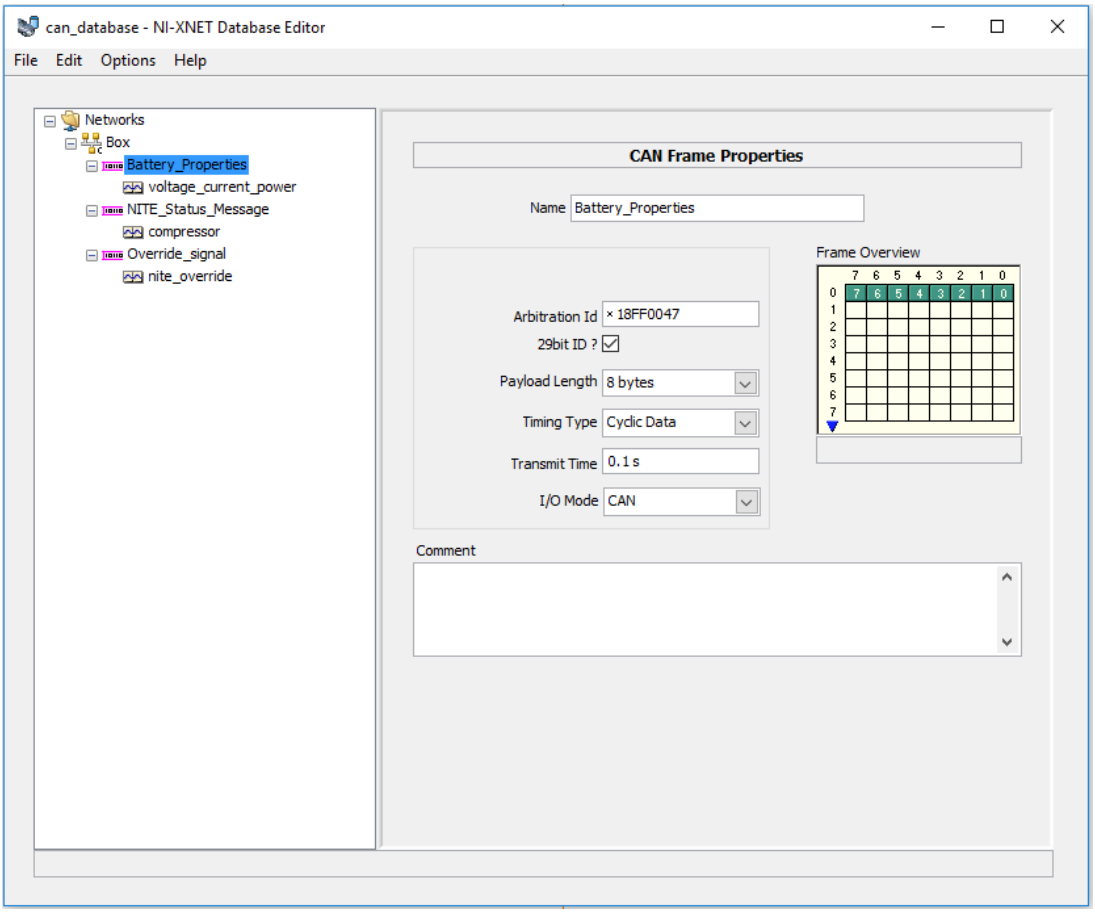

Figure B.3 Defining the general frame properties of the NITE's battery parameters message using data from Bergstrom.

