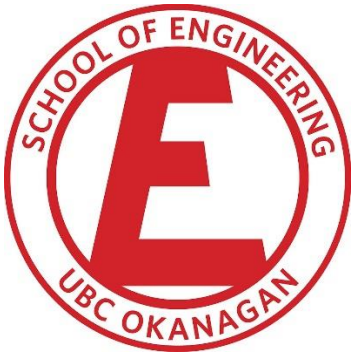
ENGR499 Group 61: Final Project Report



Distributed Sensor Network for Production System Monitoring and Control

Group 61 Members

Jared Paull 63586572

Miguel Villarreal 41613910

Garrett Birch 74130337

Sol Thiessen 85823037

Jonathan Lake 43265818

April 15, 2021

Table of Contents

Table of Figures

Table of Tables

Project Introduction

The world is ripe with operating manufacturing facilities that are not implementing modern technology. Oftentimes, these existing manufacturing production systems would require enormous cost to revamp with modern industry 4.0 specifications. As such, these plants may be missing out on unrealized productivity that modern technology could bring. This project aimed to design sensor nodes in a low-power-wide-area wireless network (LPWAN) which can tie into the existing plant control systems to provide additional information. This information could then be used for monitoring, preventive maintenance, or tie into control systems based on the applications. LPWANs are highly efficient and send information at low data rates, which can allow nodes to run on battery for tens of years [1]. LPWANs also have the capacity to connect with many nodes at once over a long range [1]. Thus, LPWANs running cost are less than traditional alternatives such as Bluetooth or Wi-Fi which have lower range and are less energy efficient. A higher range means that a facility would likely only need one or two gateways to send sensor information to the internet, while a longer battery life requires less maintenance hence cost savings.

The sensor nodes should be able to sense changes in the manufacturing process, then map those changes to real world events. Each node will be able to do all processing onboard, this makes integration of the nodes as simple as possible. Each node will be able to share the data it gathers through an LPWAN and make it widely accessible. Existing control systems should be able to read the sensor node data and be able to make changes to the manufacturing process in real time. These remote nodes can then be placed alongside the existing manufacturing process in order to gain additional data that can improve the existing architecture. The nodes will serve as an easy to install sensor that can provide information beyond the current systems. All while being done without the need to overhaul the entire sensor network; the nodes can simply be installed to the desired location and turned on. Having these sensor nodes put in well thought out locations across the manufacturing production system may result in a more efficient running operation, while costing very little money to install. Our project statement for the project is as follows:

A way to implement modern sensor and data processing techniques in aging manufacturing facilities to improve plant efficiency.

Need and Constraint Identification

Project Need

Manufacturing production systems strive to be as efficient as possible, but plants built before the rise of Internet-of-Things (IoT) do not have the equipment to implement modern data processing techniques. The barrier to entry to IoT implementation is the enormous cost associated with overhauling the entire production sensor network. Instead, by inserting sensor nodes in key areas of the production system, some benefits of IoT can be gained for a fraction of the cost. These sensor nodes can do additional monitoring, or for data collection to be further analyzed to optimize the system. The nodes being designed are not intended to be a substitute to existing sensors, these existing sensors are typically more specialized. In contrast, the sensor nodes being designed should be very flexible, the sensing and processing techniques should be applicable to a large array of objects. This makes the sensor nodes very viable in existing facilities since the nodes are equipped for a wide range of sensing applications. An analysis of the current solution landscape also shows a lack of full integration from sensor to network to a control system, especially when considering low power technology.

Constraint Identification

The scope of our project will be constrained by predetermined factors such as resources, budget, knowledge, and time. The timeline of the project is set by Dr. Kenneth Chau who has set a final due date of April 15, 2022. Throughout this time frame our group is expected to complete several checkpoint deadlines such as the submission of this project definition report due on November 12th, a conceptual design presentation held throughout the weeks of November 15th to December 6th, an oral poster presentation held on April 11th, 2022, and the final report due on April 15th, 2022, to end the project. Further adding to our time constraints, our group's hands-on time with the smaller scale production equipment we will use to experiment our sensors on, will be limited by the availability of the lab, EME 2220, and our group's access to it. As well, the time our group members can dedicate to this project can also be considered.

While the group possesses some initial knowledge about components involved in the project, individually our group members will still have to research certain topics in order to fill in

potential gaps which may arise as we progress into the complexities of our solution. Therefore, our previous technical knowledge can also be considered a constraint affecting our scope. The initial resources provided by our faculty advisor Dr. Dean Richert included: a microcontroller, a transmitter, an auditory microphone and a gateway used to establish connection. Outside of these components, any extra resources needed to be purchased using the $300 budget provided by the School of Engineering.

A further constraint of our project is to keep in mind the low power aspect goal of our design and as such, our group is tasked with designing the optimal solution that maintains this requirement. While there exists many alternatives that would provide faster processing and communication speeds that draw higher power, our unique solution will strive to extend battery life and meet these other requirements as it can.

While we were luckily provided with most of the equipment necessary to complete our proof of concept, some were outdated or less efficient which prevented us from optimizing our solution. However, we were able to predict performance of a sensor node based on optimal components, shown in the Future Improvements section of this report.

In correspondence to the ongoing COVID-19 pandemic all provincial, local, and university restrictions and policies will also have to be observed and upheld. Unless necessary to meet virtually, in person meetings will be conducted with appropriate social distancing measures, masks, sanitation supplies, and other advised materials.

