THE UNIVERSITY OF BRITISH COLUMBIA

School of Engineering
Okanagan Campus

Capstone Final Design Report: Development and Optimisation of an Industry 4.0 Framework for Composites Manufacturing

Faculty Advisor: Dr. Abbas Milani
Clients: Bryn Crawford & Reza Sourki

Group 34 Authors:
Graeme Paul
Daara Abbassi-Mohadjel
Darryl Lam
Louie Federico
Connor Gaudreau

Date of Submission: April 12, 2020

Contents

List of Figures

Executive Summary

At UBC Okanagan's School of Engineering, fourth year engineering students are required to complete the two semester Capstone design project. Over these two semesters, students of all engineering disciplines work in teams to solve a specific design problem for a client. For this project, our client was the UBCO Composites Laboratory with the project supervisor Bryn Crawford. This report's purpose is to detail the project's final design and the results achieved over the 2019/2020 semesters. The final report contains the project's needs and constraints, problem specification, design process, solution generation, final design details, and the final design's evaluation. The project's fundamental design problem is finding a solution for improving the manufacturing process of composite materials by specifically optimizing the curing cycle of composite materials. The goal was to replace the controller for the composite curing oven which is a standard PID controller with a smart solution implementing Industry 4.0 concepts. Industry 4.0 is the new paradigm digitizing and automating the industrial sector by using digitally connected sensors, machine learning, and real-time data usage. To complete this task, the needs were catalogued. The main needs were established to be finding the soak/cure time through the usage of machine learning and implementing a data collection unit to attain data. Based on this need of finding cure time, a solution was generated and selected by doing an extensive literature review, analysing the key findings, and creating a feasible solution.

Two machine learning implementations were selected for the preliminary design. The first was random forest which uses decision trees to make predictions on data. The second was long short term memory (LSTM) which is based on a recurrent neural network and is a more traditional neural network machine learning algorithm. To gather data, a Raspberry Pi and ADC were used together to read NTC thermistors inside of the heating chamber. A modified part which allowed for a thermistor to be inserted provided truth data which was used to train the neural network. Air temperature data inside the heat chamber over a cure cycle was also collected which became the input data to be used to make a prediction with the machine learning algorithms. In addition to real data, simulations were used to develop additional data to help iterate and tune our designs. After turning the machine learning algorithms and setting the necessary parameters, a prediction capability with a percent error of under 4% for the entire run was found. Due to the COVID-19 virus suspending laboratory times, we encountered further constraints that limited testing and the amount of data that could be collected in the lab. Code was then created to estimate the soak time, make future predictions on data, and return control decisions based on machine learning predictions. It was concluded that our implementation provided a good estimate of the internal temperature of the part . Finally, our recommendations for future topics which could be researched to improve or expand the project include additional validation and testing, implementation with a cloud service for data storage, data assimilation, transfer learning, and improved simulations.

1 Introduction

The UBCO Composites and Optimization Lab is interested in modernizing the production and fabrication of fibrous and polymeric composites, known as composites manufacturing. The modernization of these processes using the technologies developed over the past two decades is encompassed by the manufacturing paradigm of Industry 4.0. By focusing on the specific process of composite curing, a framework for applying Industry 4.0 principles to composite manufacturing processes can be developed.

Composite curing is the process of heating a composite material until it reaches a specific internal temperature, in which it undergoes a chemical strengthening process. This temperature will be reached at a specific time, the soak time, which is determined, but not easily calculable, by the initial conditions and physical parameters of the material. This entails that an object heated for its soak time will be cured. For a manufacturing process, the materials are heated in a curing oven, which on a large scale can be expensive to run. In a lab environment, where the soak time of a given material may not be known, curing ovens are left to run much longer than what could reasonably be deemed necessary to compensate for this uncertainty. To minimize the expense, and thus the time the oven is left running for, a new method to determine the soak time for a given material needs to be found.

Industry 4.0 is a modern paradigm in manufacturing processes whose principles include inter-connectivity, information transparency and decentralized decisions. In practice this amounts to using internet-connected technology to make data-driven decisions for the manufacturing process. With regards to composite manufacturing, the technological framework to accomplish this would include digitally-connected sensors, a centralized database, machine learning, and microcontrollers. For the specific process of composite curing, the sensors would detect temperature and relay this information back to the processing unit, which could then control the curing oven accordingly. By establishing a solution under Industry 4.0 principles, a transferable workflow can be established forming a basis for future research.

The objective of our project is then to apply the principles of Industry 4.0 to optimize the composite manufacturing process of composite curing. This report will detail the problem formulation, design process, the end product and an evaluation of the overall success of the project.

2 Problem Specification

The main objective of our project was creating an Industry 4.0 framework using a machine learning model to optimize the curing process by accurately estimating the soak time required. Specifically for our case, as a proof of concept we acquired an aluminum cylinder with a radius of 3cm and length of 5cm. The purpose of this proof of concept was to address the problems with the current solution in the UBC Okanagan Composites lab. It was expected that the new framework would implement Industry 4.0 principles integrating data acquisition devices with machine learning. Our clients also expressed a desire for the framework to have the sensor data and the machine learning model uploaded online to the IBM Cloud service. This desire was however dependent on time and costs because the IBM Cloud adds costs based on data usage. Therefore it was clearly stated that the amount of data should be limited as much as possible. However, building a robust machine learning model often requires large amounts of data for training and testing. For addressing the issue, we did not specify a maximum numerical value for the amount of data collected, instead we looked to reduce the amount of data used for training and testing the model until its performance suffered significantly. Using this strategy would be key to producing a framework to satisfy the problem.

The temperature range to be used for curing parts was specified by the client early in the project. The allowable range was between 25°C and 80°C for curing parts. Therefore any material placed inside the heat chamber had to be capable of withstanding temperatures of at least 80°C. However, our clients did have aspirations for using the final product to test composites at temperatures over 200°C. Therefore the specification for our final solution included handling temperatures up to 250°C which was the maximum temperature the thermistors could withstand. As the project budget was limited to $300, it was necessary to minimize the cost of hardware devices used for our final design. To stay within budget, we limited our costs on hardware down to devices which included a 16 bit ADC for data acquisition, and a Raspberry Pi for storing the data from the ADC and extracting it for analysis. Ultimately by limiting the costs of hardware devices, we were able to stay within the $300 budget.

3 Needs and Constraints Identification

3.1 Stakeholders

The official partner involved in this project is The Composites Research Network (CRN) Okanagan node. The CRN is a group of several Canadian universities and industry partners involved in composites manufacturing. The purpose of the CRN is having academic institutions and industry leaders collaborate to find solutions to problems facing the composites manufacturing industry [1]. The resulting intellectual property produced from this project could impact the members of the CRN outside of the Okanagan node, so they're considered external stakeholders even though they were not consulted during this project.

The main stakeholders for this project are Project Supervisor Bryn Crawford, Facility Supervisor Dr. Abbas Milani, and Ph.D. student Reza Sourki at UBC Okanagan campus. These main stakeholders work at UBCO's Composites and Optimization Laboratory doing research and development on polymeric composite materials. The Project Supervisor Bryn Crawford and Reza Sourki were met with throughout both semesters. Meetings consisted mostly of conceptualizing the problem and understanding their expectations in the first semester. Once the problem was defined, meetings transitioned towards providing updates and progress reports.

The main internal stakeholders at the UBC Okanagan Composites and Optimization Laboratory were impacted positively by the results of this project. Their needs for minimizing soak time and accurately predicting the curing cycle of a composite material were met to an acceptable level. The intellectual property created can also impact other laboratories in the CRN doing composites research and development. There could be benefits for these labs if our framework is adjusted to work for predicting the curing cycle on more complex composite materials.

3.2 Needs

Through early discussions with the stakeholders and group meetings, the needs for the project were established. It was found that the main need was to accurately predict a composite's internal temperature and minimize the soak time of the curing cycle. Also made clear by the stakeholders was the need to implement an Industry 4.0 framework creating an innovative and robust final design.

The stakeholders expressed an additional need for implementing an Industry 4.0 framework, combining sensors with machine learning algorithms for estimating soak times. It was determined that using machine learning algorithms would provide a more robust solution. This meant the framework had to include a way for users to easily store and manipulate data. Finally, the idea of designing a process for uploading data to the IBM Cloud service was expressed if it was feasible. This would give the stakeholders a way of accessing data across multiple devices and provide a platform for doing data calculations. Due to the cost of storing extensive amounts of data, it was defined that our implementation should try to limit the amount of data as to minimize the burden of costs on the lab. It was determined that meeting these needs defined by the project stakeholders would satisfy the requirement of producing an Industry 4.0 framework for this project.

Accurately finding a given part's internal temperature over time for the purpose of reducing soak time was considered the most important need for this project. The solution had to satisfy the main need using Industry 4.0 principles including data collection, data streaming, machine learning, and cloud services. Lastly, the overall design framework required detailed documentation so it could be easily replicated over a range of composites.

3.3 Scope and Constraints

Many constraints were identified during the definition stage of this project. The scope of the primary deliverable was building a framework for obtaining internal part temperature using a machine learning algorithm for simple metal cylinders and using this data to estimate the soak time. Studying objects with different materials, sizes, and shapes were considered within the project's scope if the primary deliverable was completed before the project's deadline. Metal cylinders were proposed by the stakeholders as the limit of our scope for testing and implementing our Industry 4.0 framework because the heat transfer characteristics are easily known whereas for composite parts the characteristics are not always available. So estimating soak times on composites with epoxy exhibit exothermic properties was considered outside the scope of this project in order to simplify the process for developing a framework.

With a budget of $300, the primary cost constraints were first identified for the project. These included temperature-based sensors, data collection devices like Arduinos and Raspberry Pi, and costs of acquiring metal parts. NTC 100k Thermocouples were implemented for sensing air temperature and part temperature in the heat chamber. Also a 16 bit analog-to-digital converter (ADC) microcontroller and Raspberry Pi were used for creating the data acquisition system for storing and streaming real-time data from the curing cycles. It was planned to store data on the IBM Cloud computing service keeping in mind the volume of data would lead to higher monthly costs for storing this data thus data must be minimized.

It was found that the accuracy of the thermocouples was critical because the data collected would be used for training and testing our machine learning algorithms to estimate a part's internal temperature and soak time required. Therefore constraints on the thermocouples and all hardware devices had to be considered such as the latency and signal-to-noise ratio. To satisfy these constraints, it was important to ensure that the measurements obtained by the hardware devices in this project were consistent and repeatable under specific testing conditions.

The amount of tests performed was also a recognized constraint as it placed restrictions on the volume of data collected and time availability. An objective was to determine the amount of tests necessary for obtaining a large enough data set to meet our expectations. It was found that the time consumed for a single test depended on the target temperature, thermal properties, and the size of the metal cylinder. Due to these constraints we selected an aluminum cylinder with a 3cm radius and cured it at lower temperatures than typically used for composite curing to have more test runs and a larger data set. Another strategy for meeting these time constraints related to test runs was implementing MATLAB heat simulations to generate more data sets for training our machine learning algorithms and testing them.

There were some risks identified during this project with regards to health and safety. There were potential dangers working with the heat chamber and infrared lamp because of the high temperatures involved. It was necessary to practice the safety precautions laid out by our clients when using the lab equipment. There was minimal technical risk of losing data due to the use of cloud storage and revision control. Finally there were no significant environmental or societal impacts to be considered within the project's scope.

4 Solution Generation and Selection

4.1 Previous Solutions by Stakeholders

To define the problem with the current solution, an assessment of the client's current solution was done. The lab was using a heat chamber with an infrared (IR) lamp as the heating source for curing composites. Inside the heat chamber, three thermocouples are used to measure the air temperature. Due to variability in temperature inside the chamber and variability between the sensors themselves, an average of all the thermistors are used as the temperature which the Ardiuno acts on. This average value gives an accurate enough reading of the air temperature recorded over the entire curing run. The chamber uses a PID controller which controls the IR lamp by sending a PWM signal to a MOSFET. The heat chamber's temperature rises a few degrees per minute until the desired curing temperature is reached. As specified earlier, the heat transfer characteristics of a composite part are not easily determined, and therefore the soak time cannot be analytically determined. The method currently used is to hold a part at the target temperature for a period of time determined heuristically. This time is long to ensure the part temperature reaches the threshold and is cured completely. This is where the current solution needed improvement because the minimum soak time required for the part to reach equilibrium temperature was being exceeded. Because of this, the heat chamber was being used longer than necessary which led to wasted time and power.

4.2 Industry Solutions

When initially formulating solutions to the problem, a literature review was done to assess current industry solutions for accurately predicting part temperature and estimating soak time. This section reviews current industry solutions for optimizing composite curing and evaluates their potential application to this project.

It was found the key parameters involved in the cure cycle are the heat transfer characteristics, material properties, heat convection through the material, and surrounding air temperature. In the composites manufacturing industry, using control strategies and numerical simulations is a technique for developing an optimal cure cycle. The numerical simulations involve adjusting parameters previously mentioned to build prediction models for improving the degree of cure and minimizing the total time of cure. By defining boundary conditions inside the heating chamber, finding an energy equation combined with a kinetic model, and using finite element software, the degree of cure and temperature across a composite can be simulated [2]. Here the numerical simulations showed the degree of cure and temperature at various points on the composite with different thicknesses. The paper however did not have experimental data of an actual cure cycle to compare with the simulation results. This process can be validated as seen in [3] which used similar numerical simulations and had results showing accurate estimates of part temperature and better degree of cure at various points on the composite. The solutions proposed in these studies struggle to address the problem of producing a non-uniform temperature distribution across composite materials. This causes some spots on the composite to be burnt and others processed below the desired temperature. Also, the models still cannot always accurately predict the soak time.