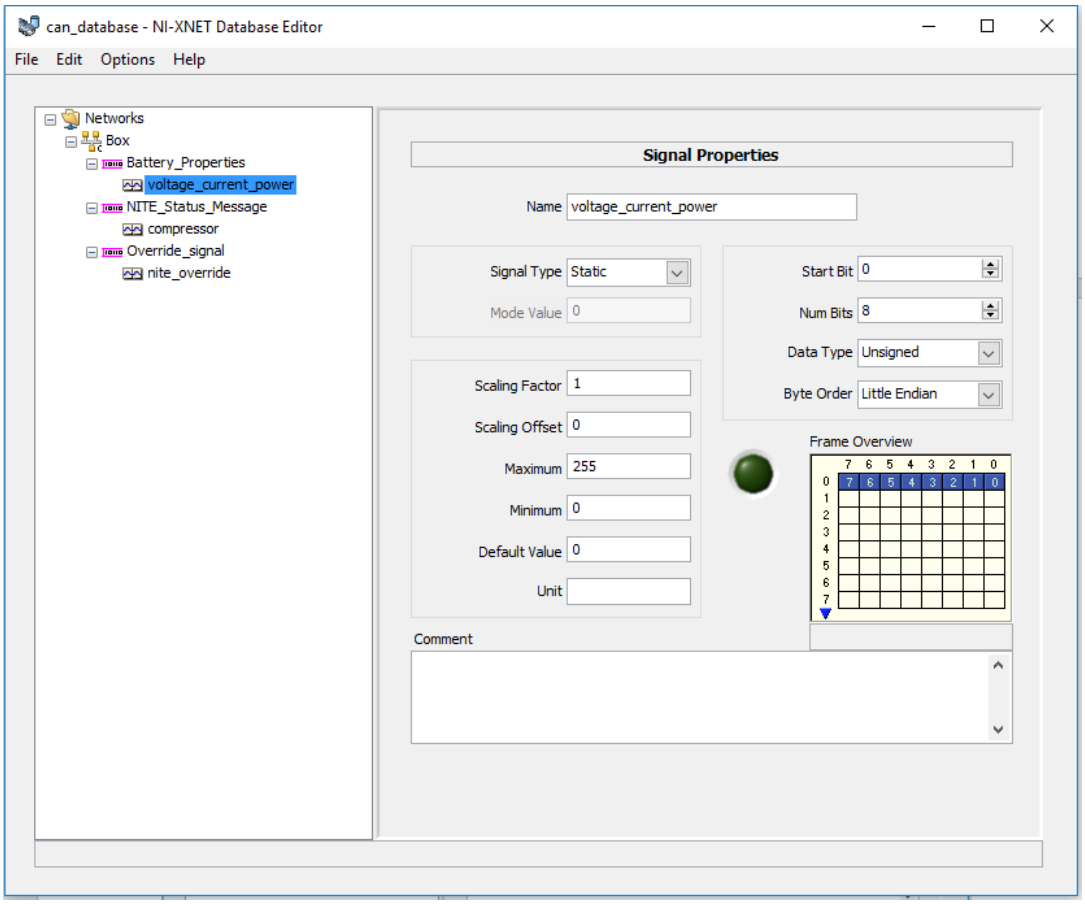

Figure B.4 Defining the CAN signal's specific properties.

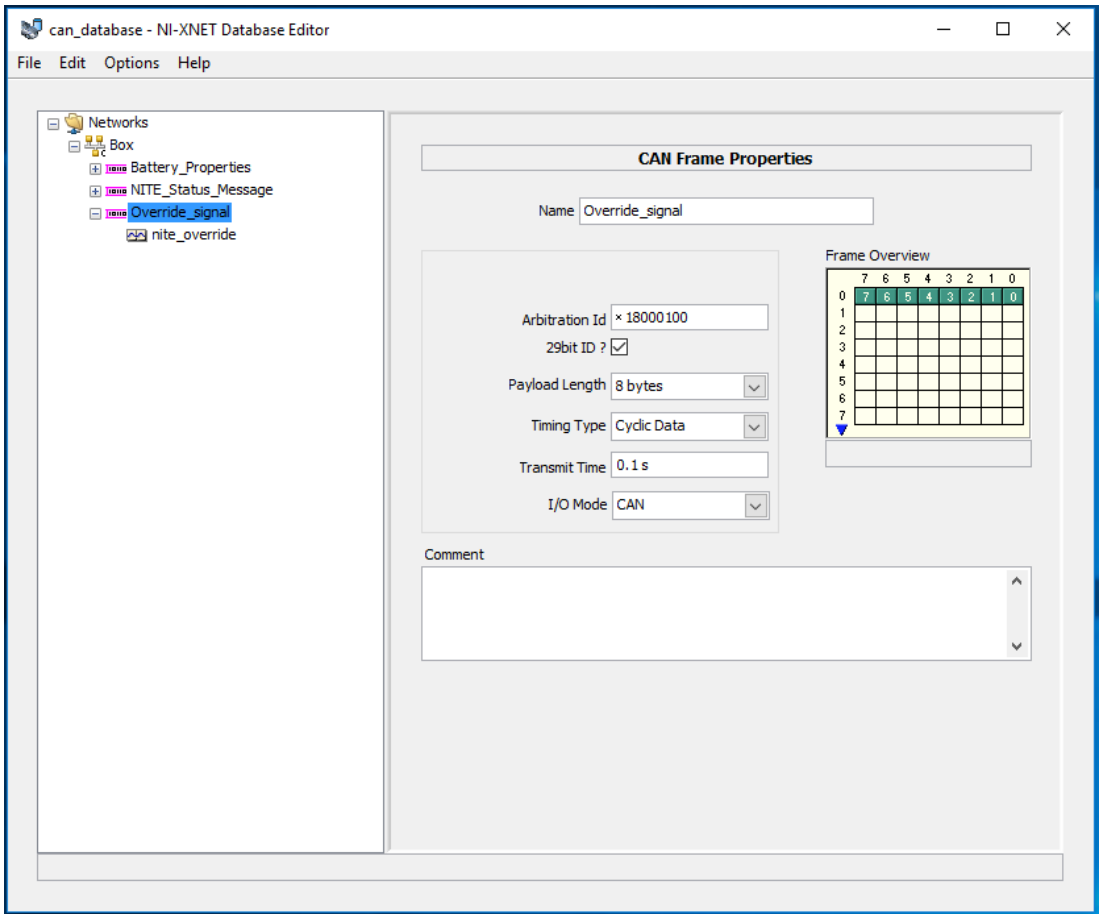

Figure B.5 Defining the frame properties of the NITE override signal using data from Bergstrom.

