Internet of Things - SEIS744

Wade Lykkehoy

Student ID: 101217575

[lykk3260@stthomas.edu](mailto:lykk3260@stthomas.edu)

Project Paper

Exploration of an IoT Solution for

Predictive Maintenance of Residential Furnaces

1. Introduction
   1. Motivation

Last December I noticed when my home furnace was starting, it sounded ‘different’. Though it started fine and seemed to be operating normally. There was nothing indicated on the control panel as to a possible issue. I dismissed it thinking perhaps I was not remembering or for some reason just noticing the startup moreso. I would try to pay closer attention and see if the sound changed. This was of course during our first true cold period of the winter.

Very shortly thereafter, the furnace experienced total failure. The soonest I was able to get a service company to come out was 4 days. The issue was a small fan/blower used during the ignition process that ‘self destructed’. Luckily, the service company had one (and only one!) locally in stock. The service tech shared that if they had not had it in stock, it would have taken 1-2 weeks to get.

* 1. State of the Art

In researching home furnace technology, I was surprised at the lack of technology being applied to monitoring and maintenance. Technology application is focused on furnace operating efficiency and ‘ease of use’[1]. I was unable to find any references or description of technology currently being applied to monitoring and predictive maintenance, beyond a simple indicator of total failure.

There are multiple companies who offer maintenance plans and/or tune-up services for home furnaces. One example is CenterPoint Energy[2, 3]. However, the maintenance plans are of a break-fix nature. If the furnace breaks, call the service provider, and a service technician will be sent out to fix/repair it. Tune up plans are based on a period of time, generally 12 month intervals. Neither are based on actual need as determined from the unit itself.

Looking beyond maintenance to unit end-of-life and replacement, criteria for replacement is entirely antidotal / subjective. Following is a consolidated list of ‘criteria’ which I distilled from multiple sources[4, 5, 6]>:

* Is the furnace 15 years or more old?
* Does it break down often or has it broken down in the last 2 years?
* Does it make strange noises?
* Are your energy bills going up?
* Does your home seem warmer or colder?
* Did your carbon monoxide detector go off or do any members of your household experience frequent headaches / burning feeling in the nose or eyes / nausea / flu-like symptoms?
  1. A Better Way

With the current IoT technologies available, one has to ask, “is there a better way?” Can current technologies be used to provide data related to the most common furnace failures? Could we gain insights from this data that enables us to take action before complete failures occur?

That is what this project will explore...

1. Solution Prototype
   1. Scope

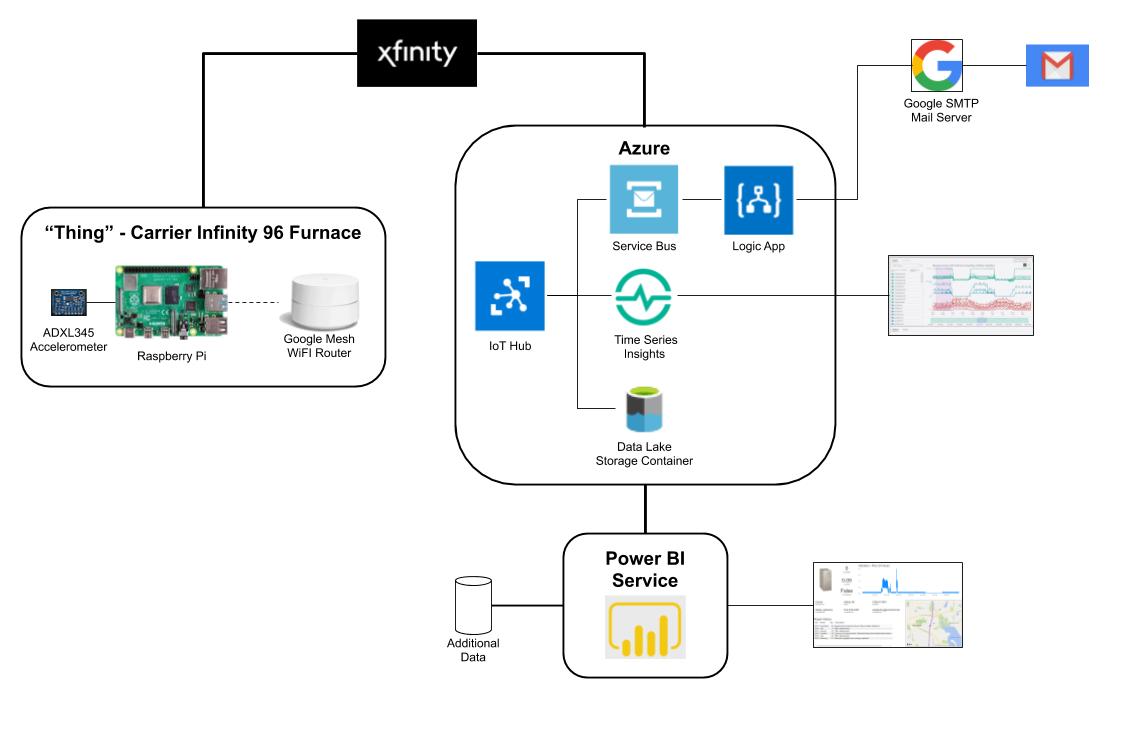
The scope of this project is to explore the technical feasibility of an IoT solution for predictive maintenance of residential furnaces. In researching the most common failures / maintenance items on home furnaces, I found the following general themes/categories[7, 8, 9, 10]:

* Thermostat not working
* Dirty filters
* Electric motor failure; noisy operation
* Out of tune; dirty burners or not adjusted properly
* Ignition failure
* Fuel not getting to unit

I decided to narrow the scope further, to the category most close to my heart, electric motor failure. More specifically, detecting a motor that was making unusual noise due to increase in vibration.

* 1. Architecture

The following diagram depicts the overall physical architecture of the prototype:

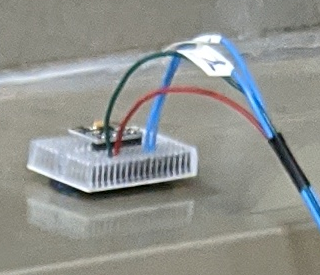


The following sections break down and discuss the architecture in more detail per the 7 Layers of the Internet of Things Technology Stack.