Another possible solution was a self-tuning method used for finding the optimal curing parameters using a programmable microcontroller to create a self-tuning algorithm [4]. First, the oven parameters were used to create an electrical model and then derive a differential equation which could be used for finding an optimal cure cycle. A PID controller was implemented to adjust the oven temperature based on the differential equation and electrical model. Finally, simulation results were compared with experimental results to prove the self-tuning method's accuracy. The results of this study reveal similar problems found with our clients curing process, such as the inability to fully predict when the part is within the desired curing temperature. The solution also does not provide the level of innovation we hoped to achieve in this project.

4.3 Machine Learning Applications in Composite Curing

For this project it was desirable to implement an Industry 4.0 design as a solution, and specifically an Artificial Neural Network (ANN). The motivation behind this decision was supported by evidence from literature on applying an ANN to composites curing. It was also found that the problems associated with the other current solutions discussed earlier can be mitigated using an ANN, from sources such as 'Prediction and optimization of cure cycle of thick fiber-reinforced composite parts using dynamic artificial neural networks' [5]. This study implemented a neural network with the purpose of optimizing cure cycles using the surrounding air temperature and the characteristics of the composite. This goal is achieved by first building a finite element model (FEM) on the composite part. The FEM provided sample data of the variables time, autoclave temperature, and composite part temperature at specific points. Then this data was used for both training and testing two different neural networks. One network was used to model the output and control of the composite's temperature and the other for the composite's degree of cure. Doing this allowed them to see relationships between a part's temperature and the degree of cure at various points on the part. The study then looked at several different training algorithms and found Bayesian Regularization had the strongest prediction ability. Also, it was found that making changes to the ANN architecture like changing the number of neurons had no significant changes to the predicted output. Overall, the ANN applied in showed results that reduced soak time and accurately monitored a given part's internal temperature.

Then in 'Optimization of the Temperature-Time Curve for the Curing Process of Thermoset Matrix Composites' [6] it showed a recurrent neural network (RNN) which is a specific type of ANN. This was applied towards optimizing cure cycles and producing a uniform cure on fiber-reinforced thermoset matrix composites. The RNN used two hidden layers between the input and output layers which allowed for increased prediction ability without adding significant amounts of time to train the network. This structure also allowed the number of neurons to be decreased because the additional hidden layer's outputs are fed back into the network. As before, there were two RNN's used, one for predicting the composite's temperature and one for the degree of cure. The results produced by the RNN's showed the optimal curing times for a composite which achieved a uniform degree of cure across the material. These results are relevant to our project because having a solution that achieves a uniform cure across a material is important to the client. These papers use epoxy parts with complex properties which are outside the scope of this project. However, the concept of training an RNN with data generated from a FEM is still useful for this project.

4.4 Project Workflow

To facilitate the solution generation process, a project workflow was developed. While this project was focused on designing for a specific task, the overall goal was to develop a generic workflow for applying Industry 4.0 principles to composite manufacturing processes. The workflow was generated at the start of the concept generation part of our design, allowing other solutions to fit in with each of these points. The abstracted workflow for the project was developed as follows:

1. Identify all input and output parameters of process
2. Use Industry 4.0 technology to automate data collection of input and output parameters
3. Process input and output data into supervised learning data sets, in which each set of inputs is mapped to one output
4. Determine models most applicable to current problem and eliminate others
5. Design machine learning models to take input parameters and produce output parameters
6. Train ML models using collected supervised learning data set
7. Determine models that are working well with data set and eliminate others
8. Refine model parameters to enhance predictive accuracy
9. Validate model with test cases, including fringe examples
10. Integrate machine learning module into data collection network

4.5 Machine Learning Model Selection

Machine learning algorithms can be categorized into four main types: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. In supervised learning, the algorithm takes training data and generates a function made up of biases and weights that relates the input features to the output labels of the data sample. The goal of supervised learning is to train the algorithm on known data in order to later predict on unknown data. In unsupervised learning, the algorithm does not have access to the output labels. In this case, the goal of the algorithm is to model the structure or distribution of the training data in order to later predict on unknown data. Semi-supervised learning can be considered part supervised learning and part unsupervised learning. In this case, the data contains some output labels, but output labels are not present in all of the data. In reinforcement learning, like unsupervised learning there are no output labels to train the algorithm. Rather than trying to model the structure or distribution of the data, reinforcement learning uses an agent that learns through experience and tries to maximize reward which is defined by a reward function.

Additionally in machine learning, there are two main categories of tasks: regression and classification. In regression tasks, the output label being predicted is numerical while in classification tasks, the output label is a category. In our case, the problem of finding the internal temperature of an object is a regression task.

Supervised learning was selected as the most suitable machine learning domain for our use case, as it is difficult to determine the optimal machine learning algorithm for an application without testing and comparing the results. Selection of models was also aided by the structure of inputs and outputs. Machine learning models that work well with time series data and output a real number should be prioritized.

4.5.1 Machine Learning Development Platform Selection

As part of our concept generation we had to decide how to implement the machine learning models described. There are many possible languages and packages which can implement these algorithms. Two languages which our group was both familiar with and had well-documented libraries were Python and MATLAB. Python was chosen due to the packages available, flexibility, experience of our group members, and preference.

Keras and Tensorflow

Tensorflow is an open-source library for numerical computation and large-scale machine learning. Keras is used as a high-level neural network API for Tensorflow. Keras was chosen for this project due to its accessibility in implementing machine learning algorithms using Tensorflow. With Keras, we were able to implement the various RNN model types, such as LSTM and GRU, using Python. Using these tools, we were able to train and run the RNNs from the data gathered to accomplish our project needs.

Scikit-Learn

Scikit-Learn is a machine learning library for Python. Scikit-Learn was used for this project due to its simplicity in implementing the Random Forests machine learning algorithm. It also enabled us to remain consistent with using Python for all our implementations of the machine learning methods.

4.6 Simulation Tools

MATLAB was chosen because of its built-in toolboxes and functions like the Partial Differential Equation (PDE) Toolbox™. This toolbox can be used for solving partial differential equations and FEM heat transfer problems [7]. The PDE toolbox allows users to create a thermal model which holds the geometry, boundary temperatures, initial conditions, and material properties of the thermal system and simulate the transient response. Furthermore, the entirety of the group was familiar with MATLAB. This provided a basis to get into the complex field of heat simulation while focusing on more pertinent problems.

5 Design Process

As the project was both modular and largely virtual, large scale prototypes were not developed. The two large components of the project, the data acquisition unit and the machine learning unit, were developed concurrently with their integration being overseen as the final step. As the data acquisition unit was simple in comparison there was no need found for a preliminary design. For the software components, the design was continually iterated upon and tweaked until desired results were obtained with all iteration tracked using the revision control software Git. Therefore the preliminary software implementation in our design slowly morphed into the final design through iterating upon it.

5.1 Design Process of Data Acquisition

The basic concept for data acquisition was to measure the air temperature of the curing oven as well as the internal temperature of an aluminum cylinder. The preexisting curing oven setup was used, being controlled by a PID controller being fed air temperature data using the average of four 100k NTC thermistors.

For the first design, air and internal temperature measurements were taken using five 100k NTC thermistors to ensure consistency with the original setup and to determine an accurate temperature profile for the immediate vicinity of the cylinder. The temperature data was read and logged by a Texas Instruments TM4C123G microcontroller, which was chosen due to its affordability. The cylinder was suspended by a wire cage, to maximize the thermal absorption of the cylinder.

While this setup produced data relatively consistent with what was expected, a number of issues were identified. The biggest issue was the variance in temperature readings between the thermistors. A 3 degree difference between them was found by exposing the thermistors to a constant heat source. As the machine learning model requires accurate data to provide accurate predictions, this posed a serious issue. Possible issues were determined to be faulty connections, an incorrect circuit configuration, a wrong thermal model, and the ADC on the microcontroller. Different thermistors produced the same variance, which implied that the issue laid in the ADC of the microcontroller most likely due to a faulty board.

Another issue came from a characterization of the temperature profile in the curing oven. While the temperature on the horizontal plane was constant, there was a steep temperature gradient vertically, which entailed that the inconsistency of the wire suspension setup would make it difficult to generate reproducible data. On the other hand, the constant temperature on the horizontal plane eliminated the need for many thermistors, as only 3, for the top, middle, and bottom of the object would be necessary to determine the air temperature. For the final design, a Raspberry Pi 4 microcontroller was used to read and log thermistor data, as its readings were found to be consistent when exposed to constant heat. As the position of the cylinder in the chamber was found to be a significantly larger influence on temperature than the existence of other thermally conducting masses, a wooden base was used to support the cylinder rather than the wire suspension. Studs were cut into the top to isolate the cylinder. The specifics of the final design are shown in Design Details section 6 of the report.

5.2 Simulation for Data Generation

Using machine learning algorithms to predict the soak time for a curing cycle could possibly require a large amount of data for training and testing. During the planning stages of this project, we determined that heat chamber testing may be insufficient for generating the required amount and different types of data. In addition, simulation of the process allows us to tweak and iterate on the design with different data and different amounts of data easily. Physical tests would add extra cost and time running the heat chamber, along with the difficulty of acquiring parts with different materials, various geometries and properties.

To acquire data more economically, we found that the best way to address this problem would be to implement finite element software to build a thermal model and simulate curing cycle data sets, a technique used in articles from the literature review [6]. Using the PDE Toolbox, we were able to model our aluminum cylinder and the heat chamber conditions then simulate the transient response for the system. Generating simulated curing cycles in MATLAB produces useful data sets with similar features to real curing cycle data. These simulations are important because providing a neural network or machine learning algorithm with data sets accurately representing real data sets will increase the robustness. The resulting simulations can effectively train and test a neural network, reducing the time and money needed for experimental testing needed for generating data sets. Fundamentally the simulated data allowed us to quickly iterate and tweak the machine learning system and check its validity in predicting under different circumstances.

5.3 Machine Learning Model Design Process

Through research and experimentation, we narrowed down the algorithms we were interested in and selected two final algorithms to implement and compare. The two supervised learning techniques that we focused on were Random Forests and RNN specifically Long Short-term Memory networks (LSTM). To better understand the applicability to our problem, we continued researching these two algorithms and implemented them into our preliminary design.

The fundamental design of these models are accessed through library's and packages mentioned in the Solution Generation in section 4.5.1. Both Keras/Tensorflow and Sci-kit learn were used to implement the bulk of the learning networks which make up the machine learning algorithms. This left us with having to design the machine learning parameters, implement the surrounding code to access the packages and libraries, and then output the performance of the machine learning processes. In addition to this, there was also the task of applying code to make predictions, and show the capability as a control source for the data acquisition unit. To quickly prototype the algorithm on larger data sets, the simulated data was used. The simulated data allowed us to quickly asses the feasibility of the machine learning algorithm and quickly iterate on the design to improve it. As mentioned above, iterating on software was simple with the help of revision control software. Once real data was obtained, the machine learning algorithms proved to be well designed because of the previous work done to validate our designs on the simulated data.

6 Final Design Details

The final design can be segmented into 2 stages: the data acquisition unit and the machine learning unit. Connecting these two parts of the project is the data refinement process. In a broad overview, the data generated by either the heating chamber or the MATLAB simulation were processed and input into the Random Forests and LSTM neural network. Each component of this is detailed below.

6.1 Data Acquisition

The data acquisition unit consists of the curing oven, the PID control system, the temperature sensors that provide the temperature data, and the microcontroller that reads and logs the data. The curing oven and PID control system remain unchanged from their initial setup.

6.1.1 Curing Oven Configuration

The curing oven configuration consisted of four 100k NTC thermistors for recording the external air temperature around a test object with an internal thermistor. The object used in the experiment was an aluminum cylinder of 3 cm radius, and 5 cm height. A hole was drilled into the cylinder with a depth of 2.5 cm in which we inserted a thermistor. The metal cylinder was held by a raised 10cm wooden platform with studded ends. This was to keep the cylinder isolated from other heat sinks. A mockup of the design is shown in Figure 1 with the heat source in red, the cylinder in white, and the thermistors connected to the black wires.

6.1.2 Temperature Sensor Configuration

The three external thermistors were placed close to the surface of the cylinder. They were positioned above, below, and on the side of the cylinder. It was found that due to constant temperature on the horizontal plane, any extra thermistors would unnecessarily increase the amount of data. While three thermistors were used, due to the vertical temperature gradient of the heat chamber, the central side thermistor was weighted higher than the other two for determining the average external temperature as it was reflective of the mean temperature due to symmetry.