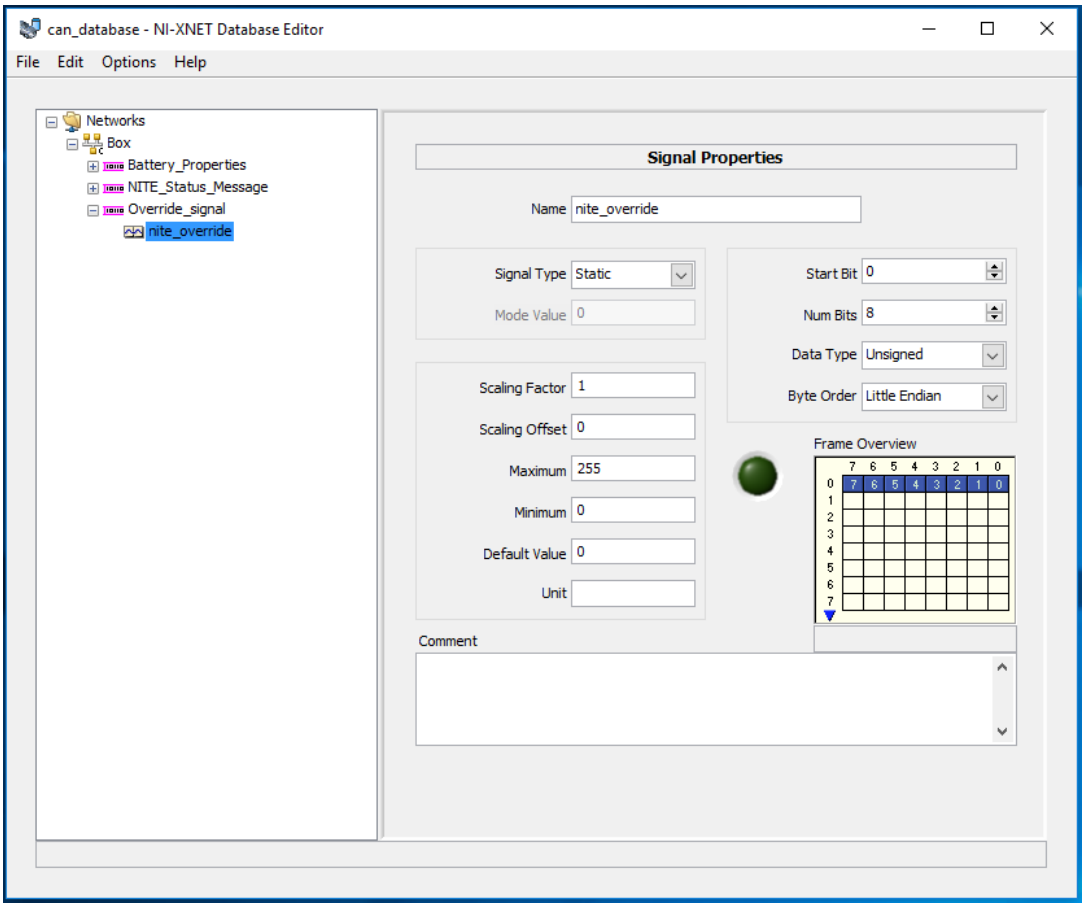

Figure B.6 Defining the CAN signal's specific properties.

Having initialized the CAN reading/writing session, the next step is to connect the output of the Create Session VI to the corresponding CAN Write or Read VI, which is also found in the NI-XNET library in LabVIEW. To read data, use a for loop along with an unbundle by name function to unpack the data from the output of the CAN read VI. To write data, bundle all desired data using the bundle by name function and route the signal to the input of the CAN write VI. An example of reading and writing CAN frames can be seen in Fig. B.7 below.

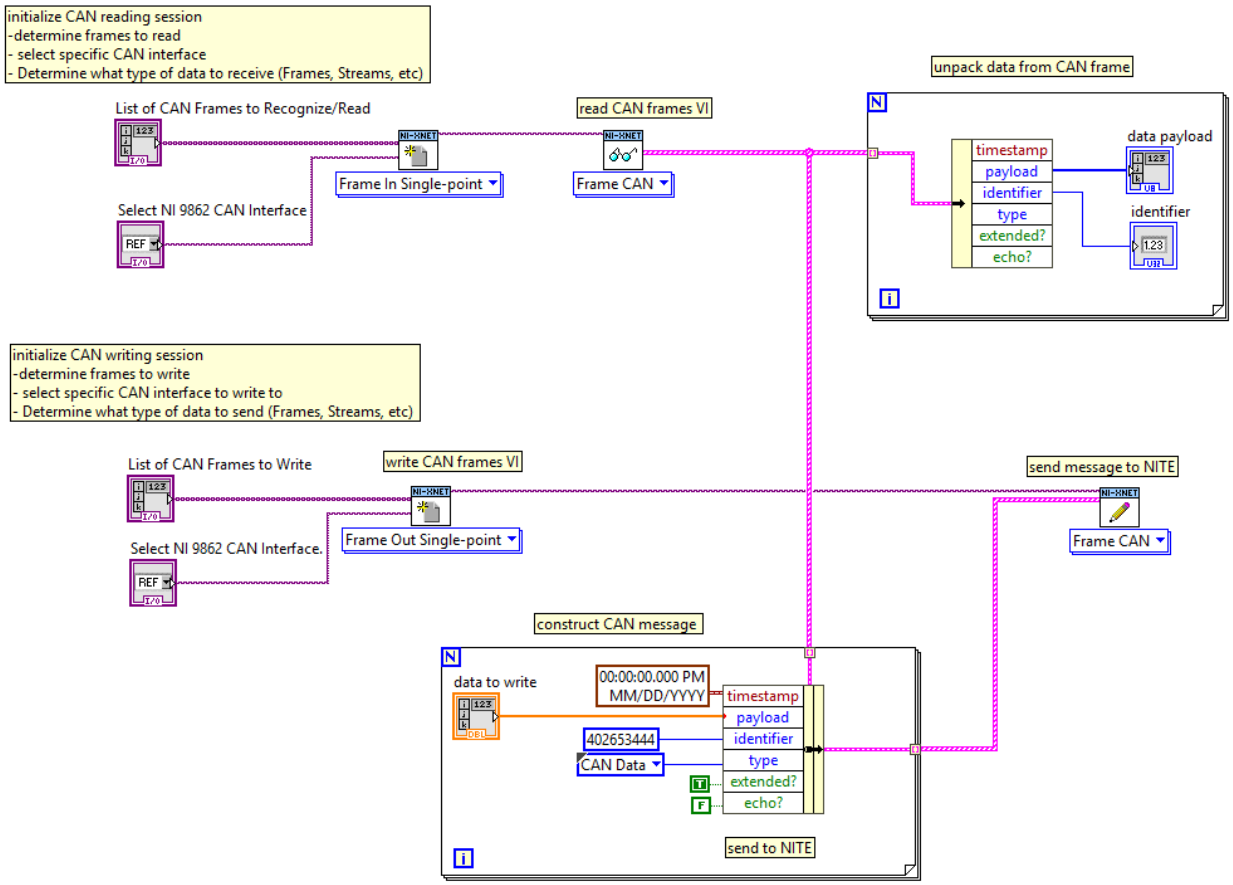

Figure B.7 Examples of reading and writing CAN messages sent from and to the NITE system respectively.