* 1. Things

My “thing” is comprised of 4 main hardware components:

1. Carrier Infinity 96 Gas Furnace
2. Adafruit ADXL345 accelerometer[11, 25]
3. Raspberry Pi 4[12]
4. Google Mesh WiFi Router[13]





The Adafruit ADXL345 is a 3-axis MEMS based accelerometer which supports I2C and SPI interfaces. For this project, I used the I2C interface. Adafruit provides a Python library[14, 15] based on CircuitPython[16] for interfacing with the sensor. With this library, reading data from the sensor takes only a few lines of code:

import board

import busio

import adafruit\_adxl34x

# Initialize the accelerometer

i2c = busio.I2C(board.SCL, board.SDA)

accel = adafruit\_adxl34x.ADXL345(i2c)

accel\_x, accel\_y, accel\_z = accel.acceleration

In addition to acceleration, the library supports tap detection, free-fall detection, and motion detection. For this project, only acceleration is used.

The accelerometer is hard-wired to the Raspberry Pi via a T-hat and breadboard (visible in the photo above). These items merely aid in the wiring, making it easier than plugging wires directly onto the Raspberry Pi interface connector.

The Raspberry Pi is a model 4 with 4 Gig of memory with a 32Gig Micro SD Card acting as its ‘hard drive’. The Raspberry Pi runs the Raspbian operating system, which is a variant of the Debian Linux operating system. I set the Raspberry Pi up to run in a “headless” mode, meaning there is no attached monitor or keyboard. It is accessed from my development pc via WiFi.

See References & Resources 17, 18, 19, 20, 21, 22, and 23 for Raspberry Pi documentation and related information.

The Raspberry Pi is connected via WiFi to my home Google Mesh WiFi Router[13], which provides internet connectivity through my internet provider, Xfinity.

* 1. Connectivity / Edge Computing

Internet connectivity is via a hard-wired connection between my Google Mesh WiFi Router and my internet provider, Xfinity.

I am performing multiple “edge computing” type tasks on the Raspberry Pi:

1. Deriving a value for “vibration” from accelerometer readings

The accelerometer data I am using are measures of acceleration in the X, Y, and Z planes. For this application, I need an indication of vibration. There are multiple ways to transform acceleration into vibration[24]. I chose the simplest method that appeared to meet my needs; consider vibration as a difference in acceleration over time. Thus I take the difference between a reading and the previous reading, and sum the absolute values for the X, Y, and Z planes:

delta\_x = new\_x - prev\_x

delta\_y = new\_y - prev\_y

delta\_z = new\_z - prev\_z

sum\_delta = abs(delta\_x) + abs(delta\_y) + abs(delta\_z)

1. Smoothing the readings via a moving average model

I found the above, while providing a nice indicator of vibration, produced values which tended to be quite “spiky” over time. I added a moving average, which smoothed out the “spiky” aspect yet still provided a good indicator of on/off. After a fair bit of tinkering and experimenting, I landed on 20 sensor readings per second with a 20 reading moving average, thus yielding one data point per second.

1. Determining run state (on/off) based on the vibration value

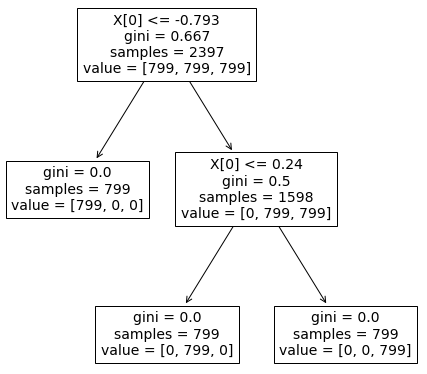
Given there is no baseline vibration data available, I had to determine my own thresholds for the furnace being on or off.

I gathered sample data when the furnace was on and when it was off (see *Appendix B - VibData.py*). Analyzing the data in Microsoft Excel, I could see the two states were easily separable.

I debated whether to just create a simple “if-then-else” decision boundary for determining on/off or use an AI algorithm. Given I am in the Data Science program, I had to opt for the AI approach (and I sense real-world problems are generally more complex where an AI approach would make more sense). I tried 3 AI models, a Support Vector Machine, a Classification Decision Tree, and a Neural Network. The Neural Network, I know is way overkill for this problem, however since I am also currently in the Neural Net class, I had to try it.

The winning algorithm was the Classification Decision Tree (see *Appendix C - DecisionTree\_BuildAndSave.py*). I was surprised the Support Vector Machine did not do as well separating the states as in theory it should maximize the separation, creating the optimal decision boundary. I have not (yet) had bandwidth to delve deeper into understanding why I saw the result did.

Following is a graphical depiction of the decision tree model:



1. Tracking how many seconds the furnace has been in the current state. Which, I am somewhat embarrassed to confess, was more of a “seemed like a good idea at the time” and I did not end up using it.
2. Determining an “unusual” amount of vibration

Determining what is an “unusual” amount of vibration is of course a challenge given there is no baseline data available. Typically, one would employ formal anomaly detection methods. Here, I admittingly “cheated” a bit and used a simple ‘if vibration > threshold then vibration\_anomaly is true’ approach. I set a threshold that was well above normal operation and that I found I was able to easily exceed by placing my running Dremel tool near the sensor:



At the completion of the above edge computing tasks, a message is formatted in JSON and sent to the Microsoft Azure IoT Hub via a Python Azure SDK/library[28, 29]. Sample message:

{

"body": {

"dev\_id": "RazPi",

"ts": "2020-04-17T18:28:13z",

"vib": 0.07649187,

"state": 0,

"sis": 17146,

"vibalert": "False"

},

"enqueuedTime": "2020-04-17T18:28:13.701Z",

"properties": {

"vibrationAlert": "False"

}

}

Source code is attached in *Appendix D - FurnaceMonitor.py*

* 1. Global Infrastructure

I chose to use the Microsoft Azure cloud infrastructure. I debated between AWS and Azure. I have used AWS in the past for class work at St Thomas, although not the IoT specific capabilities. I have never used Azure. I saw this as an opportunity to expand my cloud knowledge and hence chose Azure.