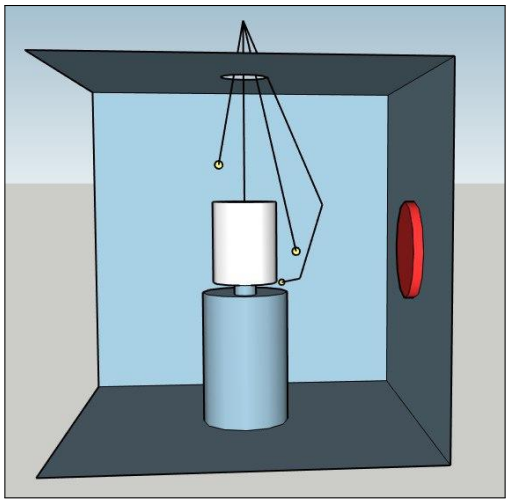


Figure 1: CAD Drawing of Heat Chamber Test Configuration

6.1.3 Microcontroller Configuration

The data we collected was collected on the TM4C123G. Our preliminary design initially developed consisted of thermistors connected to a Texas Instruments TM4C123G. To increase accuracy and computational power we changed our data acquisition unit to a Raspberry Pi 4 with an external ADC used to interface the thermistors to the Raspberry Pi. The ADC is connected to the midpoint of a voltage divider between the thermistor and a 100k Ω resistor. Steinhart-Hart equations shown below as Equation 1 & 2 were used to covert the voltage reading to a temperature reading, using coefficients found from [8] and the table [9] containing resistance values for our thermistors along.

$$R_{thermistor} = \frac{100k\Omega}{V_{dc}/(V_{dc} - V_{adcreading}) - 1.0} \tag{1}$$

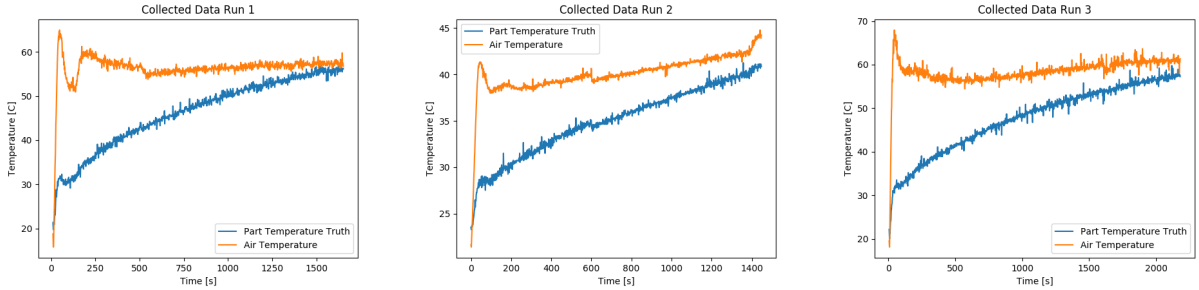
$$Where$$
$$c1 = 0.6764629190e-03$$
$$c2 = 2.230798167e-04$$
$$c3 = 0.7159342899e-07$$

$$\text{Temperature In Celsius} = \frac{1.0}{c1 + c2 * ln(R_{thermistor}) + c3 * ln(R_{thermistor})^3)} - 273.15 \tag{2}$$

6.1.4 Obtained Data

Through three separate runs, the following data was collected. The runs were collected using multiple air temperature sensors and a single sensor for the part temperature. The air temperature exhibits a very steep rise and some overshoot then a gradual rise towards zero error. The part temperature shows a initial steep

rise which is believed to be caused by a large amount infrared radiation from the lamp being on for high duty cycles. After the initial rise, the duty cycle becomes more normalized to about 30% and the rise is uniform. The end of the runs are not fully converged and usually stop before they are completely equal but for runs 1 and 2 they are within 5% tolerance. The runs also have a Gaussian noise in the readings due to variance in the electronic ADC reading, and fluctuations in air temperature.



(a) Run 1 (b) Run 2 (c) Run 3

Figure 2: Data Collected from Runs 1 to 3

6.2 MATLAB Data Generation

Here the methods for creating the MATLAB transient heat simulations for this project are explained and their results are examined. The general methodology used for generating these simulations are discussed. The specific toolboxes, functions, and strategies implemented in MATLAB are then described in this section.

The purpose of generating transient heat simulations is so the resulting data can be used for testing and training our machine learning models. Through research and investigation we found that MATLAB could be implemented for generating data sets using its finite element analysis features for modelling heat transfer problems. Building these heat simulations also made it feasible to study different materials, geometries, and temperature distributions.

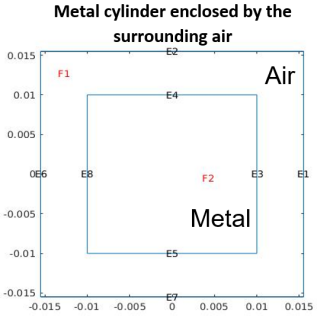
6.2.1 Governing Equations

For our transient heat simulations it was important to understand the governing equation that models the heat transfer problem. Using the approximation of a rectangular block revolved around the central axis, an idealized thermal equation was needed to create the simulations. Equation 3 shows the PDE used to model the transient conduction for the heat transfer problem. With T being temperature, ρ is material density, C_p is the specific heat value, k is the thermal conductivity, and f is the heat generated inside the cylinder. This equation is temperature dependent and is used to govern the heat conduction through the cylinder.

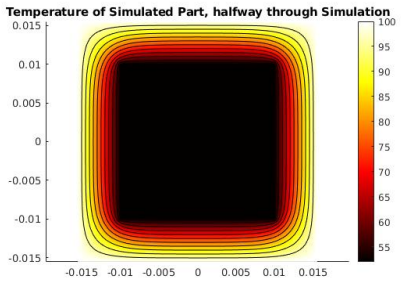
$$\rho C_p \frac{dT}{dt} - \nabla \cdot (k \nabla T) = f \tag{3}$$

6.2.2 MATLAB Simulated Data Generation

First, using the `createpde` function a PDE model of the system is created and it also sets up a transient simulation on this PDE model. Next, the geometry of the cylinder and the surrounding shape of the air is programmed. The geometry of the cylinder was approximated by using a 2D rectangle and revolving it around its central axis would then create a cylinder. This geometry is enclosed with boundary conditions set to represent the surrounding air inside the heating chamber. In Figure 3a this model is plotted showing the cylinder and the surrounding air with their position along the X and Y axes. This model also has the thermal properties programmed for the air and aluminum cylinder so the thermal conductivity across the cylinder can then be simulated. Then the temperature conditions were set to heat from the boundaries or edges 'E1, E2, E6, E7' of Figure 3a which gave a realistic simulation of the infrared lamp heating the cylinder from all directions. The `transientBCHeatedBlock` function was used to set the initial, rise, and final temperatures on these edges. This function allowed us to set the temperature conditions for the simulation to mimic the curing cycles in the physical heating chamber experiments as accurately as possible.



(a) Mesh of Metal Cylinder Approximation



(b) Heat Distribution from Simulation

Figure 3: Heat Chamber Simulations using MATLAB

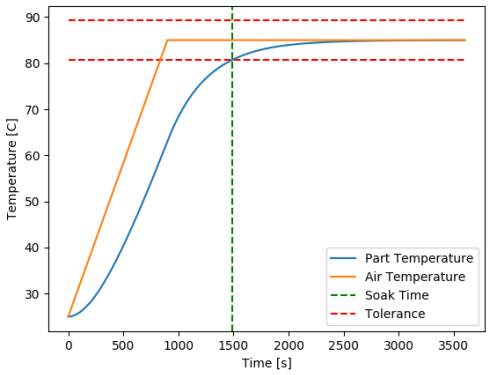
Having the model geometry, material properties, and boundary conditions symmetrical about the central axis also created a uniform heat distribution across the simulated cylinder. For the simulations a PDE with constant thermal conductivity and material properties not temperature dependent were assumed. Combining the external air temperature conditions with this constant thermal conductivity assumption produced a representation of the cylinder's internal temperature over time. The results from Figure 3b shows the simulation halfway through where there is a uniform heat distribution across the cylinder which is lagging behind the air temperature which is at 100°C.

6.2.3 Transient Heat Simulation Outputs

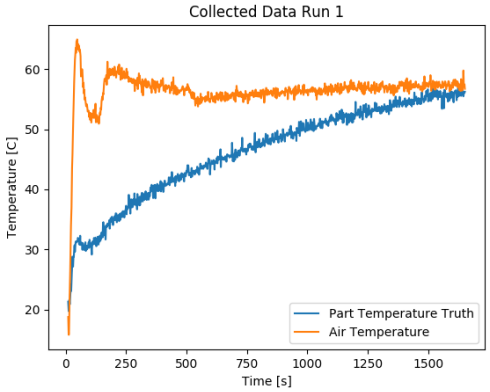
The last part of these MATLAB simulations was to generate the transient response of the model and evaluate the outputs. This was easily done using more built-in functions in the PDE toolbox. In MATLAB, the simulations were programmed to capture the temperatures at the air boundary and the center of the cylinder. The simulation is also set to save the temperature data every 1 second over the entire heating

cycle. Then when the code is run, multiple time series data sets are produced showing ideal curing cycles. An example of a simulated curing cycle generated with MATLAB is shown in Figure 4a with a maximum curing temperature of 85°C. Then in Figure 4b the curing cycle of a real data set is shown with maximum temperature at 60°C. The MATLAB data set shows the air temperature's curve having a more gradual rise compared to the curve from the heat chamber data set. This due to the settings of the PID controller in the heat chamber programmed to ramp up faster than the simulation. However, the simulations still provide robust data for training machine learning algorithms which will be able to predict the heat chamber conditions when fed experimental data after being trained on these simulated data sets.

The MATLAB transient heat simulations satisfied their overall purpose of generating multiple data sets with varying parameters. It was possible to change parameters like shape, material, conductivity of air, and temperature conditions without doing time-consuming experimental runs in the laboratory. The MATLAB code also has the ability to perform the simulations with different parameters in parallel. The parallelization of the data generation code reduced the runtime for generating 20 data sets from 1.5 hours down to approximately 15 minutes.



(a) Simulated Data with a 5% tolerance marked and soak time



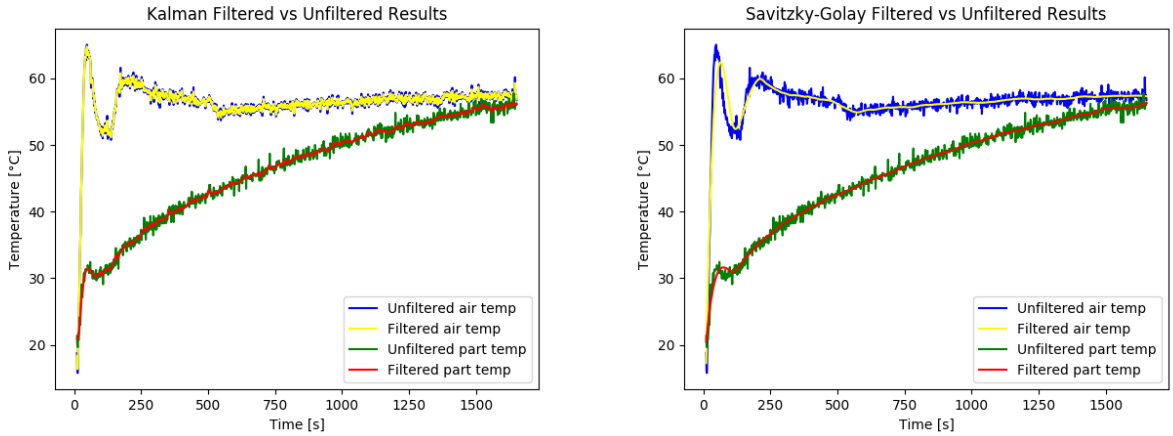
(b) Curing Cycle using Real Dataset

Figure 4: Comparison of Simulations and Real Data

6.3 Data Filtering

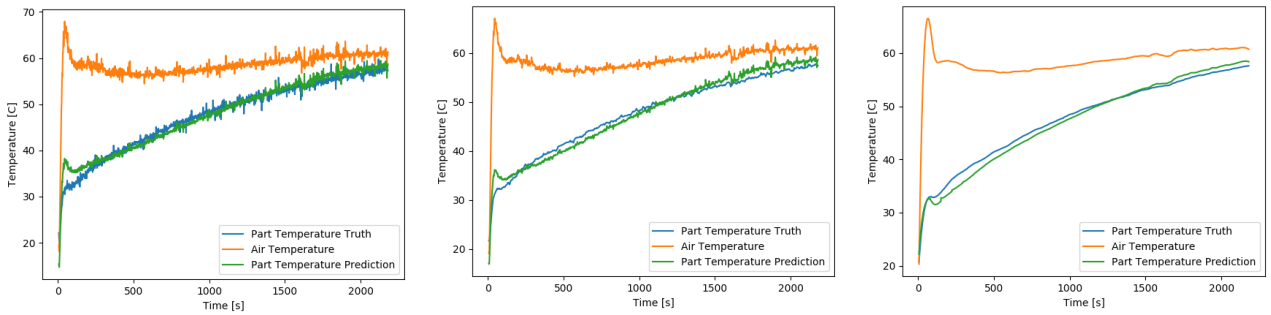
In this project, it was in our interest to understand the effects of noise from our temperature sensors on the machine learning process. We wanted to know if noise from the sensors was affecting the soak time estimate and if noise would affect the accuracy of the prediction by the machine learning methods, as the data to train on would be more inconsistent with noise present. To perform our analysis, we began by running our data through various filters: A Savitzky-Golay filter and a Kalman filter, both implemented in Python. A Savitzky-Golay filter was chosen because it is an effective filter for smoothing data through local regression and can be implemented with ease. The Savitzky-Golay filter is more sensitive to bursts of noise and therefore has the possibility of skewing the signal, however it seemed like an appropriate fit due to the uniformity of our data. A comparison of the filtered data using Savitzky-Golay filter and the unfiltered data can be seen in Figure 5b below. It can be observed that despite some inaccuracy located after the initial rise in temperature, the filter was effective in smoothing the data. Following this, we ran our sample data through a Kalman filter. For our application, the Kalman filter was used to extract the signal from the noise, which provided us with a significantly more accurate representation of the signal with smaller noise margins. The Kalman filtered data and the unfiltered data are compared in Figure 5a. The filter was ultimately able to fulfill our intended application.

To test our concerns, we trained our machine learning methods and performed predictions using unfiltered data, the Savitzky-Golay filtered data, and the Kalman filtered data. We then performed a comparative analysis on the soak time estimates of each prediction and the accuracy of each prediction relative to the true temperature of the cylinder. The figures below demonstrate the predictions from the aforementioned data through the LSTM method and the Random Forests method, respectively, for one sample run from our data set. From our analysis, we observed minor improvements through filtering the data with the Kalman filter, with similar albeit not as improved results using the Savitzky-Golay filter regarding the accuracy of the predictions. We did not observe any difference in the estimated soak time from filtering the data. We were ultimately able to conclude that the effects of noise are mostly negligible for the project application, as the noise did not have any significant impact on the machine learning process or the soak time. Further data analysis in this report is performed with the Kalman filtered data, as it provides a more visually informative and accurate view of the relationships between the curves in plotting.