B.2 LabVIEW Code

This section contains all of the LabVIEW code used to read sensor data, read and write CAN frames, and implement PI control and ESC.

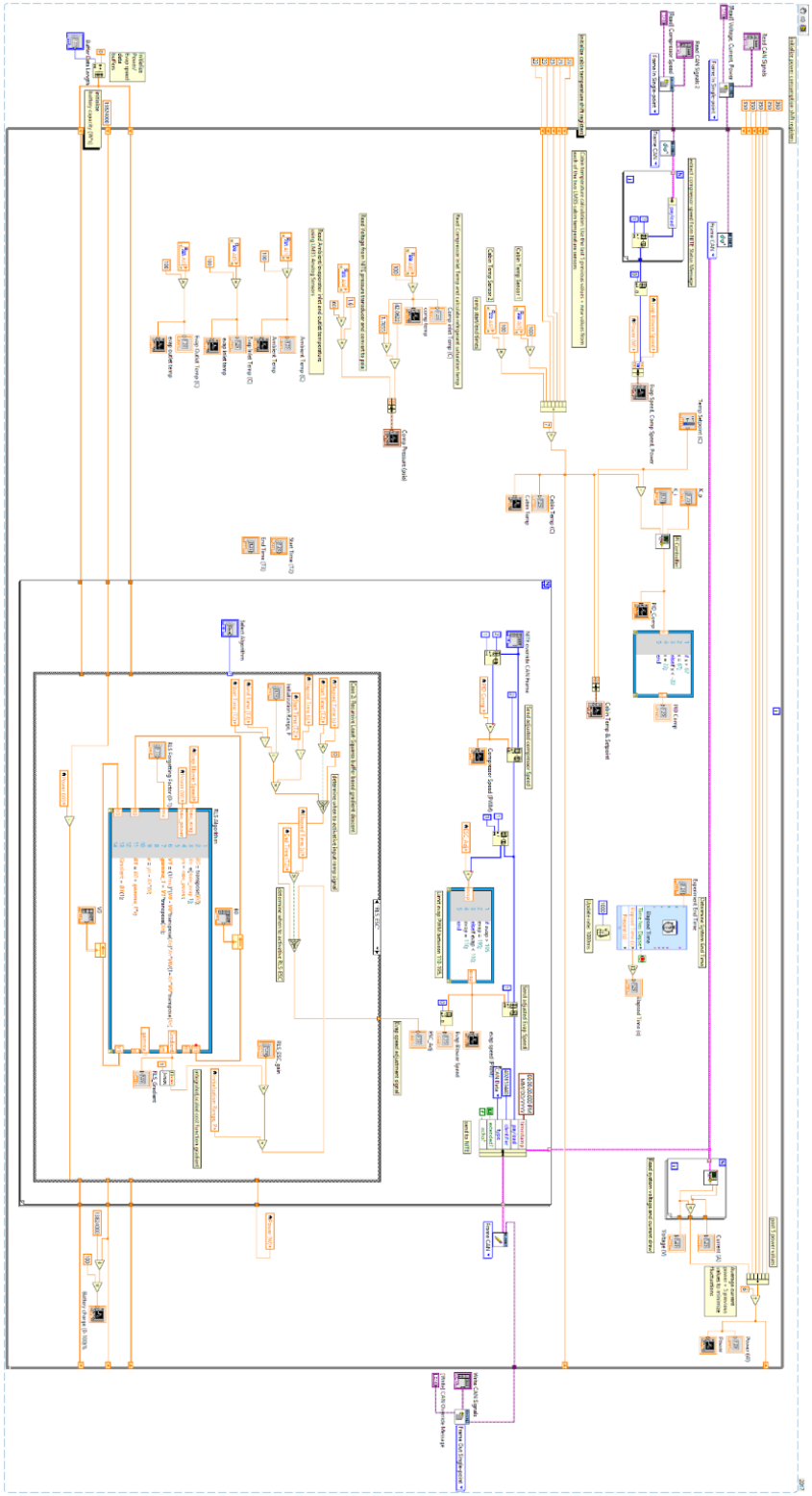

Figure B.8 A picture of the entire VI block diagram

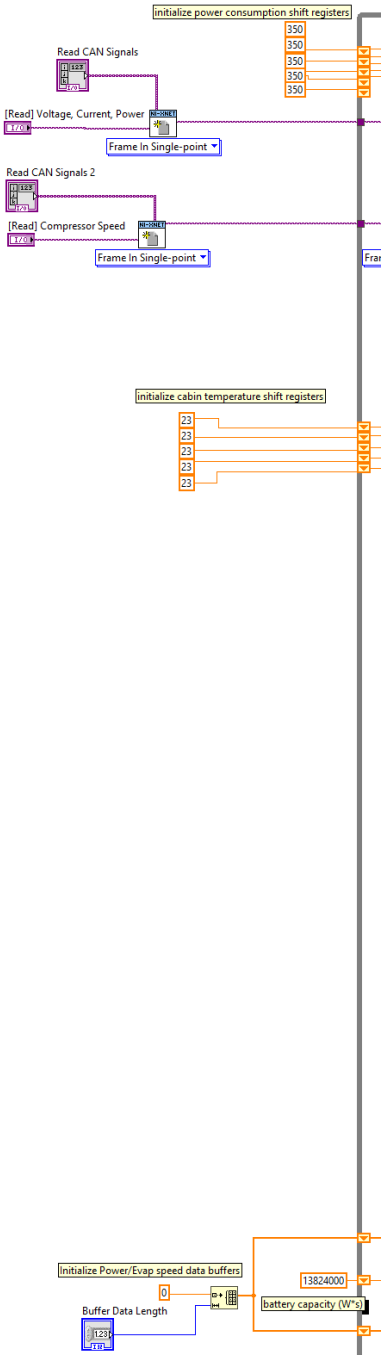

Figure B.9 Initialization of the VI block diagram.