* 1. Platforms Data Ingestion

The messages come into the Azure infrastructure via an IoT Hub[30]. IoT Hub is a cloud based service which acts as a central message hub for both device-to-cloud and cloud-to-device communications (in this project, I only used the device-to-cloud capabilities). It provides support for messaging, file uploading, and control of devices. In addition, it has high-level monitoring of the devices and communications. I found it relatively easy to connect and send messages from the Raspberry Pi via the Python SDK/library.

You can configure the IoT Hub to pass the messages along to many different destinations, called “endpoints”. A single message may be passed on to endpoints either unconditionally or conditionally, which may be based on data within the message. I used this capability to send e-mail when a high vibration alert was present in the message (see *Applications* below).

I configured three message destinations/endpoints:

1. Data Lake Storage; all messages from my “thing”
2. Time Series Insights; all messages from my “thing”
3. Service Bus; only messages from my “thing” that indicate a high vibration alert

For data storage, Azure offers several types of storage capability[26]. My data is non-relational, semi-structured, large volume (relatively anyways), and intended for analytics processing, thus I chose Azure Data Lake Store / Azure Blob Store[27].

* 1. Data Analysis

The Azure IoT environment provides a data analysis tool specifically for time-series based IoT data, Time Series Insights[31, 32]. This is a relatively new tool and is currently in “preview”.

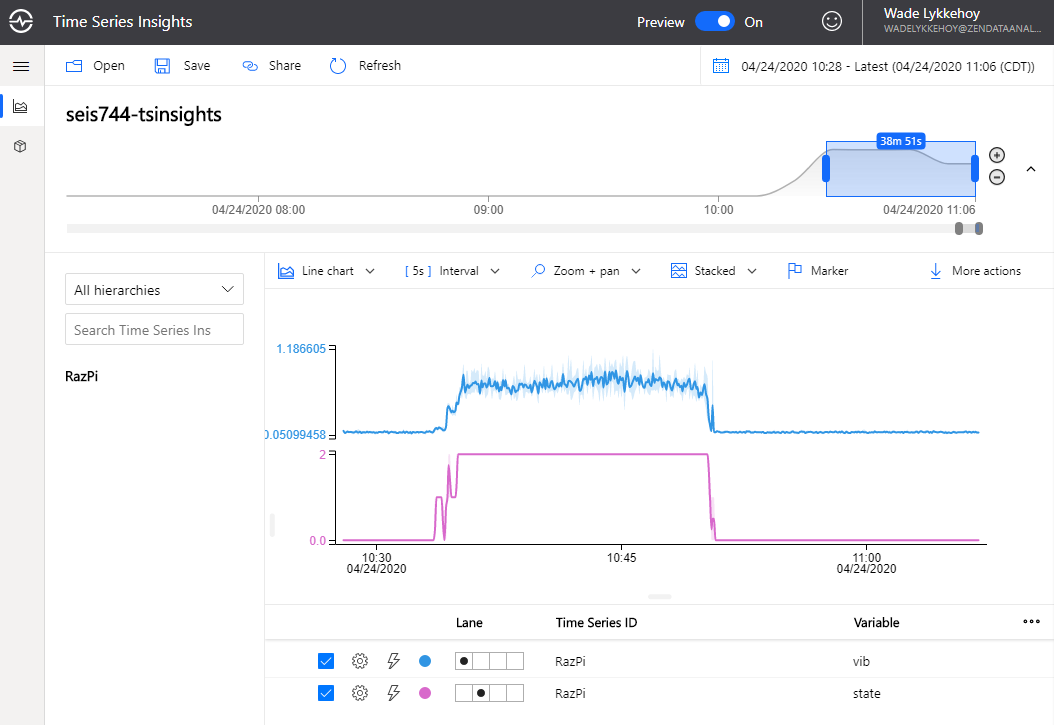
I was quite impressed with the capabilities of the tool, especially after having spent hours trying to do comparable analysis in Power BI. A high-level list of key capabilities:

* Choose between line graphs, tables, or heatmaps to view data
* Filter to specific windows of dates/times via graphical sliders
* Subset which data to display by device or any data within the messages
* Download data in CSV format

A couple caveats to be aware of:

* It is a PaaS tool, thus when you spin one up, it behind the scenes spins up server(s) and additional storage. This can add significant cost to a solution to the tune of $150 per month or more. Although there is an experimental pay-as-you-go option which, if one is very careful and only spin it up when you need it and use it, could be more cost effective.
* It uses its own storage, thus can only view data sent to it after spinning it up. You cannot look at historical data which was created prior to spinning it up.
* As noted above, it is still in preview, thus one can expect it can and will change quite a bit.

Following is a screen-shot with my “thing” data displayed:



* 1. Applications

I created two exploratory applications:

1. An application to send end an email when a high vibration alert is contained in the “thing” message
2. A dashboard with information about my furnace and current monitoring data
3. Email application

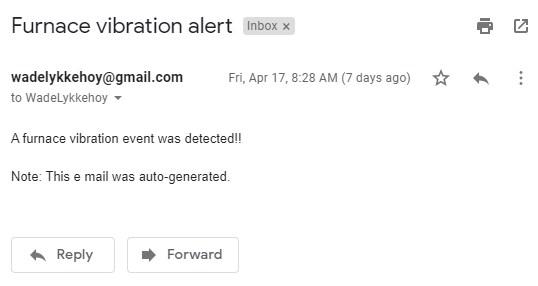
Within the “thing” message properties is an indicator that high / unusual vibration has been detected:

"properties": {

"vibrationAlert": "True"