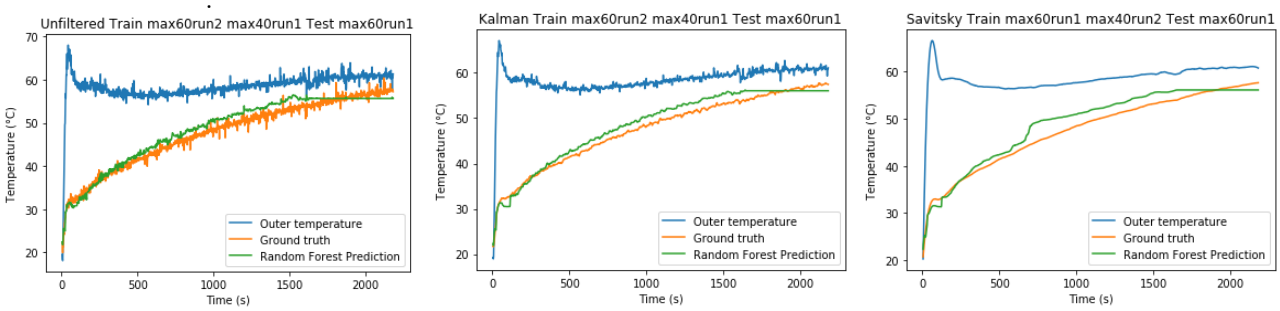
(a) Kalman Filtered Data (b) Savitzky-Golay Filtered Data

Figure 5: Data Filtering Techniques



(a) Unfiltered Data with LSTM (b) Kalman Data with LSTM (c) Savitzky Data with LSTM

Figure 6: Predictions using Filtered Data with LSTM



(a) Unfiltered Data with Random Forest (b) Kalman Data with Random Forest (c) Savitzky Data with Random Forest

Figure 7: Predictions using Filtered Data with Random Forest

6.4 Data Scaling

To input the data into a neural network, it improves performance to manipulate the data and scale it between values of 0 and 1. Large inputs which are disproportionate to each other can slow down the convergence of the training algorithm and can in some cases prevent the neural network from classifying the problem [10]. A `min_max` function is used to scale the input data between ranges min and max, which are commonly set to values 0 and 1 respectively.

$$X_{std} = \frac{(X - X_{min})}{(X_{max} - X_{min})} \tag{4}$$

$$X_{scaled} = X_{std}(max - min) + min \tag{5}$$

Time, if desired as an input feature to the neural network, can be scaled from 0 to 1 signifying the duration of the run from start to finish. As all runs follow a predictable start and finish, where at the start part temperature is equal to air temperature and at the end of the run the same is true. The time data can then be a feature which shows where in the run the network is, and if needed expanded beyond 1 to predict future data which has never been seen by the neural network. Temperature data can be scaled so the minimum temperature across all runs training runs is the minimum value used in equation 5, and the max temperature value across all training runs is the value used in 5. The scaling is shown below in Figure 8. It can be seen that the two runs which were set to 60°C are similar in nature and the run which is set to 40°C is obviously lower. Time is scaled from 0 to 1. All runs are interpolated to be the same size which is used for the LSTM network.

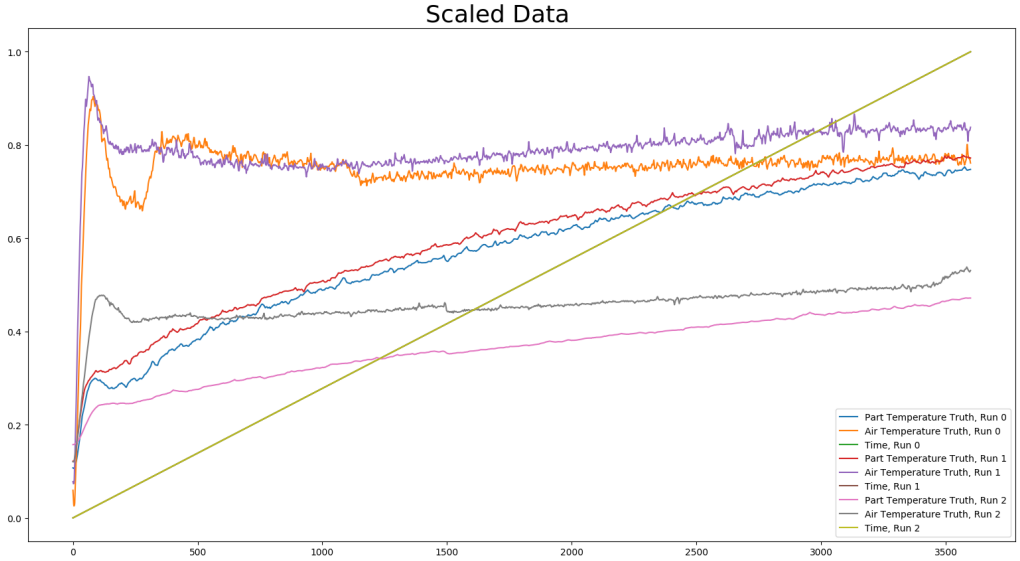


Figure 8: Scaling Input Features Part and Air Temp, and Time using Predefined Constants for Temperatures

6.5 Recurrent Neural Networks

RNNs are particularly powerful for time series prediction and classification, employing the linear and unidirectional nature of time to achieve better predictions. RNN, like other neural networks, are at the most basic level a collection of obfuscated neurons which when given an input perform arithmetic operations mainly weighting and biasing to give an output. RNN are different, in that they introduce a feed backward implementation which takes an output time step and reintroduces this as an input. The simplest is a single neuron which receives inputs and sends the output back to itself as a new input. Extending this concept, multiple neurons can be chained together with each output feeding to a neighboring neuron based on time. This version forms neuron connections aligned temporally based on the input time-steps. This is shown in Figure 9 with the simplest case on the left containing one neuron and a chain of neurons on the right. One of these yellow boxes can be implemented as a layer of neurons so that a layer of neurons feedback to a neighboring layer of neurons. Due to the action of feedback, RNNs can be thought of as having memory where past inputs are recalled as they are propagated through the chain of neurons. Many types of RNN neurons exist including standard RNN, LSTM, and GRU.

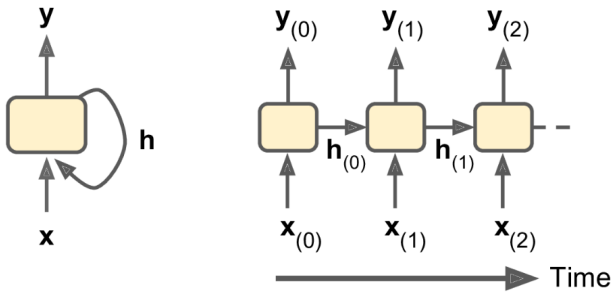
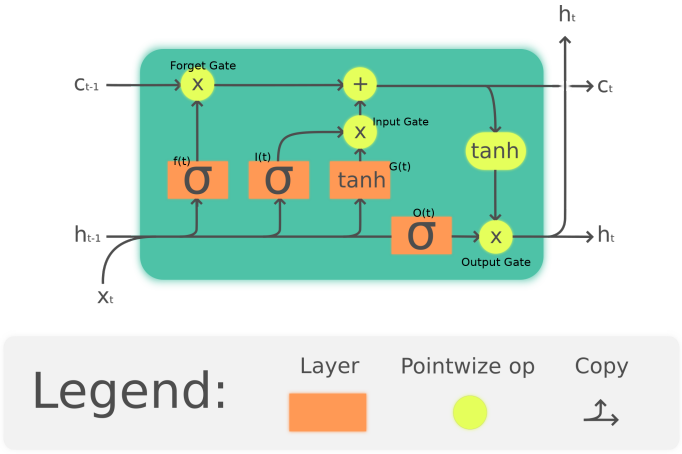


Figure 9: Block Diagram of a RNN

6.6 Long Short Term Memory Networks

LSTM networks are a particular subset of RNNs. The motivation and usage of an LSTM network are increased performance in long term dependencies in data at the cost of some additional computation power. An LSTM block takes an input x_t and two states, the long term c_{t-1} and short term h_{t-1}, which are used to determine the output state h_t which is sent to the output layer. Three gates regulate information through the LSTM dropping or allowing memories based on pointwise operation controlled by weights σ. Essentially, the LSTM attempts to recognize important input. The Forget weight controlled by $f(t)$ regulates long term memories which should be erased. Input gate controlled by $i(t)$ adds memories to the long term memory. Output gate controlled by $o(t)$ allows long term states to be read by the output layer and passed on. The governing equations for the LSTM block are shown below taken from [11]. $W_{xi}, W_{xf}, W_{xo}, W_{xg}$ are all neuron weights for the cell. $W_{hi}, W_{hf}, W_{ho}, W_{hg}$ terms are weights from the previous short term state. Terms b are bias terms or offsets. $Tanh$ functions can be changed and are the activation functions of the LSTM cell. The long term memory output c_t of the cell is a function of the Forget gate and new inputs through the input gate. The short term memory and output y(t) is determined by the activation of the previous output modified by the activation of the long term memory. These gates can be seen in Figure 10. Essentially,

the LSTM network implements gates to increase the stability of learning both short term and long term memories in time series data by allowing and dropping certain memories from the cell.



$$i(t) = \sigma(W_{xi}^T x(t) + W_{hi}^T h(t-1) + b_i \quad (6)$$

$$f(t) = \sigma(W_{xf}^T x(t) + W_{hf}^T h(t-1) + b_f \quad (7)$$

$$o(t) = \sigma(W_{xo}^T x(t) + W_{ho}^T h(t-1) + b_o \quad (8)$$

$$g(t) = tanh(W_{xg}^T x(t) + W_{hg}^T h(t-1) + b_g \quad (9)$$

$$c(t) = f(t) \bigotimes c(t-1) + i(t) \bigotimes g(t) \quad (10)$$

$$y(t) = h(t) = o(t) \bigotimes tanh(c(t)) \quad (11)$$

Figure 10: Diagram of LSTM Neuron

6.7 LSTM Parameters

Multiple different implementations were attempted to try and attain the lowest possible error. Two implementations were created, one using a simple method with a single feedback loop and another implementing data windowing. In addition to trying these two implementations, the parameters for each of these implementations were adjusted to attain the lowest possible loss on predicted data.

6.7.1 Simple LSTM

This method of LSTM network is the easiest implementation of an LSTM model which acts on time series data. The model acts on a single time step at a time, a diagram of the model is shown below in Figure 11. The time of the run and the air data are fed into the neural network one time sample at a time. The output temperature data is fed back into the LSTM. The process repeats until all samples are consumed. The LSTM network is a collection of N neurons which are formed together to create a layer of LSTM cells which all receive the data and output the data. The time data and air temperature data is scaled to better

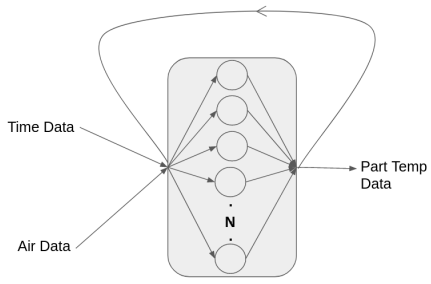


Figure 11: Input Output and feedback of LSTM layer

accommodate the activation functions within the LSTM blocks which perform between the range of 0 and 1. The data varies in length but is set to 3600 in our LSTM networks to evaluate 60 minute runs with a single sample per second. In the case of the real data which is shorter than 3600 samples data is interpolated to increase its length.

6.7.2 Parameters

The key parameters of the network include number of neurons, input dimension, dropout layer, and output layer. The LSTM layer consists of 25 neurons which take a three dimensional input array. Twenty-five neurons performed well on the data. A span between 10 to 50 resulted in fairly equal performance. The neurons are contained in a single hidden layer. Increasing the number of neurons has the possibility of over-fitting the data especially due to the low number of real runs that are available to train on. A higher number of neurons will also increase the training time. The network was set to 'Stateful' which means the network state is conserved through training batches. Through experimentation of different activation functions in the LSTM neuron, it was found that Softsign activation performs adequately. A Rectified Linear Unit (ReLU) activation also provided extremely similar performance. A dropout layer is added to the network to reduce over-fitting. The dropout layer thins the connections between neurons by randomly setting an input to a random neuron to zero depending on a certain rate [12]. The rate is set to .0001 which is very low, as we are not overly concerned about over-fitting in our data set as the data set is so small it is almost unavoidable. An output layer is added to the LSTM network and adds a bias to the output of the LSTM layer and performs an activation function on it. A linear activation function was chosen for the output layer as it was found to be adequate, and changing to other activation functions produced worse predictions.