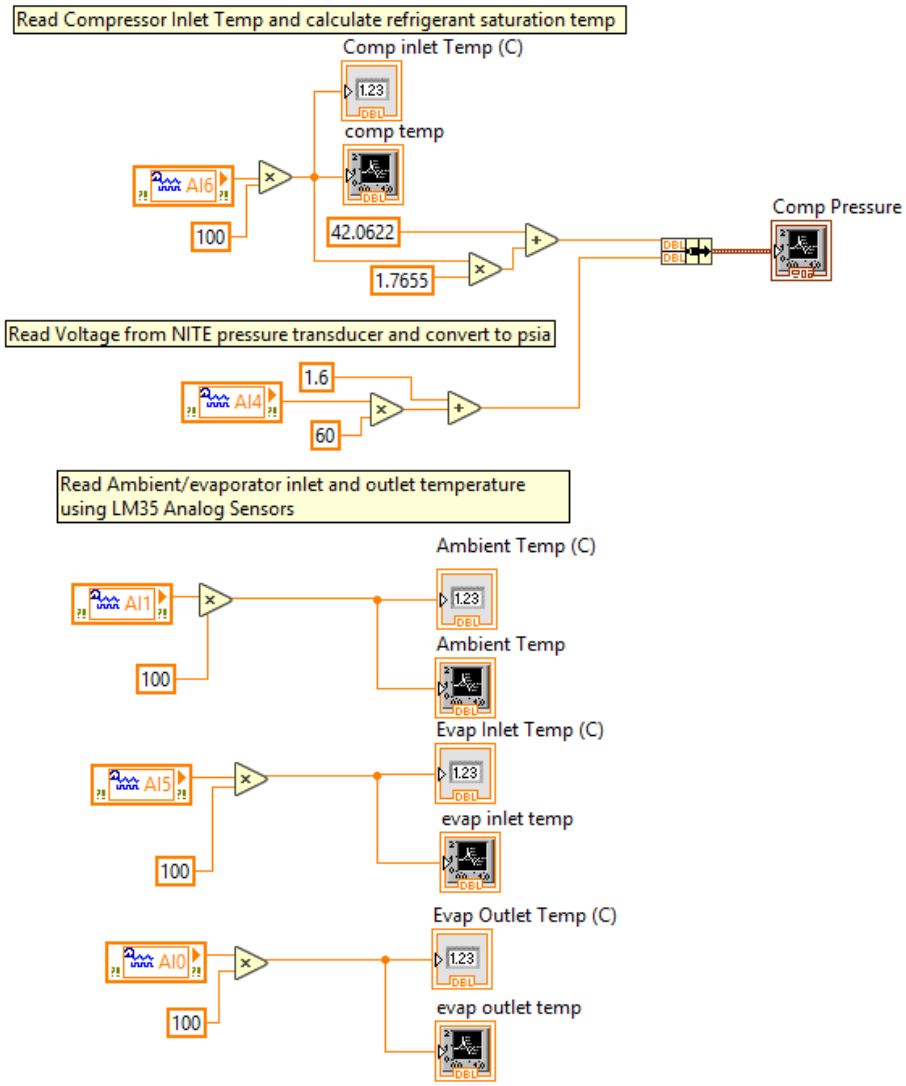

Figure B.10 Reading analog sensor data consisting of temperature and pressure readings.

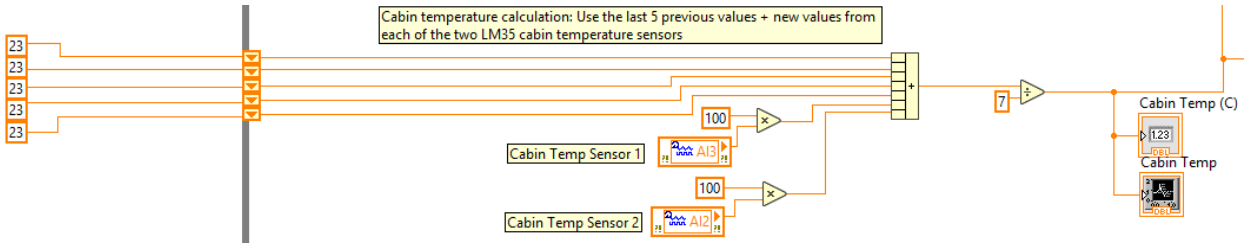

Figure B.11 Calculating cabin temperature by averaging previous readings to mitigate noise.

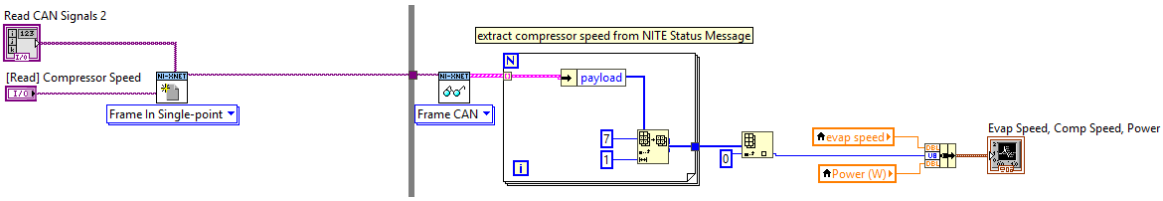

Figure B.12 Reading a CAN message containing the NITE compressor speed.