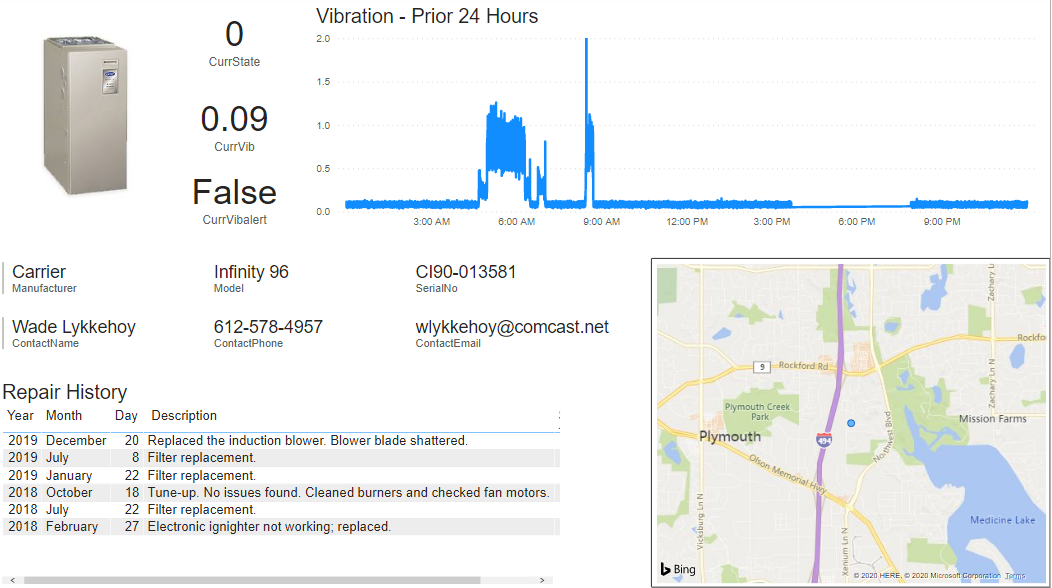
}

When the IoT Hub receives a message where this property has value of “True”, it forwards the message on to a Service Bus. The Service Bus is a message FIFO message queue from which applications can retrieve messages. I created a Logic App[33] which checks the Service Bus every minute for messages. If there is one, it leverages Google’s SMTP Mail Server to send an email (in this case to myself). Following is an example of the received email.



1. Dashboard

I created a simple dashboard to explore and illustrate how one might leverage the data from my “thing”, augment it with additional data, and provide at-a-glance information about my furnace. For additional data, I added general information including furnace location, home owner contact information, plus repair history. For this project, I stored the additional information in a set of CSV files. A screenshot of the dashboard:



* 1. People and Process

With the scope of this project to explore the technical feasibility of an IoT solution for predictive maintenance of residential furnaces, I can only speculate how decision making and processes might be transformed:

* As a home-owner
  + Having quantitative data rather than qualitative (i.e. it sounds different) will enable you to seek repair rather than waiting until total failure (as I did).
  + Knowing when the furnace needs a tune-up rather than basing on a regular schedule (e.g. every 12 months) could result in cost savings via either less frequent tune-ups or better/proper operation.
* As a furnace repair provider
  + Ability to take early action when there is a developing issue enables preventing total failure and inconvenience for the homeowner (the middle of winter with an inoperable furnace is not pleasant).
  + Having data on what the root cause of an issue is prior to making a service call enables the service technician to be better prepared. This could reduce most issues from two service calls (one to diagnose the problem and one to fix it) to a single service call. Result is higher customer (home owner) satisfaction and cost savings which could be passed along to the customer.
  + Savings on annual maintenance plans from knowing when a furnace really needs a tune-up vs a regular schedule. Even a 1 month extension (i.e. 13 months vs 12) would be a little over 8% reduction in maintenance activity costs.
  + You could offer an as-a-service model, where you provide “HVAC as a service” to homeowners with a high SLA.

1. Challenges & Lessons Learned

There were many challenges and learnings from the project. Often I felt as though I was trying to push a loaded semi-truck up a very long hill. Following are what I would consider the key / high-level challenges & lessons learned.

1. Complexity of the tech stack. Following is a list of the key tools and technologies in my tech stack:

* Raspberry Pi
  + Raspberry Pi / Raspbian operating system
  + VNC Server
  + Thonny Python IDE on the Raspberry Pi
  + Adafruit sensor Python library
  + Azure IoT Python library
  + Python 3
  + Scikit-learn
* Development desktop
  + MS Windows
  + SD Card Formatter
  + Putty
  + WinSCP
  + VNC Viewer
  + Git / GitHub
  + Python 3
  + Scikit-learn
  + Excel
  + Azure IoT Explorer
  + Azure Storage Explorer
  + Power BI Desktop
* Azure Cloud
  + Azure Portal
  + Azure IoT Hub
  + Azure Data Lake Storage
  + Azure Time Series Insights
  + Azure Service Bus
  + Azure Logic App

In looking at the above, there are three very different platforms I was working across; an ARM based system with Linux, an Intel based system with Windows, and a system that is somewhat of a hybrid between Linux and Windows (the Azure Cloud). I found when I developed code or an ML model on my desktop, I could not always directly transfer it to the Raspberry Pi. I had to re-create / re-compile / re-something on the other platform. Shell scripts are of course specific to the operating system.

Python is a powerful and flexible environment with libraries available to do almost anything you can imagine. The downside is managing all of the libraries and their specific versions can become challenging.

1. The amount of data generated adds up quickly. I initially thought of pushing the raw data readings at 20 per second to the Azure cloud. After calculating the volume of data that would result in, I quickly re-thought that approach and went with one data point per second. Even with that lower rate, I am generating about 55 Meg of data per day.
2. I naively assumed the furnace was either on or off. After collecting initial data I realized it is more complex than that. In researching furnace operation, I found there is a ‘startup’ state where a smaller blower runs to build up airflow prior to burner ignition[34]. Upon collecting more data, I noticed there is also a distinct ‘shutdown’ state. This is where domain expertise would certainly have been valuable!
3. Measuring vibration is non-trivial

* There are multiple ways to turn accelerometer readings into an indication of vibration.
* Readings can be noisy, thus methods to deal with this noise is important.
* Placement of the sensor is very important to get quality data. Not wanting to place the sensor inside the furnace (I would have been kicked out of the house had I caused any issue with the furnace in February), I tested various locations on the outside.
* Sensor interference from other sources must be considered. I found that people walking above the furnace plus the clothes washer/dryer running affected the vibration readings.