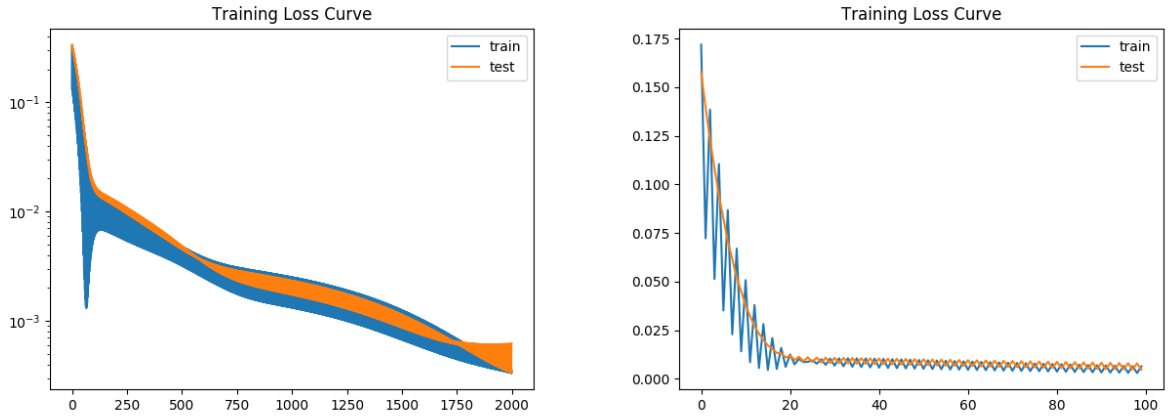
6.7.3 Training

The training optimizer used was Adam, it performed the best out of the ones tested including Adaptive gradient algorithm (AdaGrap) and Root Mean Square Propagation (RSMProp). Adam is the typical default optimizer used in machine learning algorithms [13]. A slightly reduced learning rate of 0.0005 was used, which provided lower loss over the typical rate of .001. This is likely due to the network 'learning' the data more effectively, as opposed to over-fitting the data. The downside of reducing the learning rate is the network will take more iterations and more time to train. Each trial was trained on the LSTM network for a single epoch, then a new trial was selected from the training set and this was repeated until the loss was reduced adequately. The model is fit in batches meaning only portions of the data is given to the network at a time. Smaller batches can increase the stochasticity of the gradient descent during training which can allow the algorithm to jump out of local minima and find the proper minimum for the network [13]. A batch size of around 180 samples was found to be adequate. To calculate the loss, the mean-squared-error (MSE) loss function was used. This loss function is the standard and provides a harsh loss compared to other functions meaning small deviations will produce a higher relative loss. The loss function is shown in Equation 12. It was found that moving to other loss functions always produced worse prediction capability. Usual training

took up to 1000 epochs before the loss stopped being reduced on the validation set.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{12}$$

The loss during training of the network is shown in the figure below. It can be seen that the loss drops quickly at first and then slowly later, and the test loss is higher than the training loss. The network takes under 15 minutes to train on an Intel® Core^{TM} I5-4760k 4core 3.4GHz Processor with time depending on the amount of training data used.



(a) Loss Curve 2000 epochs

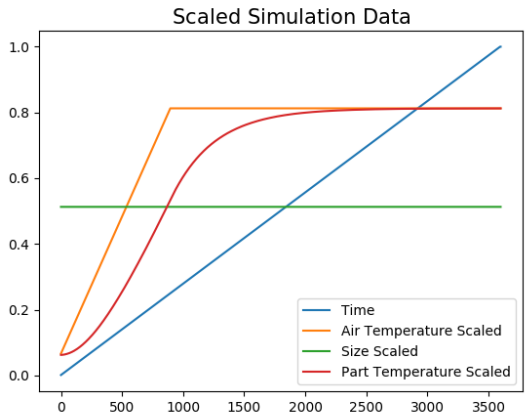
(b) Loss curve 100 epochs

Figure 12: Loss Graphs using MSE

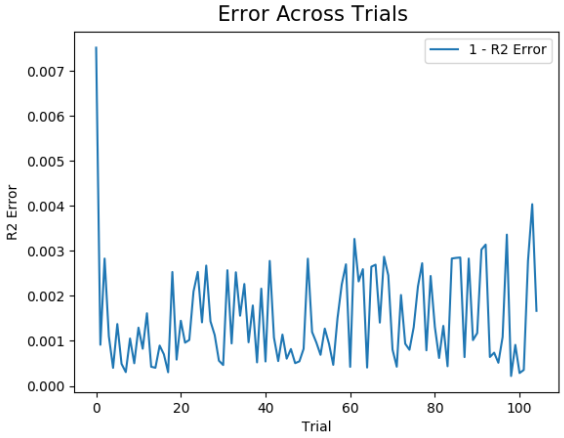
6.7.4 Simple LSTM Simulated Data Performance

The simulated data is a good case where there is a large pool of data and an interpolation is made on unknown data. The data is presented below in Figure 13a. The machine learning algorithm shows a choppy output and the batch size can be seen where the machine learning algorithm was trained in batches. The error is low for all runs except for the first run shown in Figure 13b but it can be seen that even this run is sufficient enough to make a prediction on soak time. It was also found that the performance could be vastly improved on simulated runs by scaling the temperature of trials using an individual max and min found from solely that trial. This would result in every temperature curve starting at 0 and ending at 1, and the rise of the part temperature would be extracted solely from the rise of the air temperature. Doing this kind of scaling would result in worse performance on real data as the air temperature does not exhibit a similar rise, so it was not used.

.

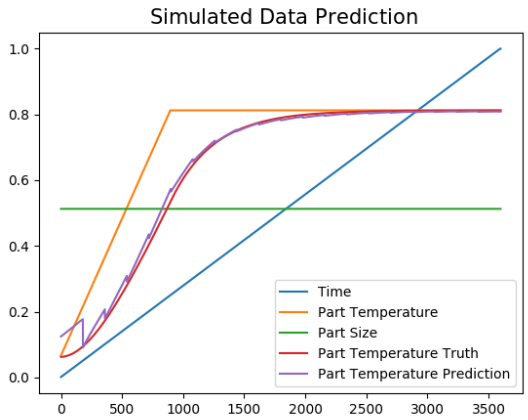


(a) Scaled Simulated data Input Features and truth

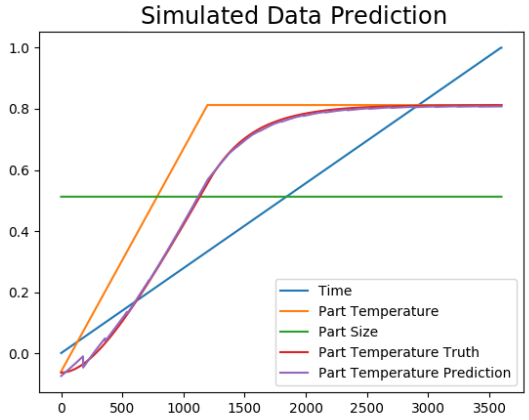


(b) Loss Across Runs R2 Error

.



(c) Highest Loss Run



(d) Average Loss Prediction

Figure 13: Simulated Data Information

6.7.5 Simple LSTM Real Data Performance

Real data was input two runs at a time and a prediction was then made on the third run. The summary of predictions is shown below in Figure 14. The predictions are better when stateful is set to false, most likely because the air data has a lower impact on the part temperature in the real data. The prediction of the 40°C run by training on two 60°C runs was poor due to the fact that no similar data was used to train and there was a limited amount of data to train on.

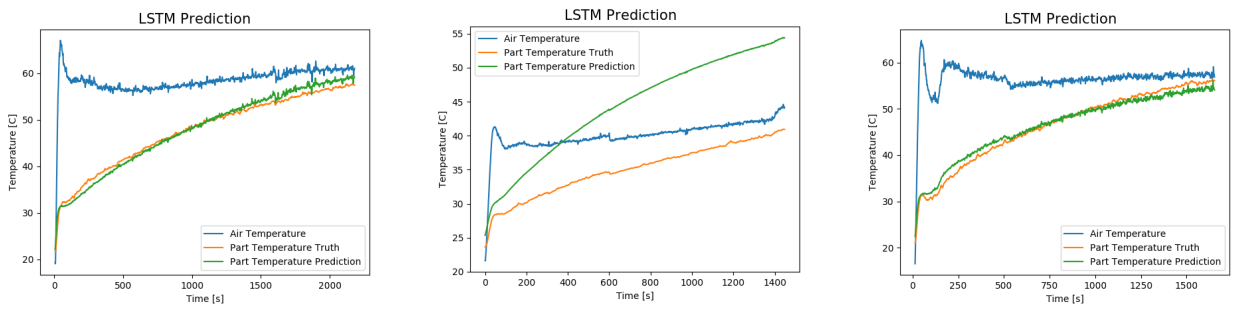


Figure 14: Real Data Predictions

6.7.6 LSTM Windowed

In an attempt to improve upon the simple LSTM model outlined above, another LSTM model was created using multiple time steps to make a decision. Unrolled, this model would be like the network below on the right. The network shares similar parameters to the other simple LSTM network. This network takes longer to train but can result in better predictions. The parameters used for the windowed method are the same as above and can be found in section 6.7.2.

6.7.7 Windowed Data

Windowed data is a way of organizing input features of the neural network to input samples and feed time data forward. The window implements a collection of data of past t-1 to t-N samples which are used to make a prediction on sample t. N can be any value, for our implementation a 50 sample window was chosen. An example of a window of data pointing at part temperature is shown in Figure 15 to the right, demonstrating what a window of data attempts to do.

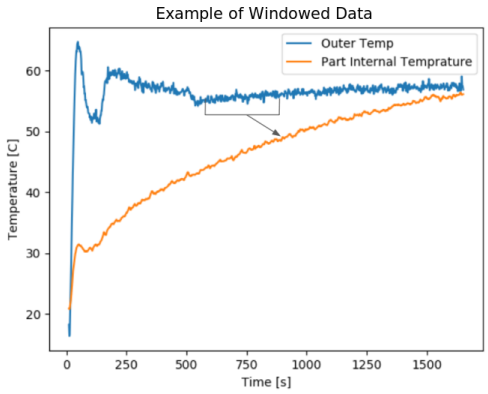


Figure 15: The window of data underlined which would be used to predict a single sample of part data which the arrow is pointing too

6.7.8 Windowed Method Prediction on Real Data

The windowed method predicts slightly better than the simple method and is able to more closely fit data that it is trained on. It may also over-fit the data more. A dropout layer can help prevent over-fitting of the data although with this small of a data set to train on it makes minimal difference. From looking at the runs, it seems that the data is being learned more than overfit. This is exemplified by the dependency that part temperature has on the external air temperature shown by the hump from 500s-1000s in Figure 16. The end of the run matches closely with a percent error below 2%, which is important for determining the soak time. Using Equation 16 the R^2 score was found for the first run to be 0.9824 and 0.9834 for the second run. Below in Figure 16 & 17 the predictions are shown. The 40°C run was not shown as it has a similar poor performance as the simple LSTM method shown above in Figure 14 in section 6.7.5

.

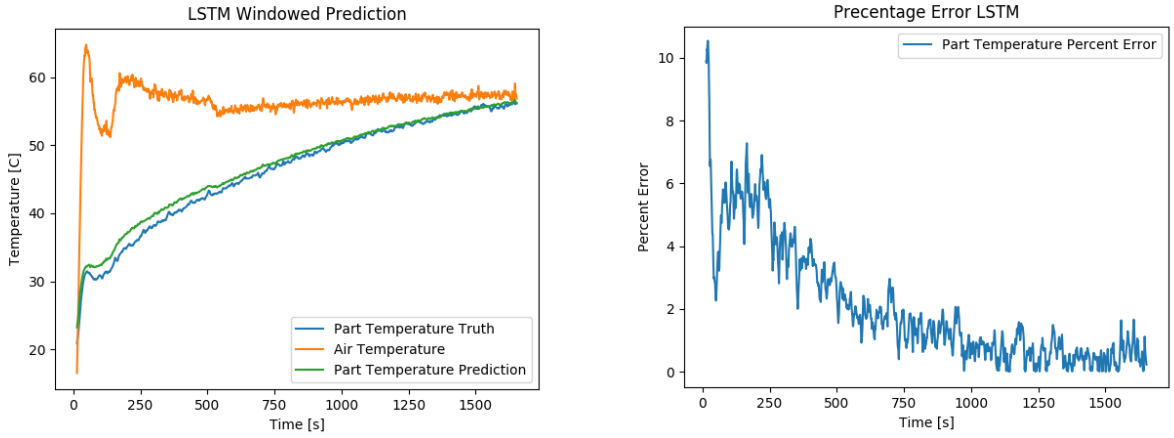


Figure 16: Real Data Predictions Windowed Data 1

.

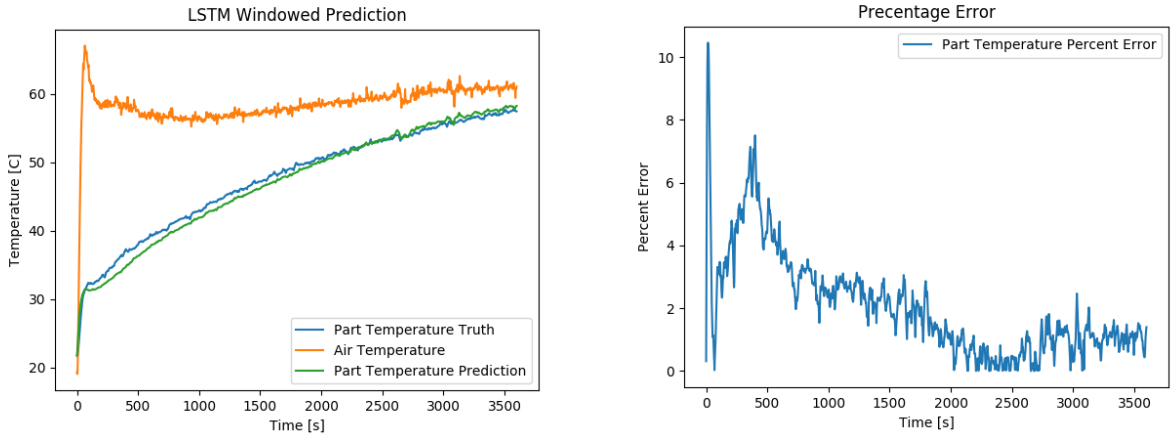


Figure 17: Real Data Predictions Windowed Data 2

6.8 Soak Time Determination with LSTM

The LSTM can also be used to make predictions from air data while the run is executing. This means that not all air temperature data is available so it is instead extrapolated to try and evaluate when the run should be finished. The simplest way to predict what the air temperature will do is to extrapolate by assuming it will hold the same temperature now until the end. Alternatives to this method include trying to do a polynomial regression or creating a neural network which predicts on air temperature. The extrapolation happens continually until a consensus on soak time can be made.