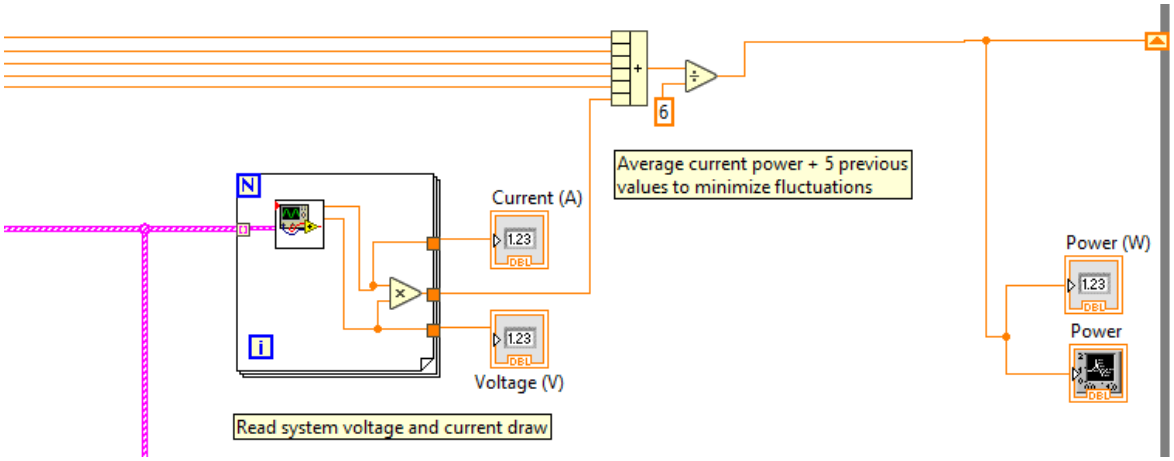

Figure B.13 Reading a CAN message containing the NITE current draw, voltage, and calculating the power. Power readings are averaged like with the cabin temperature to mitigate fluctuations.

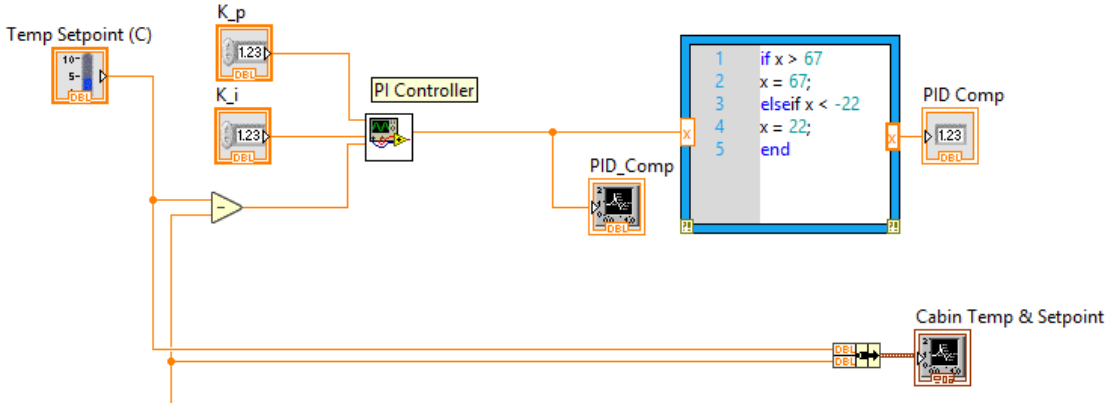

Figure B.14 Implementing PI control on the cabin temperature by modulating the compressor speed.

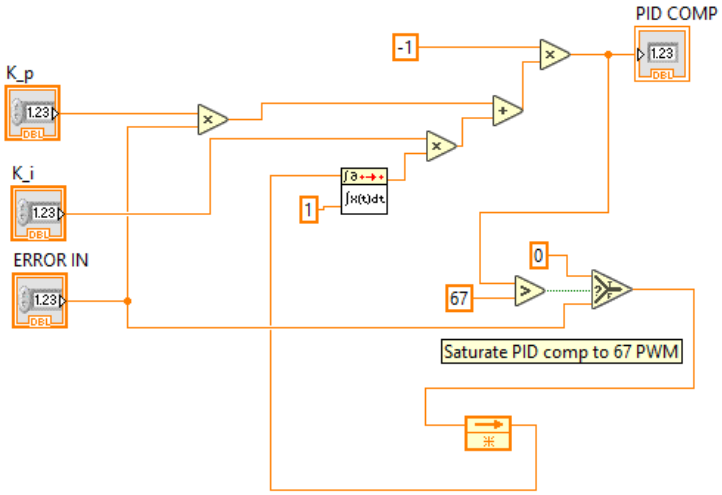

Figure B.15 Looking inside the PI control subVI.

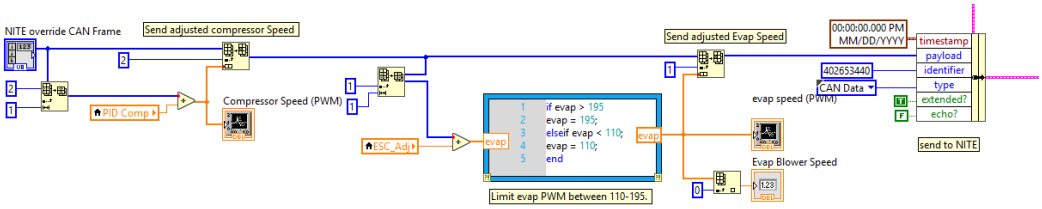

Figure B.16 Sending the override CAN signal to the NITE. The PI control determines the message's compressor speed, while ESC determines the message's evaporator blower speed.