1. Determining thresholds for knowing if the furnace is on or off plus what level of vibration would be considered abnormal is challenging without some baseline to work from. I collected data for how the furnace is operating currently and created thresholds from that. It could very well be the furnace is currently operating abnormally and my model would hence never predict abnormal behavior (or only would if the furnace was literally about to blow up).
2. Documentation for various technologies/libraries/etc can be sparse, non-existent, or out of date. I particularly found this true with some of the Azure components as they are evolving and changing rapidly. Some articles and how-to’s plain did not work or the UI had changed making them difficult to follow or understand.
3. The Azure ecosystem is very large and complex. Determining which parts and pieces to use was non-trivial. Plus there are often multiple ways to accomplish any specific task. I found I had to experiment with various alternatives to see which was best for my purpose.
4. Azure IoT had some surprises in store:

* Azure IoT Hub adds significantly to your “thing” message. It adds device information, a timestamp, and a long list of various properties. My actual device message only accounted for 20% of the IoT Hub message; 80% was added information. One must take this into account when calculating storage needs and bandwidth for retrieving messages.
* For whatever reason, when the IoT Hub sends the messages to Data Lake Storage, it base 64 encodes the “thing” part of the message. Thus when you retrieve the messages for reporting, etc, you must decode it prior to being able to use its data. There currently appears to be no way of controlling this behavior; there is a request logged to allow you to turn that off.
* The Azure Data Lake storage partitions your data into multiple files; one file per minute of data. Hence you end up with LOTS of small files. You have no control over this behavior and cannot define your own partitioning scheme.

1. Deciding how much edge processing to do vs cloud processing. I ended up doing more processing on the Raspberry Pi than originally planned as at that point in the project I was less familiar with Azure plus the cost on Azure would have been greater.
2. To avoid ending up with extraneous “junk” data, it is essential to start with the end objective and work backwards to identify what data you really need to accomplish that. It is easy to fall into the “maybe someday someone will find this piece of data useful; let’s store it too” trap.

Many of the above we discussed in class as ‘typical’ for IoT projects and what make IoT projects more complex and challenging than many/most other types of technology projects.

1. Conclusion

The primary objective of this project was to explore the technical feasibility of an IoT solution for predictive maintenance for residential furnaces. Narrowing the scope further, I explored vibration detection. I believe this project shows that it is indeed very feasible to create such an IoT solution.

Expanding the solution to include additional contributors or indicators of furnace failure could be accomplished through additional sensors, such as a CO2 sensor, plus additional analysis of data, looking for unusual patterns. More strategic placement of sensors would enable higher quality of data to be collected, leading to higher quality of information, hence higher quality of insights.

Such a solution opens the door to enhanced “service centric” business models. Companies offering service contracts could move beyond the break-fix or annual models to a model where service is provided when needed. Both the customer and service company could see reduced cost. Customer satisfaction would increase due to less interruption from eliminating multiple service calls to fix an issue and from greatly reduced outages.

Taking this to the next level, a company could offer HVAC as-a-service. In such a model, the homeowner would not own the HVAC equipment, rather, it would be owned by the HVAC as-a-service company. The homeowner would pay a periodic (e.g. monthly) fee. For this fee, the homeowner never has to worry about their furnace and very rarely, if ever, experience any outages. I personally think this would be fascinating to explore further in both the residential and commercial spaces.

**Acknowledgments**

A thanks to Logan Butler, who early in the project pointed me in the direction of the Adafruit Python libraries for reading data from sensors. Using these libraries saved me countless hours of “pain and agony”.

A thanks to my family for putting up with me turning the furnace on and off at various times throughout February/March without giving them a heads-up.