Below in Figure 18 is an example of a prediction being made half way through one of the data sets. Here the soak time is set so that the part temperature should be within 5 percent of 55°C so within about 53°C and 57°C. In Figure 18 the prediction is being made at the blue vertical line using extrapolated data shown in red, the predicted soak time is shown in grey and the true prediction using part temperature truth is shown in black. It can be seen that the part temperature truth lost accuracy due to the extrapolation of air temperature. Click or double click on Figure 19 below to see a video of the live prediction being graphed out a step rate of roughly a prediction every five seconds.

.

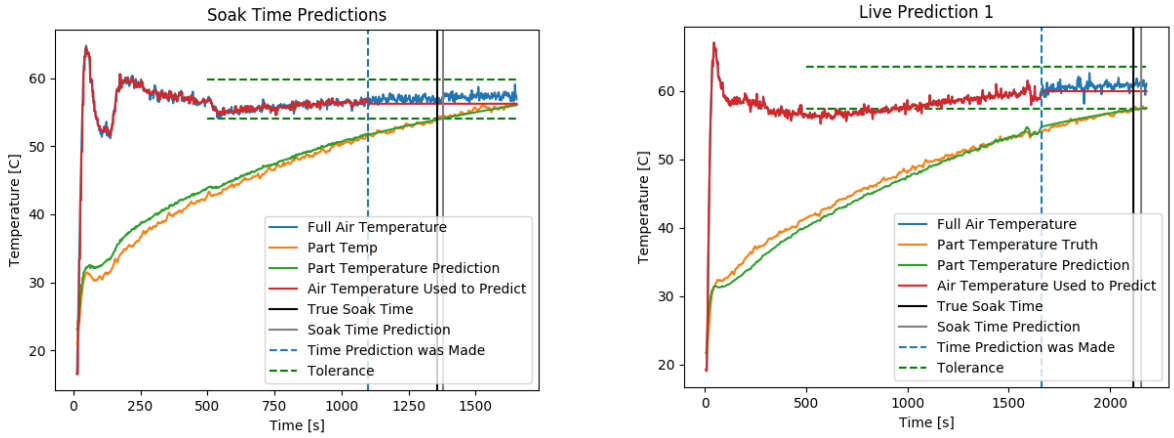


Figure 18: Live Predictions Windowed Data 1 & 2

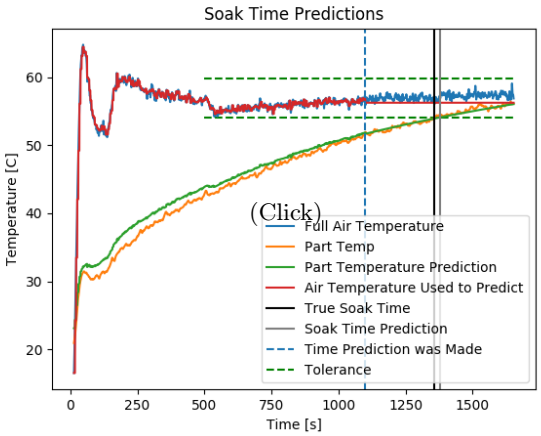
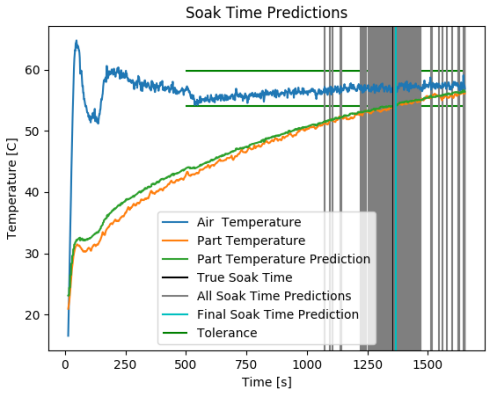
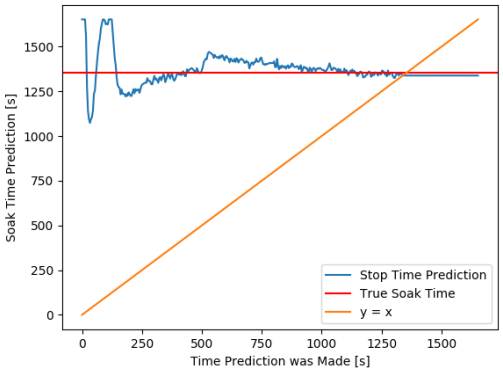


Figure 19: Prediction video showing live prediction (double click, works in firefox), or follow the Link

In Figure 20 below, predictions were made on extrapolated data every five seconds. The plot in Figure 20a shows vertical lines in grey for every soak time that was predicted and a single black line for the true soak time. Figure 20b is a plot which shows the predicted soak time versus when that prediction was made. If the prediction is on the left half of the orange line, then that means it is predicting a soak time that is in the future. Once it starts predicting on a soak time in the past, the greatest confidence in accuracy has been achieved. One could then make a control decision to use the prediction that occurs as close as possible to the orange line to obtain the best accuracy. Using this control choice, a result was found where the estimated soak time of 1338.656 seconds versus the real soak time of 1356.877 seconds was found. This would result in a soak time about 20 seconds shorter than needed in a run which takes 22.3 minutes to complete. The other run estimated soak time as 2029.401 seconds when the true soak time was 2115.839 seconds and was 86 seconds off in a run that is 2179.91 seconds or 36.3 minutes long. It can be seen that as the run progresses and gets closer to the true soak time, its accuracy increases and converges to its final prediction. There is a small error in the soak time determined as the prediction is slightly different than the true part temperature curve.



(a) Soak Time Prediction Made every 5 seconds, Correct Prediction in Black, LSTM Predictions in Cyan/Grey



(b) Soak Time Prediction and When it was made

Figure 20: Live Predictions

6.9 Forward Prediction with LSTM

It is also possible to train on every trial that is available in our data set then extrapolate the trend of air temperature to try to predict the future. This has use cases in seeing where the internal temperature is and how it will be in the future. This shows how soak time can be determined before it is reached allowing a control decision to be made. The simple LSTM's prediction is shown below in Figure 22. The trends are sensible except for the 40°C run.

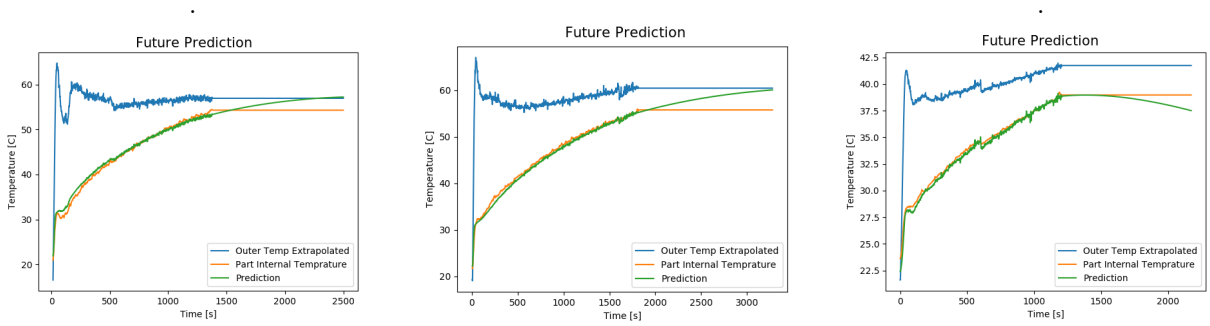


Figure 21: Future Data Predictions Simple Method

Below in Figure 22 is the LSTM prediction into the future with the LSTM windowed method. As the air and part temperature converge the part temperature's slope does not decrease as expected. This would most likely result in an underestimate in soak time. Future predictions are an interesting look into how the LSTM believes the run will progress. It is hypothesized that with data that fully converges, these predictions would become more accurate.

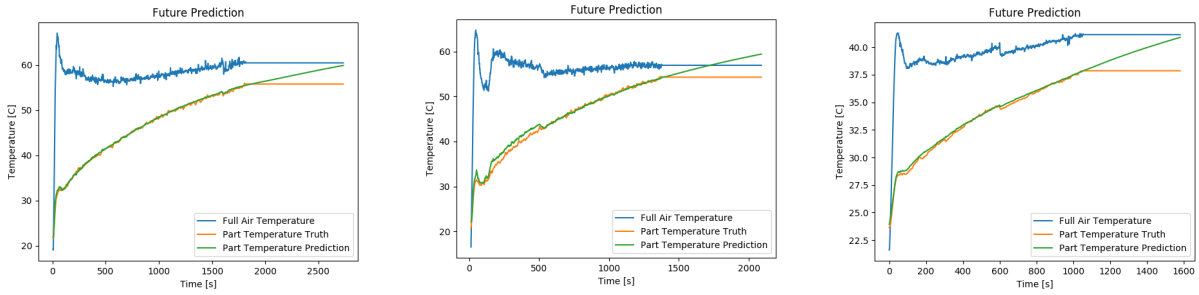


Figure 22: Future Data Predictions Using Windowed Method

6.10 Random Forests

Random Forests operate on two main principles, decision trees and bagging. While according to [14], Random Forests are popular mostly in classification tasks, it is a versatile algorithm that can be used to solve other tasks, such as regression. According to [14], "A decision tree is a set of questions organized in a hierarchical manner and represented graphically as a tree". For each input, the decision tree asks successive questions about its known properties and splits into different paths. Depending on the path traversed through the tree, the questions asked will differ. Once the terminal leaf node (end of the path) is reached, the output is predicted.

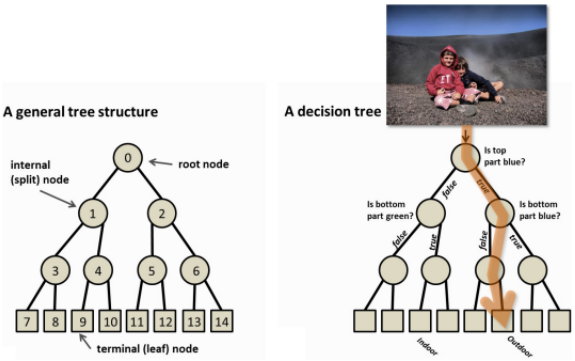


Figure 23: Diagram of Decision Tree Structure retrieved from [14]

To decide how paths are split, split functions based on metrics such as Information Gain, Gini Index, or Mean Squared Error are used. In Scikit-Learn's implementation of decision trees and Random Forests for regression, Mean Square Error is used by default (Eq. 14) where m represents a node, R_m represents a region, and N_m is the number of observations. Mean Absolute Error can also be selected as a split function for regression and is shown in Equation 15.

$$\bar{y}_m = \frac{1}{N_m} \sum_{i \in N_m} y_i \tag{13}$$

$$H(X_m) = \frac{1}{N_m} \sum_{i \in N_m} (y_i - \bar{y}_m)^2 \tag{14}$$

$$H(X_m) = \frac{1}{N_m} \sum_{i \in N_m} \mid y_i - y_m \mid \tag{15}$$

Random Forests use the concept of bagging and an ensemble of decision trees to increase the performance over an individual decision tree. Bagging is a technique in which several weak learners, in this case decision trees, are trained on bootstrapped data sets with randomly selected features. Bootstrapped data sets for each decision tree are created by sampling the training data set with replacement, effectively resulting in

different data sets. By randomly training decision trees on bootstrapped data sets, the correlation between each tree is reduced, improving generalization and robustness of the ensemble [14].

6.11 Random Forests Parameters & Performance

In this section, we analyze the parameters and performance of the Random Forests machine learning algorithm on the generated and real recorded data.

6.11.1 Parameters

While there are various parameters such as the number of estimators, criterion, maximum depth of the tree, the minimum number of samples required to split a node, and the minimum number of samples required to be a leaf node, the parameters we used in our Random Forest model were left at the default settings. The reasoning behind this decision was because Random Forests are known to typically require minimal parameter tuning. Additionally, we did not have enough real-world data sets to cross-validate and tune the performance of our model before validating on a test set.

6.11.2 Simulated Data

Using simulated data, we can test the scenario in which we have a large pool of data and want to make a prediction on unseen data. The performance of Random Forests was evaluated with leave-one-out cross validation. In leave-one-out cross validation, given a pool of data sets, one data set is selected for testing while the remaining data sets are used as a training set. The score for each data set was then averaged which gives us our cross validation score. Scikit-Learn determines the performance of a model using R^2 which is determined by Equation 16 where u is the residual sum of squares, v is the total sum of squares, and $\bar{x}_{y_{true}}$ is the average of the true output labels. The score ranges from a value of 0-1 where 1 is the best possible score.