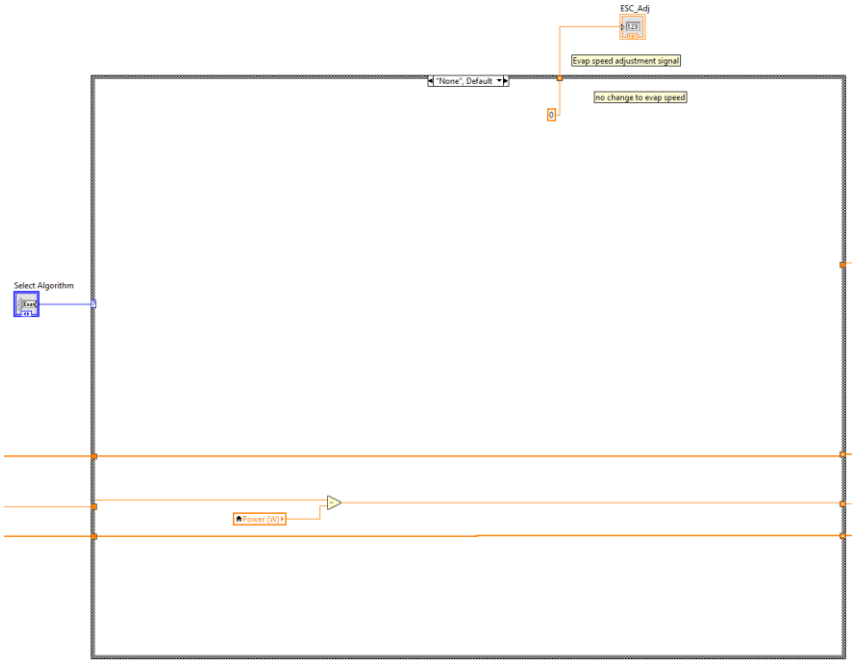

Figure B.17 The ESC case structure. We can select between four different cases: None, P-ESC, LS-ESC and RLS-ESC. Here, we choose None where the blower speed is unchanged.

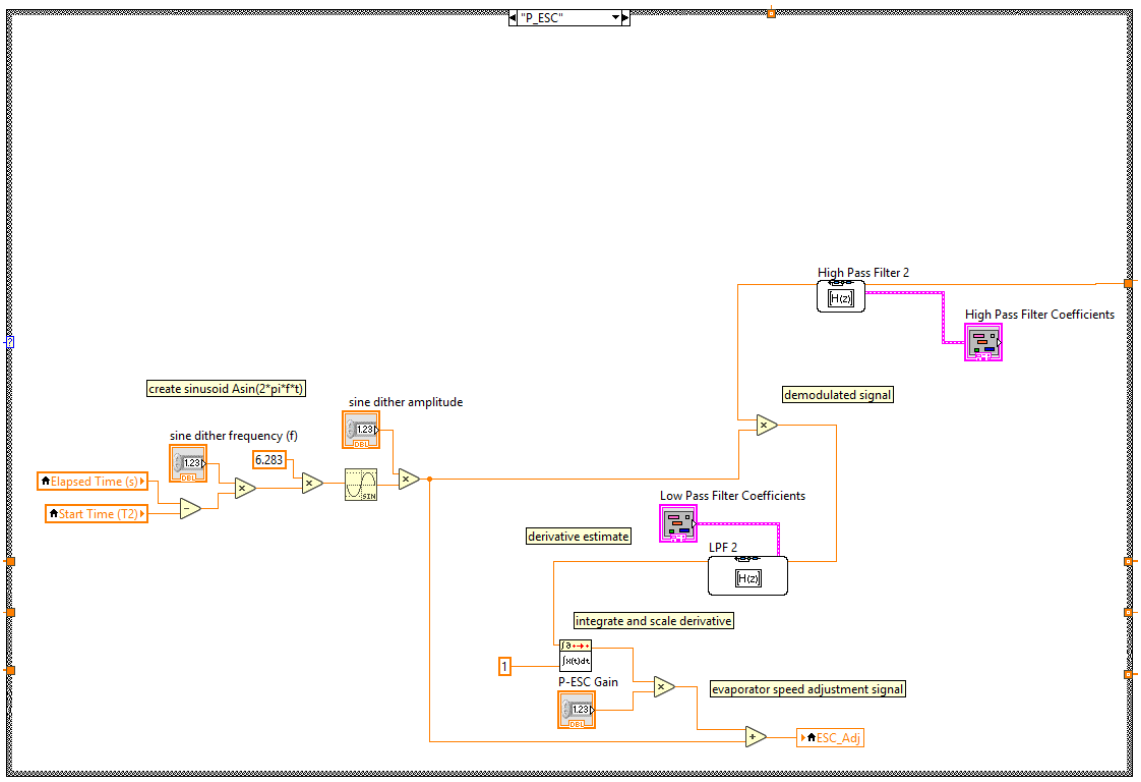

Figure B.18 The P-ESC algorithm case.