**References & Resources**

1. Infinity 98 gas furnace with Greenspeed - 59MN7 | Carrier - Home comfort. (n.d.). Retrieved April 15, 2020, from <https://www.carrier.com/residential/en/us/products/gas-furnaces/59mn7/>
2. Home appliance repair & service plans | CenterPoint energy home service plus. (n.d.). Retrieved April 15, 2020, from <https://www.centerpointenergy.com/en-us/home-service-plus/plans/basic-repair-plan>
3. Furnace tune-up & boiler tune-up | CenterPoint energy home service plus. (n.d.). Retrieved April 15, 2020, from <https://www.centerpointenergy.com/en-us/home-service-plus/services/maintenance-services/tune-ups/furnace-tune-up>
4. How long does a furnace last & how do I know when to replace it?. (2020, March 9). Retrieved April 15, 2020, from <https://www.sedgwickheating.com/long-furnace-last-know-replace/>
5. Is it time for a new furnace? Signs your furnace is failing. (n.d.). Retrieved April 15, 2020, from <https://www.centerpointenergy.com/en-us/HSP/Pages/HSP-Experts/Time-For-New-Furnace.aspx?sa=MN&au=res>
6. When is it time to replace your oil or natural gas furnace | Petro. (n.d.). Retrieved April 15, 2020, from <https://www.petro.com/heating/is-it-time-for-a-new-furnace>
7. Spezza, T. (2019, October 7). What are the most common furnace problems?. Retrieved April 16, 2020, from <https://www.aaaheatingandcoolinginc.com/10-common-furnace-problems-repairs/>
8. Top 10 causes of furnace failures | Action furnace. (n.d.). Retrieved April 16, 2020, from <https://actionfurnace.ca/education-centre/top-10-causes-of-furnace-failure/>
9. Top five causes of furnace failures. (2015, April 22). Retrieved April 16, 2020, from <https://thewirenut.com/education-center/top-five-causes-of-furnace-failures/>
10. Understanding common furnace problems. (n.d.). Retrieved April 16, 2020, from <https://www.pippinbrothers.com/blog/article/understanding_common_furnace_problems>
11. ADXL345 digital accelerometer. (2013, March 26). Retrieved April 17, 2020, from <https://learn.adafruit.com/adxl345-digital-accelerometer/overview>
12. Buy a raspberry Pi 4 model B – Raspberry Pi. (n.d.). Retrieved April 17, 2020, from <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>
13. Nest WiFi. (n.d.). Retrieved April 17, 2020, from <https://store.google.com/us/product/nest_wifi>
14. Adafruit-circuitpython-adxl34x. (n.d.). Retrieved April 17, 2020, from <https://pypi.org/project/adafruit-circuitpython-adxl34x/>
15. Raspberry Pi accelerometer using the ADXL345. (2019, November 16). Retrieved April 17, 2020, from <https://pimylifeup.com/raspberry-pi-accelerometer-adxl345/>
16. CircuitPython on Linux and raspberry Pi. (2018, June 30). Retrieved April 17, 2020, from <https://learn.adafruit.com/circuitpython-on-raspberrypi-linux>
17. Linux. (n.d.). Retrieved April 17, 2020, from <https://www.raspberrypi.org/documentation/linux/>
18. Setting up your Raspberry Pi (n.d.). Retrieved April 17, 2020, from <https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up>
19. Using your Raspberry Pi (n.d.). Retrieved April 17, 2020, from <https://projects.raspberrypi.org/en/projects/raspberry-pi-using>
20. Raspberry Pi documentation. (n.d.). Retrieved April 17, 2020, from <https://www.raspberrypi.org/documentation/>
21. Remote access. (n.d.). Retrieved April 17, 2020, from <https://www.raspberrypi.org/documentation/remote-access/>
22. Teach, learn, and make with raspberry Pi – Raspberry Pi. (n.d.). Retrieved April 17, 2020, from <https://www.raspberrypi.org/>
23. PiMyLifeUp (n.d.). Retrieved April 17, 2020, from <https://pimylifeup.com/>
24. Arduino forum. (n.d.). Retrieved April 17, 2020, from [https://forum.arduino.cc/index.php?topic=273086.0](https://forum.arduino.cc/index.php)
25. Types of vibration sensor and how to buy one for Arduino?. (2020, January 7). Retrieved April 17, 2020, from <https://www.seeedstudio.com/blog/2020/01/07/vibration-sensors-types-how-to-use-with-arduino-and-raspberry-pi/>
26. Azure data storage options: Know where to store your data in Azure. (2020, March 11). Retrieved April 17, 2020, from <https://stackify.com/store-data-azure-understand-azure-data-storage-options/>
27. Normesta. (n.d.). Azure data lake storage Gen2 introduction. Retrieved April 17, 2020, from <https://docs.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-introduction>
28. Azure/azure-IoT-sdk-python. (2020, April 15). Retrieved April 17, 2020, from <https://github.com/Azure/azure-iot-sdk-python>
29. Wesmc7777. (n.d.). Send telemetry to Azure IoT hub quickstart (Python). Retrieved April 17, 2020, from <https://docs.microsoft.com/en-us/azure/iot-hub/quickstart-send-telemetry-python>
30. Robinsh. (n.d.). Azure IoT hub documentation. Retrieved April 17, 2020, from <https://docs.microsoft.com/en-us/azure/iot-hub/>
31. Deepakpalled. (n.d.). Quickstart: Azure time series insights explorer - Azure time series insights. Retrieved April 17, 2020, from <https://docs.microsoft.com/en-us/azure/time-series-insights/time-series-quickstart>
32. Deepakpalled. (n.d.). Tutorial: Create an environment - Azure time series insights. Retrieved April 17, 2020, from <https://docs.microsoft.com/en-us/azure/time-series-insights/tutorial-create-populate-tsi-environment>
33. Robinsh. (n.d.). IoT remote monitoring and notifications with Azure logic apps. Retrieved April 17, 2020, from <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-monitoring-notifications-with-azure-logic-apps>
34. Sequence of operation – Gas furnaces. (2017, October 30). Retrieved April 20, 2020, from <https://heatingcooling.fergusonhvac.com/mid-south/2017/10/30/sequence-of-operation-gas-furnaces/>

**Appendix A - GitHub Repository**

All code, data files used by the code, this document, and the project presentation may be found in the following GitHub repository:

<https://github.com/wlykkehoy/SEIS744_Project>

**Appendix B - VibData.py**

import time

import collections

import statistics

import board

import busio

import adafruit\_adxl34x

# Change this to flag the state the furnace is in

FURNACE\_STATE = 0 # 0 => off; 1 => startup/shutdown; 2 => on

# Some constant values; change for experimenting

READINGS\_PER\_SECOND = 20

RUNNING\_AVG\_WINDOW\_SIZE = 20

NUM\_CASES\_TO\_COLLECT = 1000

sleep\_time = 1.0 / READINGS\_PER\_SECOND

# Initialize the accelerometer

i2c = busio.I2C(board.SCL, board.SDA)

accel = adafruit\_adxl34x.ADXL345(i2c)

# Do a running average of the readings

running\_avg\_window = collections.deque(maxlen=RUNNING\_AVG\_WINDOW\_SIZE)

# Initialize previous reading variables, else will start off with undefined deltas

prev\_x, prev\_y, prev\_z = accel.acceleration

time.sleep(sleep\_time)

# Loop and dump readings to stdout

case\_count = 0

while case\_count < NUM\_CASES\_TO\_COLLECT:

new\_x, new\_y, new\_z = accel.acceleration

delta\_x = new\_x - prev\_x

delta\_y = new\_y - prev\_y

delta\_z = new\_z - prev\_z

sum\_delta = abs(delta\_x) + abs(delta\_y) + abs(delta\_z)

running\_avg\_window.append(sum\_delta)

running\_avg = statistics.mean(running\_avg\_window)

print('{}, {:>10.6f}'.format(FURNACE\_STATE, running\_avg))

case\_count += 1

prev\_x = new\_x

prev\_y = new\_y

prev\_z = new\_z

time.sleep(sleep\_time)