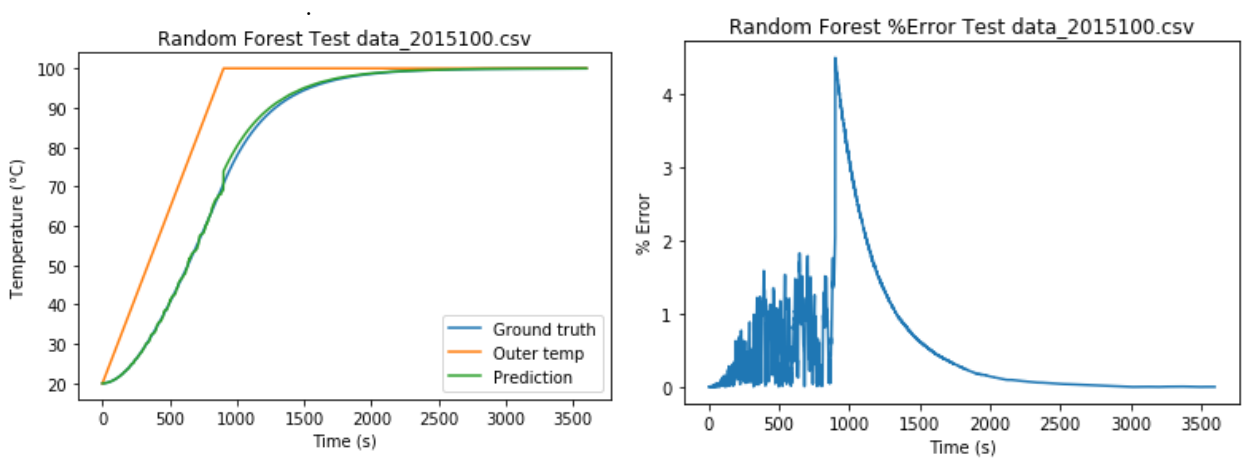
$$R^2 = (1 - \frac{u}{v}) \tag{16}$$

$$u = \sum (y_{true} - y_{pred})^2 \tag{17}$$

$$v = \sum (y_{true} - \bar{x}_{y_{true}})^2 \tag{18}$$

In our testing, Random Forests performed quite well on simulated data and managed to get a cross validation score of 0.9986. Thus we can determine that on simulated data, Random Forests are a suitable method. An example of one of the tests in Figure 24a shows that the ability of the Random Forest is near perfect. In this

case, the percent error of a prediction at any given time constantly remains below 5% as seen in Figure 24b. The performance of the Random Forest appears to worsen as the slope of the internal temperature curve decreases, however once the internal temperature curve begins to flatten and the soak temperature is met, the Random Forest predictions stabilize.



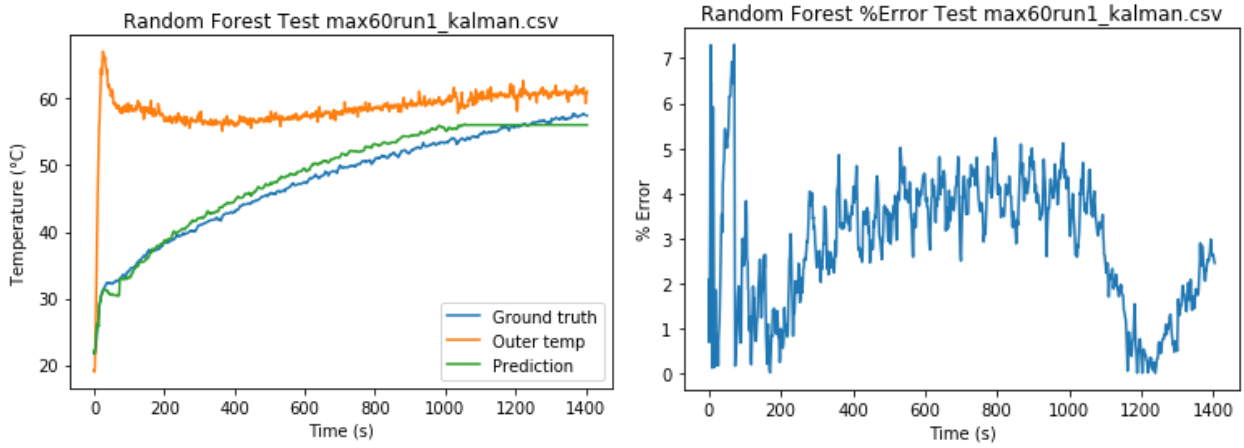
(a) Prediction with Random Forests

(b) Percent Error

Figure 24: Simulated Data Performance using Random Forests

6.11.3 Real Data

When testing on our real-world data, we once again used leave-one-out testing. In this case since the amount of data we have is minimal, we will look at the results of each test individually. As our data was initially very noisy, we first ran our data through a Kalman filter before training the Random Forest. Looking at the results of max60run1 in Figure 25a and 25b, we can see that the Random Forest has decent performance with an R^2 score of 0.9608 and typically has a % error of less than 5%.

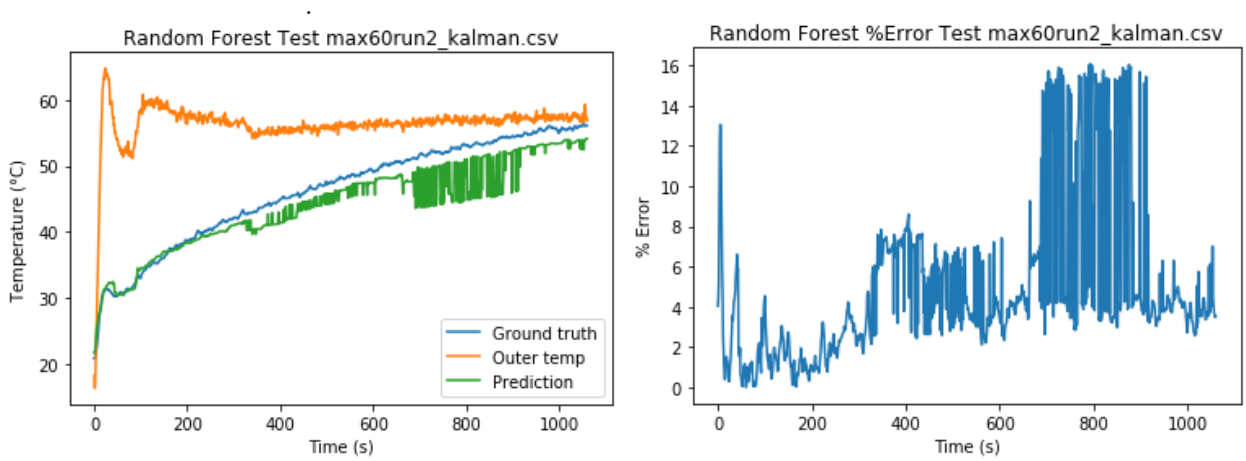


(a) Prediction with Random Forests

(b) Percent Error

Figure 25: Real Data Performance using Random Forests - Testing max60run1

For max60run2 in Figure 26a and 26b, the performance is initially quite good before diverging around the 350 second mark. Around the 700 second mark, the predictions became very unstable before stabilizing when the part gets close to the designated soak temperature. The poor performance in this case can likely be attributed to over-fitting due to a lack of variety in data.

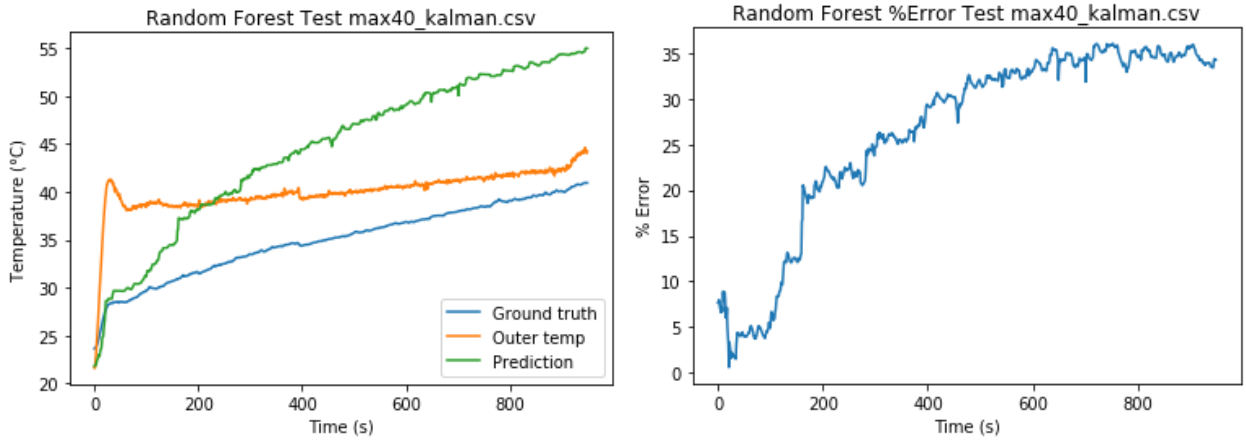


(a) Prediction with Random Forests

(b) Percent Error

Figure 26: Real Data Performance using Random Forests - Testing max60run2

On the max40 run, in Figure 27a and 27b, the performance as expected is incredibly poor. This poor performance is expected as the Random Forest was not trained on a temperature curve with a maximum temperature of 40°C. The typical scenario that should be expected is to train the model on a range of temperature curves where the test data set will fall in between.



(a) Prediction with Random Forests

(b) Percent Error

Figure 27: Real Data Performance using Random Forests - Testing max40

7 Final Design Evaluation

7.1 Assessment of Final Design

Upon evaluation of the project requirements, needs, and constraints identified, it can overall be stated that the design project was a success and completed to the satisfaction of the stakeholders. The primary deliverable of the project was to develop a process for predicting the lag temperature of the metal cylinder using a machine learning algorithm and to use the information to estimate the soak time. We were able to accomplish our objective using Python along with LSTM neural networks and Random Forests for our machine learning algorithms. A comparison of the effectiveness of the LSTM method and Random Forests method can be seen in Figure 28 and Figure 29. It can be seen from the graphs that the LSTM method is overall more reliable and superior compared to the Random Forests method, as the percent error is generally smaller, more consistent, and less sporadic. The error patterns observed for LSTM represent mostly an offset whereas the patterns observed for Random Forests are more random. Furthermore, the LSTM method is more effective for determining soak time as the prediction by the end of the cycle is more accurate. The estimation of the soak time was implemented into our Python scripts. The system designed for this project establishes an Industry 4.0 framework that can be further expanded upon, and overall the final design is an effective proof of concept for demonstrating the feasibility of this system for composites manufacturing. A practical solution for finding the soak time was implemented in a testing process where it is otherwise unknown.

In consideration of our other project needs and constraints, we were able to improve and further optimize the data collection process. Better thermistors were purchased and their optimal placement was determined through testing. An ADC was also purchased and these changes allowed us to improve accuracy in our sensor readings and confirm that our results were consistent. The signal-to-noise ratio was taken into consideration as we ran the collected data through filters in order to boost the signal-to-noise ratio. The budget constraint was not a concern, as roughly half of the budget was used throughout the entire project. Safety precautions were always taken while testing in the lab and no danger was ever present.

One disappointment in our project is that only a small sample of valid data was collected and used for developing the machine learning processes. The small amount of data retrieved was due to limited access to the Composites lab following the COVID-19 outbreak. With minimal data collected, the integration of cloud storage through IBM Cloud was invalidated as it would not be useful for a small data set. Ultimately, most of the requirements, needs, and constraints established by the stakeholders were successfully accommodated and fulfilled.

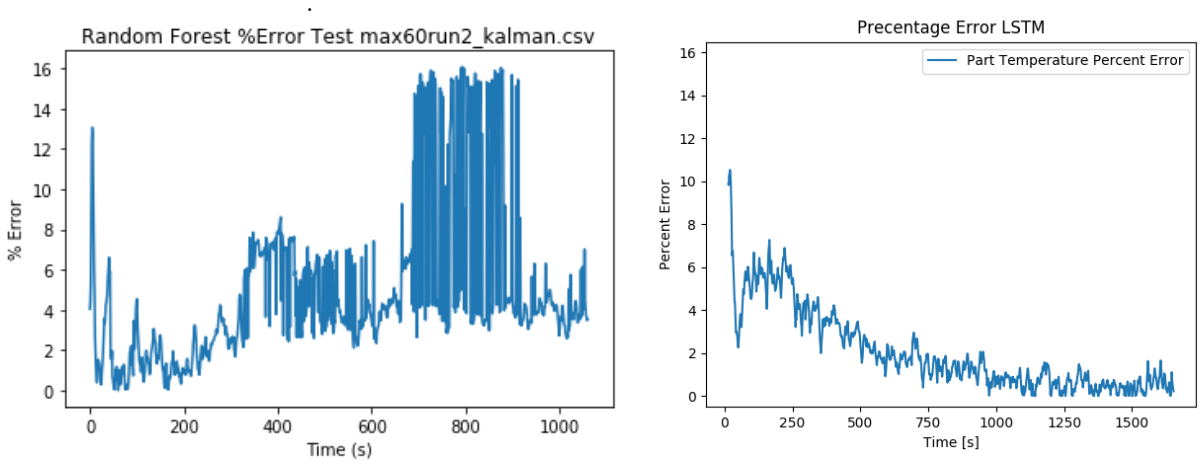


Figure 28: Real Data Percent Error Comparison

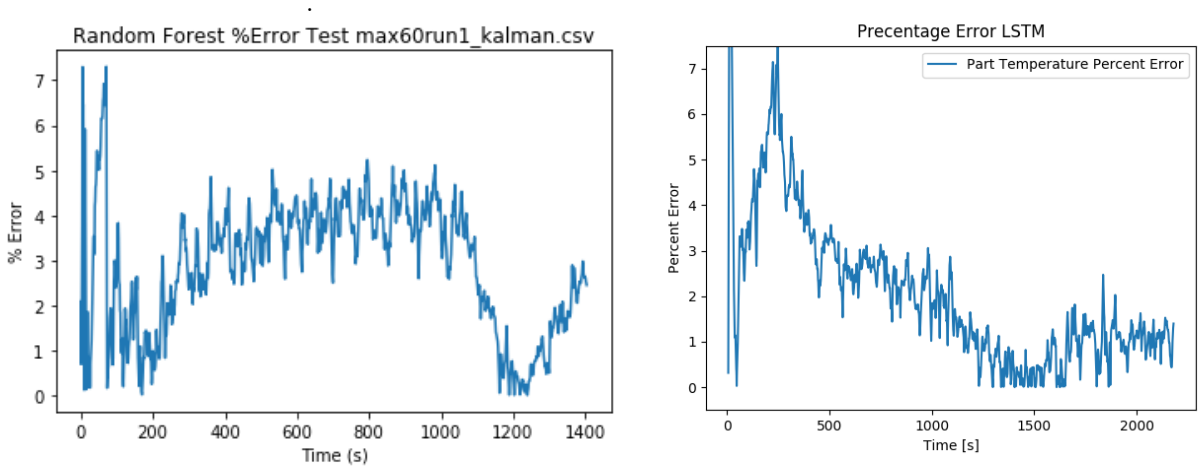


Figure 29: Real Data Percent Error Comparison

7.2 Coronavirus Restrictions

Due to closure of non-essential laboratories at UBCO in response to COVID-19 as of March 17, 2020, there were a number of topics we were unable cover or implement. In particular, we were unable to gather more reliable data using our redesigned Raspberry Pi based data acquisition unit. With more data, we would be able to more thoroughly validate our machine learning models. In addition, while the code to predict internal temperature in real-time exists, as we were unable to gain access to the laboratory, we were unable to implement or test this feature with the heat chamber.