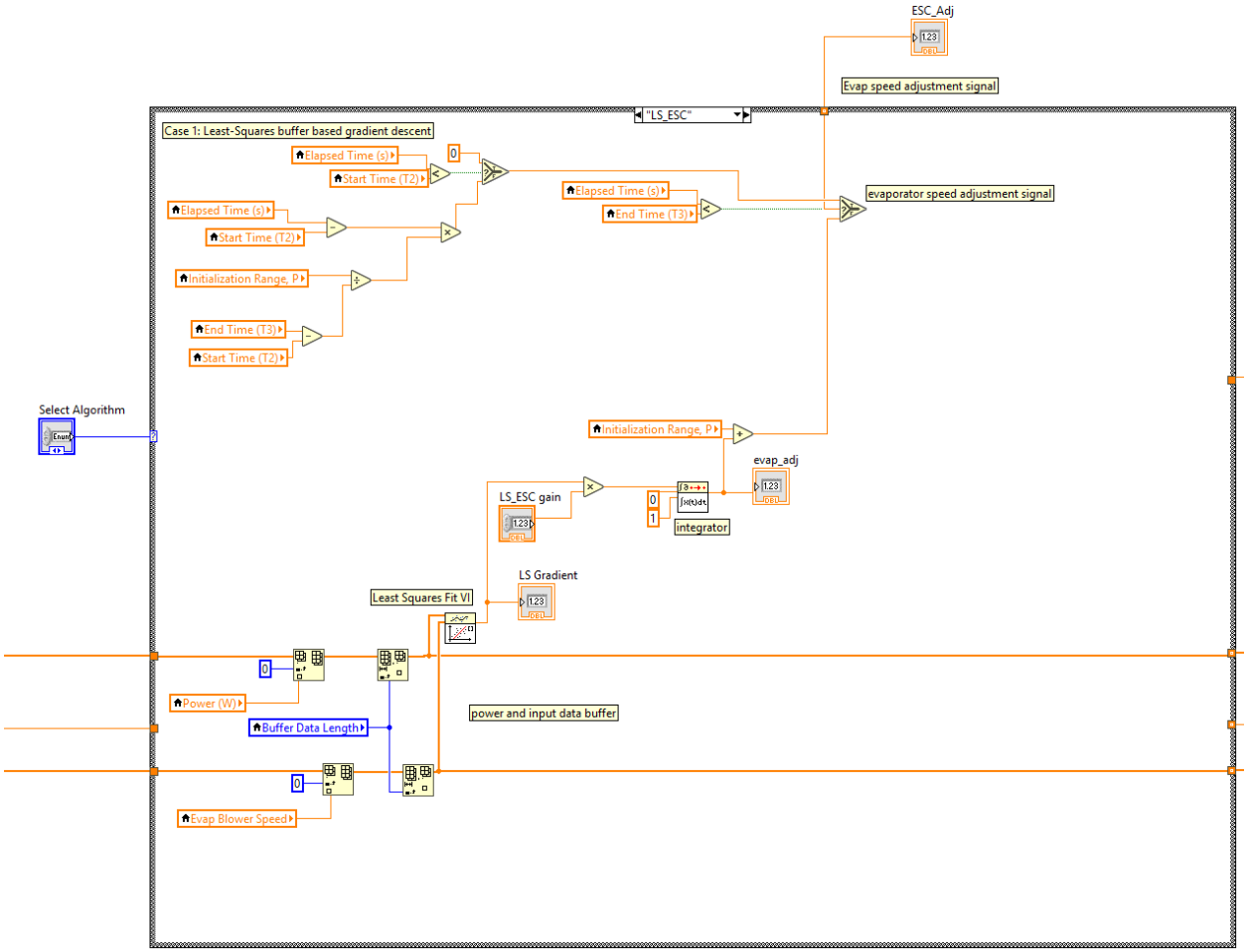

Figure B.19 The LS-ESC algorithm case.

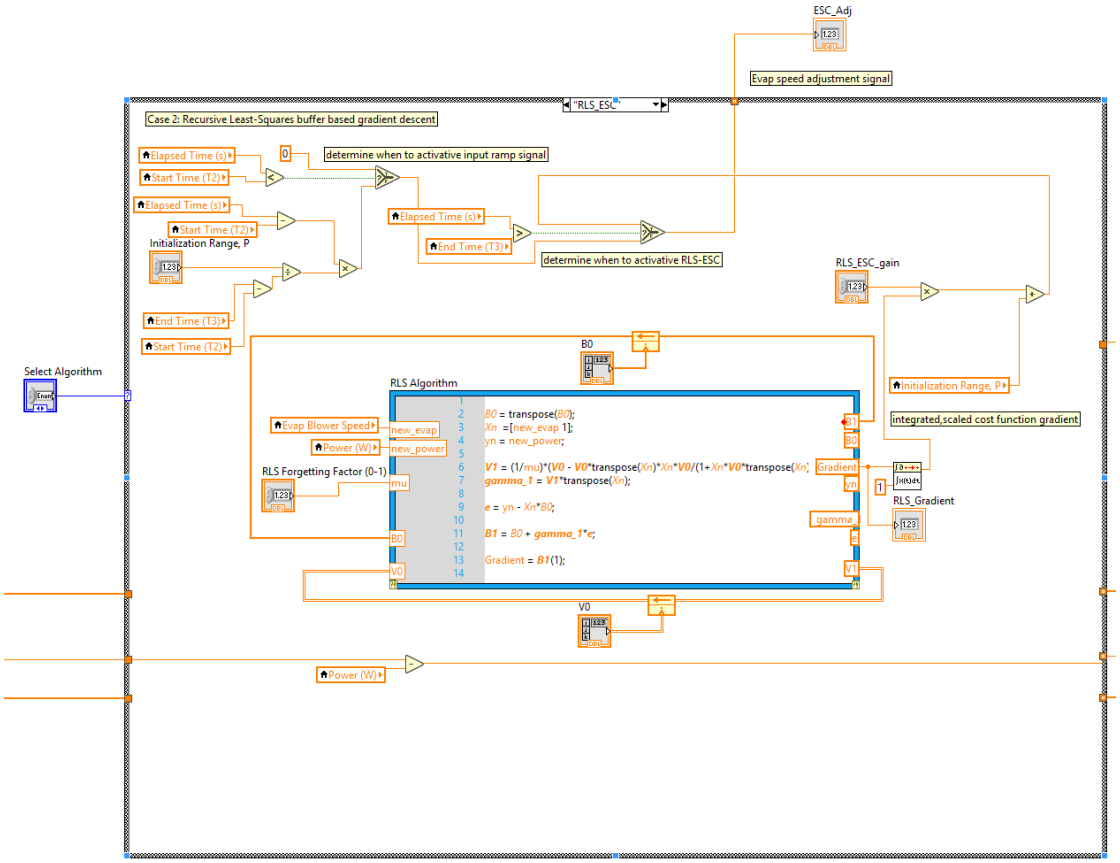

Figure B.20 The RLS-ESC algorithm case.