**Appendix C - DecTree\_BuildAndSave.py**

# ### Decision Tree Model

#

# Looking at plots of the data in Excel, it looks like the classes have clear separation.

# Let's start with a Decision Tree Classifier model and see how that does.

# Import libraries we will be using

import sys

import numpy as np

import pandas as pd

import sklearn as skl

import sklearn.metrics as skl\_met

import sklearn.model\_selection as skl\_modsel

import sklearn.preprocessing as skl\_pre

import sklearn.tree as skl\_tree

import sklearn.pipeline as skl\_pipeline

import pickle

# Always a good idea to dump verions of key libraries

print('python version: ', sys.version)

print('numpy version: ', np.\_\_version\_\_)

print('pandas version: ', pd.\_\_version\_\_)

print('sklearn version: ', skl.\_\_version\_\_)

# I read it is a good idea to seed the Numpy random number generator

# so that we get the same 'random' numbers each run for reproducibility.

np.random.seed(123)

# Loaad the dataset

dataset = pd.read\_csv('./ML\_20.csv', header=None, names=['status', 'vibration'])

# Dump some general stats

print('Number of records: {}\n'.format(dataset.size))

# Dump a few records from each 'section'; there should be 999 record with status=0,

# followed by 999 records with status=2, and 999 records with status=1

print(dataset[0:5], '\n')

print(dataset[1000:1005], '\n')

print(dataset[2000:2005], '\n')

# For a categorical values, print count and % of each class

counts = dataset['status'].value\_counts().sort\_index()

print('Class Count % of Total')

for idx, val in counts.items():

print(' {0} {1:6d} {2:6.2f}%'.format(idx, val, ((val / dataset.shape[0]) \* 100)))

# Split off the the features and target (note we only have 1 feature)

# Since we constructed the data to have status values 0-2, we can just

# use those values as the class labels without any further fuddling (e.g. label encoding)

X = dataset.loc[:, 'vibration']

y\_class = dataset.loc[:, 'status']

# Since we only have 1 feature, the above will generate a 1-dim array; we need

# a 2-dim array with only 1 column; so need to add an extra dimension, the columns

X = np.expand\_dims(X, 1)

# Split into training and test

X\_train, X\_test, y\_train\_class, y\_test\_class = skl\_modsel.train\_test\_split(X, y\_class,

test\_size=0.2, random\_state=0, stratify=y\_class)

print("X.shape =", X.shape)

print("X\_train.shape =", X\_train.shape)

print("X\_test.shape =", X\_test.shape)

print("y\_train\_class.shape =", y\_train\_class.shape)

print("y\_test\_class.shape =", y\_test\_class.shape)

print('\n')

print('Target class counts - train:')

cname, count = np.unique(y\_train\_class, return\_counts=True)

for ele in zip(cname, count):

print(' Class: {} Count: {}'.format(ele[0], ele[1]))

print('\n')

print('Target class distribution - test:')

cname, count = np.unique(y\_test\_class, return\_counts=True)

for ele in zip(cname, count):

print(' Class: {} Count: {}'.format(ele[0], ele[1]))

# let's use the standard scaler to normalize feature values

sc\_X = skl\_pre.StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

# At long last, we can create a model

tree\_model = skl\_tree.DecisionTreeClassifier() # kernel='linear')

tree\_model.fit(X\_train, y\_train\_class)

# Using the test set, let's see how well our model does

yhat\_test\_class = tree\_model.predict(X\_test)

print('Confusion Matrix - Test:\n', skl\_met.confusion\_matrix(y\_test\_class, yhat\_test\_class))

print('\n')

print('Precision - Test:', skl\_met.precision\_score(y\_test\_class, yhat\_test\_class,

average=None))

print('Recall - Test :', skl\_met.recall\_score(y\_test\_class, yhat\_test\_class, average=None))

print('F1 - Test :', skl\_met.f1\_score(y\_test\_class, yhat\_test\_class, average=None))

# Let's also see how the model did on the training set

yhat\_train\_class = tree\_model.predict(X\_train)

print('Confusion Matrix - Train:\n', skl\_met.confusion\_matrix(y\_train\_class,

yhat\_train\_class))

print('\n')

print('Precision - Train:', skl\_met.precision\_score(y\_train\_class, yhat\_train\_class,

average=None))

print('Recall - Train :', skl\_met.recall\_score(y\_train\_class, yhat\_train\_class,

average=None))

print('F1 - Train :', skl\_met.f1\_score(y\_train\_class, yhat\_train\_class,

average=None))

# Let's package the data standardization and decision tree into a pipeline

pipeline = skl\_pipeline.make\_pipeline(sc\_X, tree\_model)

# Let's test the pipeline with some randomly selected values from the

# original data file; this should predict classes 0, 2, 1

X\_pipeline\_test = [[0.074531], [0.847295], [0.194172], [0.203978], [0.074531], [0.786493]]

y\_pipeline\_test\_class = [0, 2, 1, 1, 0, 2]

yhat\_pipeline\_test\_class = pipeline.predict(X\_pipeline\_test)

print('Confusion Matrix - Pipeline Test:\n', skl\_met.confusion\_matrix(y\_pipeline\_test\_class, yhat\_pipeline\_test\_class))

# Save the model off

pickle.dump(pipeline, open('tree\_pipeline.pkl', 'wb'))

**Appendix D - FurnaceMonitor.py**

# Import libraries that we will be using

import sys

import getopt

import time

import collections

import statistics

import numpy as np

import pandas as pd

import pickle

import board

import busio

import adafruit\_adxl34x

from azure.iot.device import IoTHubDeviceClient, Message

# Some constant values; change these to change behavior of the app

DEVICE\_ID = 'RazPi' # name for the d evice; used in Azure machinery

READINGS\_PER\_SECOND = 20

RUNNING\_VIB\_WINDOW\_SIZE = 20 # num readings to average over for our vibration calc; 20 => a 1 second window

RUNNING\_VIB\_ALERT\_WINDOW\_SIZE = 40 # num readings to average over for our vibration alert; 40 => 2 second window