7.3 Recommendations for Future Work

There are a number of recommendations that can be made for further progressing this project. Some suggestions include: further testing, improving the simulations for generated data, data assimilation, and transfer learning.

7.3.1 Further Testing

One avenue for future work is expanding upon the machine learning process through further testing. This includes gathering additional data for the aluminum cylinder used in this project as well as a comparable volume of data for other parts consisting of different materials and shapes to be used for further training the machine learning models. As it is eventually the goal to implement this type of system for determining the soak time for composite materials, it is necessary to incrementally train and update the machine learning models developed for this project to accommodate other parts and materials before being accessible to composites.

Another test we wanted to perform but were not able to due to separation of the logger and PI controller was to extract the duty cycle of the PI controller over the run. As seen in Figure 30 to the right and circled in red, there is a steep rising curve initially for the part temperature which is likely due to an increased amount of heat transferred through radiation because the infrared lamp was on for high duty cycles in our collected data, and without extracting the duty cycle we could not use that information at the initial point in time. It is expected that with this additional information the machine learning algorithm would have an easier time adjusting for the offset that the part temperature is at throughout the run.

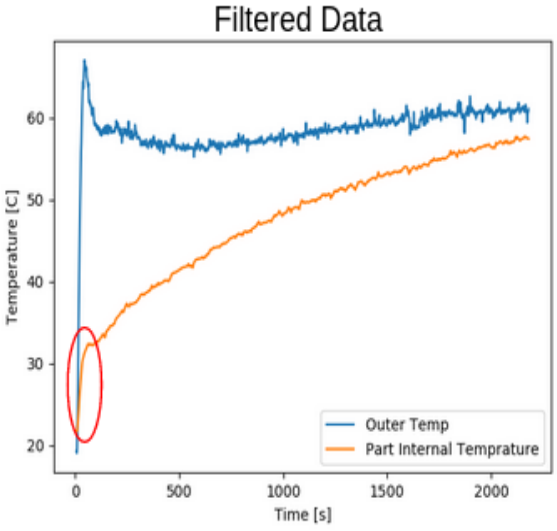


Figure 30: Filtered Data showing

7.3.2 Improving Simulations

There are some limitations in the simulations performed for this project, and further improvements in the simulation parameters would aid in validation of the machine learning models and the data collected. In the MATLAB simulations performed, a two-dimensional rectangle was modeled to approximate the three-dimensional cylinder used in the project. While this was a simpler process, more accurate and reliable data could be generated by importing an identical 3D mesh of the cylinder for the simulation runs.

Another adjustment that could be made to the simulations is to account for more parameters. For simplicity, the model used for the simulations referenced in this report only account for conductive heat transfer, but a more thorough model would also account for convective heat transfer and radiation. The thorough model is more useful as it provides a more accurate and practical view of the heating system and its properties.

7.3.3 Data Assimilation

Another possible improvement for this project is data assimilation which is the use of computer generated data such as a simulation combined with real world data to create synthetic data which is assimilated with the real data to train a neural network. As we have both of these components, further testing could be implemented. It was found that the simulation needed to be tweaked further to attain data that improved the prediction ability. A small amount of testing was conducted in this area on the LSTM model but it was found that it did not improve the prediction ability on real data. It may have only decreased over-fitting of the model. It is challenging to come to a definite conclusion, especially since the simulated data vastly outnumbers the real data. The most interesting result from data assimilation would be to decrease the number of real runs that need to be conducted therefore this would be an area for further research.

7.3.4 Transfer Learning

Lastly, a worthwhile concept to look into is Transfer Learning. Transfer Learning is a domain of machine learning which uses the training data or model from one problem and adapts it towards a new but related problem. As we saw some cases of over-fitting during our testing, transfer learning could possibly used as an option to counteract over-fitting and improve generalization performance. In particular, Transfer Learning could work well in cases where we have a limited amount of training data for a certain target scenario (e.g. new material, different object geometry) but have related data from previous experiments or other data sets. Some interesting Transfer Learning methods to look into include [15], and [16]. Both of these Transfer Learning methods have good performance and are simple to implement and require less than 10 lines of code.

8 Conclusion

Through this Capstone project, we were able to collaborate with the UBC Okanagan Composites and Optimization laboratory to create an Industry 4.0 solution to the composites manufacturing process in the lab. The objective was to develop a proof of concept machine learning process to optimize the curing cycle of composite materials by estimating the soak time. To develop our framework, an aluminum cylinder was used for testing and gathering data for the machine learning algorithm. Through our project, we were able to improve the data acquisition process by incorporating a Raspberry Pi and ADC to the test configuration. These devices were used to make the curing process simpler and to retrieve more accurate readings from the temperature sensors. From the literature review and research performed, we decided on using Random Forests and LSTM for our machine learning process in order to compare the results given by different methods and to observe which is more effective for our needs. Python was decided for implementing the machine learning algorithms for its extensive selection of machine learning libraries and its accessibility.

An engineering approach was adopted to resolve concerns that arose during the development of this project. To address the concern of noise, we experimented with different filters and analyzed their effectiveness to improve the signal-to-noise ratio while remaining accurate and seeing how it affected the results of the machine learning processes. Ultimately, we found that noise had little effect on the machine learning algorithms or on the soak time of a test run. Simulations were also performed in MATLAB to complement the real data that was obtained. The simulations allowed us to further understand the test process and allowed us to generate a greater data set to help the modeling of the machine learning algorithms. We developed two machine learning algorithms one using Random Forests and another using a LSTM model. After extensive development on the machine learning processes, we were able to generate predictions with a percent error under 4% for an entire run. The machine learning algorithms were then applied to predict the soak time of a simple metal part. This could then be used to make a control decision when to stop a curing cycle. The machine leaning algorithm was also used to try and predict how the part would heat up in the future. Despite limitations raised due to the COVID-19 outbreak, we were ultimately able to successfully fulfill the stakeholder's needs. For further development on this design, our recommendations include: accumulation of more data for further fine-tuning of the machine learning processes, more advanced simulations, data assimilation, and transfer learning.

Appendix

Github

Link to GitHub: https://github.com/darryllam/CRN_Heat_Chamber

LSTM Model and Weights

LSTM weights are saved in the git repo to see which weights are used where check the file CRN_Heat_Chamber/python_src/README which calls the python script to access the weight file.

- `model_weights_Jup_alt_scale_april.h5`, trained on variable size data found in jupyter folder
- `model_weights_real_data_1_20_70_april11.h5`, trained on kalman data kalma_max40.csv & kalma_max60run2.csv
- `model_weights_real_data_40_20_70_april11.h5`, trained on kalman data kalma_max2.csv & kalma_max60run1.csv
- `model_weights_real_data_2_20_70_april11.h5`, trained on kalman data kalma_max40.csv & kalma_max60run1.csv
- `model_weights_train_all.h5`, trained on kalman data kalma_max40.csv & kalma_max60run2.csv & kalma_max60run1.csv
- `model_weights_train_all_w.h5`, trained on kalman data kalma_max40.csv & kalma_max60run2.csv & kalma_max60run1.csv with window size 50
- `model_weights_real_window1.h5`, trained on kalman data kalma_max40.csv & kalma_max60run2.csv with window size 50
- `model_weights_real_window2.h5`, trained on kalman data kalma_max40.csv & kalma_max60run1.csv with window size 50

LSTM Code Usage

The code runs in the command line and options are given through command line options. The test path and validation path point to folders only containing csv files. Input file is to input a network model. Out file is the output file to save a network model. The window size, epochs, minimum air temp, maximum air temp can all be adjusted with integers. Values are input to point to columns in the data. See CRN_Heat_Chamber/python_src/README for files which can be used to easily execute the python lstm files using the linux command `>>source FILE`. Below is an example.

```
python3 -i lstmMethod.py -tp ../real_data_kalman/train/ -vp ../real_data_kalman/test/
-o weights_real_test.h5 -w 1 -vcol 2 -stcol 0 -tmcol 1 -min_temp 15 -max_temp 70 -e 1 -state
```

0

-tp is train path -vp is val/test path -o is output file for model or -i is input weight for model

-w is how many windows you want

-vcol is column of data that holds part temp info -stcol is column of data that holds time or other data that you want to scale on a per trial basis -tmcol is air temp column -min_temp is minimum temp used when scaling temperature data -max_temp is maximum temp used when scaling temperature data

-e is number of epochs to train that data for -state 0 is to set stateful to be off which is typical Example for predicting data for future prediction and real time prediction:

```
python3 -i predict_windows.py -vp ../real_data_kalman/test/ -i weights_real_test.h5 -w 1 -vcol
2 -stcol 0 -tmcol 1 -min_temp 15 -max_temp 70 -future 1.25 -live 900
```

-vp is val/test path

-i is input weight for model

-w is how many windows you want

-vcol is column of data that holds part temp info -stcol is column of data that holds time or other data that you want to scale on a per trial basis -tmcol is air temp column -min_temp is minimum temp used when scaling temperature data -max_temp is maximum temp used when scaling temperature data

-future is a fraction which is multiplied by data len to create a longer array, and will be predicting if longer than data_len. Must be a multiple of batch size -live is step size to be making predictions

References

[1] (2019). Composites research network, [Online]. Available: `Crn.ubc.ca` (visited on 09/18/2019).

[2] S. Nakouzi, J. Pancrace, F. M. Schmidt, Y. L. Maoult, and F. Berthet, "Curing simulation of composites coupled with infrared heating," *International Journal of Material Forming*, vol. 3, pp. 587–590, Mar. 2010.

[3] P. Shah, V. Halls, J. Zheng, and R. Batra, "Optimal cure cycle parameters for minimizing residual stresses in fiber-reinforced polymer composite laminates," *Journal of Composite Materials*, vol. 52, no. 6, pp. 773–792, 2017.

[4] R. Schoeman, J. V. Rensburg, and D. Nicolae, "Self-tuning curing oven control," *12th International Conference on Optimization of Electrical and Electronic Equipment*, 2010.

[5] P. E. Jahromi, A. Shojaei, and S. M. R. Pishvaie, "Predication and optimization of cure cycle of thick fiber-reinforced composite parts using artificial neural networks," *ournal of Reinforced Plastics and Composites*, vol. 31, pp. 1201–1215, 2012.

[6] D. Aleksendric, P. Carlone, and V. Cirovic, "Optimization of the temperature-time curve for the curing process of thermoset matrix composites," *Applied Composite Materials*, vol. 23, pp. 1047–1063, Oct. 2016.

[7] (2020). Partial differential equation toolbox documentation, [Online]. Available: `https://www.mathworks.com/help/pde/index.html?s_tid=CRUX_lftnav` (visited on 03/25/2020).

[8] S. R. S. Inc. (2019). Thermistor calculator v1.1, [Online]. Available: `https://www.thinksrs.com/downloads/programs/therm%5C%20calc/ntccalibrator/ntccalculator.html` (visited on 12/20/2019).

[9] Makeralot. (2019). Reprap hotend thermistor ntc 3950 100k with 1m cable, [Online]. Available: `https://www.makeralot.com/download/Reprap-Hotend-Thermistor-NTC-3950-100K.pdf` (visited on 12/20/2019).

[10] A. Géron, *Hands-On Machine Learning with Scikit-Lean, Keras & TensorFlow*. 1005 Gravenstein Highway Norht, Sebastopol, CA 95472: O'Reilly, 2019, ch. 2.

[11] ——, *Hands-On Machine Learning with Scikit-Lean, Keras & TensorFlow*. 1005 Gravenstein Highway Norht, Sebastopol, CA 95472: O'Reilly, 2019, ch. 12.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, Jun. 2014.

[13] J. Moolayil, *Learn Keras for Deep Neural Networks: A Fast-Track Approach to Modern Deep Learning with Python*. 1005 Gravenstein Highway Norht, Sebastopol, CA 95472: O'Reilly, 2018, ch. 2.

[14] A. Criminisi, "Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning," *Foundations and Trends in Computer Graphics and Vision*, vol. 7, pp. 82–227, Feb. 2012.

[15] H. Daumé III, "Frustratingly easy domain adaptation," in *Conference of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, 2007.

[16] B. Sun, J. Feng, and K. Saenko, "Return of frustratingly easy domain adaptation," *CoRR*, vol. abs/1511.05547, 2015. arXiv: `1511.05547`. [Online]. Available: `http://arxiv.org/abs/1511.05547`.