RUNNING\_STATE\_WINDOW\_SIZE = 100 # num readings to consider for determining current state; 100 => a 5 second window

VIB\_ALERT\_THRESHOLD = 2.0 # vibration threshold for generating an alert; would be

# better to do an anomaly detection algorithm;

# maybe later...

READINGS\_PER\_IOT\_HUB\_MSG = 20 # Thus will be pumping 1 message/sec to IoT Hub

IOT\_HUB\_CONNECTION\_STRING = "HostName=seis744-project-hub.azure-devices.net;DeviceId=razpi;SharedAccessKey=xxxx"

MSG\_TEMPLATE = '{{"dev\_id":"{dev\_id}","ts":"{ts\_data}","vib":{vib\_data:10.8f},"state":{state\_data},"sis":{seconds\_in\_state},"vibalert":"{vib\_alert}"}}'

def main\_loop(echo\_messages, echo\_sum\_delta, echo\_running\_window):

# Create an IoT Hub client

client = IoTHubDeviceClient.create\_from\_connection\_string(IOT\_HUB\_CONNECTION\_STRING)

# Initialize the accelerometer

i2c = busio.I2C(board.SCL, board.SDA)

accel = adafruit\_adxl34x.ADXL345(i2c)

# Load the AI model to determine (predict) furnace state

state\_pred\_model = pickle.load(open('tree\_pipeline.pkl', 'rb'))

# Calc the amount of time (sec) to sleep between readings

sleep\_time = 1.0 / READINGS\_PER\_SECOND

# We will be doing a running average of the readings over a given window size

running\_vib\_window = collections.deque(maxlen=RUNNING\_VIB\_WINDOW\_SIZE)

# Plus a running average of the readings over a given window size for the vibration alert

running\_vib\_alert\_window = collections.deque(maxlen=RUNNING\_VIB\_ALERT\_WINDOW\_SIZE)

# And also doing a window over the predicted state, taking the most predicted value

running\_state\_window = collections.deque(maxlen=RUNNING\_STATE\_WINDOW\_SIZE)

# We will be tracking state changes and time the state changed

current\_state = -99 # an invalid value

state\_change\_time = 0

# Initilize prevoius reading variables, else will start off with undefined or zero deltas

prev\_x, prev\_y, prev\_z = accel.acceleration

time.sleep(sleep\_time)

# Enter an infinite loop taking readings, determining state, and pushing data

# up to the Azure IoT Hub

reading\_count = 0

try:

while True:

# Take an accelerometer reading

#reading\_time = time.gmtime()

reading\_time = time.time()

new\_x, new\_y, new\_z = accel.acceleration

reading\_count += 1

# Turn the accelerometer reading into an indicator of vibration

delta\_x = new\_x - prev\_x

delta\_y = new\_y - prev\_y

delta\_z = new\_z - prev\_z

sum\_delta = abs(delta\_x) + abs(delta\_y) + abs(delta\_z)

if (echo\_sum\_delta):

print('{:>10.6f}'.format(sum\_delta), flush=True)

# Find the running average for the vibration

running\_vib\_window.append(sum\_delta)

vib\_avg = statistics.mean(running\_vib\_window)

# Find the running average for the vibration alert & determine alert status

running\_vib\_alert\_window.append(sum\_delta)

vib\_alert = statistics.mean(running\_vib\_window) > VIB\_ALERT\_THRESHOLD

# Use our AI model to determine (predict) the state, using the most predicted

# state over a running window as the 'true' current state

X = np.array([[vib\_avg]])

pred\_state = state\_pred\_model.predict(X)

running\_state\_window.append(pred\_state)

state\_vals, state\_counts = np.unique(running\_state\_window, return\_counts=True)

state = state\_vals[np.argmax(state\_counts)]

if (echo\_running\_window):

print('{}, {:>10.6f}'.format(state, vib\_avg), flush=True)

# If there is a state change, update vars

if (state != current\_state):

current\_state = state

state\_change\_time = reading\_time

if (reading\_count == READINGS\_PER\_IOT\_HUB\_MSG):

# Format the data into JSON

formatted\_time = time.strftime('%Y-%m-%dT%H:%M:%Sz',

time.gmtime(reading\_time))

seconds\_in\_state = int(reading\_time - state\_change\_time)

msg\_text = MSG\_TEMPLATE.format(ts\_data=formatted\_time, vib\_data=vib\_avg,

state\_data=state,

seconds\_in\_state=seconds\_in\_state ,

vib\_alert=vib\_alert, dev\_id=DEVICE\_ID)

if (echo\_messages):

print(msg\_text, flush=True)

# Package the message up and send on to the Azure IoT Hub

msg = Message(msg\_text)

# Add a custom application property to the message to trigger an alert email.

if vib\_alert:

msg.custom\_properties['vibrationAlert'] = "True"

else:

msg.custom\_properties['vibrationAlert'] = "False"

# Send 'er on up to the IoT Hub

client.send\_message(msg)

reading\_count = 0

# Get ready for next reading

prev\_x = new\_x

prev\_y = new\_y

prev\_z = new\_z

time.sleep(sleep\_time)

except KeyboardInterrupt:

# Just fall back to the main

print('... Ctrl-C detected ...', flush=True)

if (\_\_name\_\_ == '\_\_main\_\_'):

# Initialize some flags to control output

echo\_messages = False

echo\_sum\_delta = False

echo\_running\_window = False

# Pull off command line args & set flags

try:

args, \_ = getopt.getopt(sys.argv[1:], 'msr')

except getopt.GetoptError:

print('Usage: {} [-m] [-s] [-r]'.format(sys.argv[0]))

sys.exit(2)

for opt, \_ in args:

if (opt == '-m'):

echo\_messages = True

elif (opt == '-s'):

echo\_sum\_delta = True

elif (opt == '-r'):

echo\_running\_window = True

print ( "FurnaceMonitor started...", flush=True )

print ( "Press Ctrl-C to stop", flush=True )

main\_loop(echo\_messages, echo\_sum\_delta, echo\_running\_window)

print ( "FurnaceMonitor stopped", flush=True )