Stakeholder Analysis

A project of this scale involves many parties that are either working on the progress, or expecting the results. Among the stakeholders working on the outcome is Capstone Group 61, and Dr. Dean Richert, an assistant professor at the University of British Okanagan. Dr. Richert is the faculty advisor for the project who put forward the project idea and serves to guide our group regularly in weekly meetings, as well as provided us with initial resources to start the project. The members of group 61 chose to undertake this project and have to actively complete deadlines for evaluations for the capstone course ENGR 499 and as well are working to complete this project and to compete at the end of the academic year against other capstone groups to win

the respective category. Dr. Dean Richert and the members of group 61 share in any potential IP that is generated as a result of this project, thus they have a unique investment in this project compared to other stakeholders.

The other stakeholders in this project are Dr. Kenneth Chau and the School of Engineering who expect to receive the project deliverables and act as our bosses for the project. Dr. Chau is the course instructor for capstone and has set our deadlines and ultimately grades our work with our faculty advisor for our course mark. The School of Engineering is funding the project budget of $300 and also providing the course as a requirement for the Bachelor of Applied Science degree our group looks to complete from the University.

As well, future stakeholders could be considered in our project as our solution could be implemented by groups currently unknown to us. Not only will our solution be of interest but the steps of our process as well in order for them to understand how we got to each point and for any adaptability or troubleshooting they need to do. While our solution will be specifically made for acoustic sensors in theory the sensor network we implement will be able to work with different kinds of sensors making its range of problem adaptability wide for many applications.

Beyond this, the manufacturing industry as a whole serves to gain from any potential findings in this project. Breakthrough designs, or unique insights to low power sensor networks can act to further the industry as whole. Thus, while the industry has no influence on this project; it should be considered how this project fits into the bigger picture, and what would be considered a success considering past projects.

Lastly, end users such as manufacturing plant operators can positively benefit from this project. An acoustic sensor network could work to reduce operating costs, and improve overall plant efficiency. Thus, facilities that implement the sensor network designed in this project could be positively affected.

Current Solution Landscape

Predictive Maintenance (PdM) techniques are a method implemented with sensing technology that proactively monitors systems to detect failure early and initiate a fix. A similar concept, Preventative Maintenance, still has the same goals of early failure detection but is based

on scheduling of a routine check instead of constant monitoring and as such can still lead to failure in between checks. A study done by the company SKF that tested 30 identical bearings and found that the time to failure varied widely between 15 to 300 hours which made it difficult to foresee failure time and set a maintenance interval [2]. In this case implementing PdM would prevent under and over maintenance as well as increase worker safety, and reduce material waste [2].

The idea of implementing additional sensors to monitor a system is not unique to this project. The key element to this project is the low power element, and the ability for the nodes to tie into existing control systems. It is easy to find sensor nodes that require an electrical power supply, but very few solutions exist that are battery powered let alone have the infrastructure behind them to link to common control systems like a PLC. Being said, there are projects that have implemented technology similar to this project. For instance, in 2018 the City of Calgary implemented a LPWAN acoustic sensor node network throughout their city to measure the noise level and report any potential public disturbances [3]. These nodes would measure the average noise level at the node, then if the decibel level was over a specific threshold it would send an alert that could be further investigated in person [3]. The sentiment of the project was to reduce costs associated with noise complaints by proactively pursuing them as they occur. The hardware used in this project is very similar to what is required for this project, however the software side will vary greatly. The City of Calgary project is closely related with monitoring the noise level and identifying particular sounds, while our project is using past sensor data to investigate changes in the environment. These changes in pressure waves could be an object sliding off of a conveyor belt, or a motor grinding due to a lack of lubrication. This highlights the difference in software between the projects, where our machine learning algorithm needs to be tuned to detect these changes for a specific application. The other differentiating factor is that the City of Calgary simply required a sensor network to passively monitor sound levels, then a person must act on the warnings sent by the nodes. Whereas our project will be able to connect with existing control systems through a common medium. This means that our sensor nodes do not need to be specifically monitored in a dedicated location. They can instead be implemented into existing human machine interfaces (HMIs), or the sensor data can be acted on by a programmable logic controller (PLC), instead of requiring human intervention.

Other projects, such as that in [4] implement a microphone in order to monitor the sound emitting from a fan. The continuous stream of data is then analyzed to determine if the fan should be relubricated. The system implemented is very bulky, expensive, and would require the entire motor housing to be revamped in order to be implemented since the system is wired to an electrical source. This project is designed to be a substitute for these projects, while being cheaper, and much easier to implement. The important distinguishing factor between a wired connection and battery powered is the transmission frequency. Sending data at a millisecond rate would quickly consume power, and make a small battery operated node unfeasible. Thus, this project is designed for employment in situations where extremely quick response time is not vital.

Finally, there are projects such as that in [5] that implement a wireless sensor network (WSN) into industrial environments. The project in [5] uses a WSN to track the axle temperature for a freight train to ensure the heat generation does not exceed design limitations. The project uses low power sensors spread out beneath the train to monitor the temperature using a temperature sensor [5]. The solution was found to be accurate, while maintaining a low cost [5]. This WSN implementation is a great model for what this project aims to accomplish. The defining difference is again the use of the output signal from the sensor network. The project in [5] did not incorporate the signal into control of the train, instead it just warned the conductor. This is a common theme of existing WSNs, they collect data, but the data must be interpreted by a human to be acted on. Implementation into a control system reduces human intervention, which improves overall efficiency, and decreases likelihood of errors occurring.

Discussion of Design and Implementation

Chosen Proof of Concept

To demonstrate the viability of the wireless sensor node, we opted to develop a proof of concept that demonstrates the projects' purpose. The proof of concept will work to classify on a small FESTO work machine that detects whether a workpiece has a lid on it. The workstation is shown in figure 1 below.

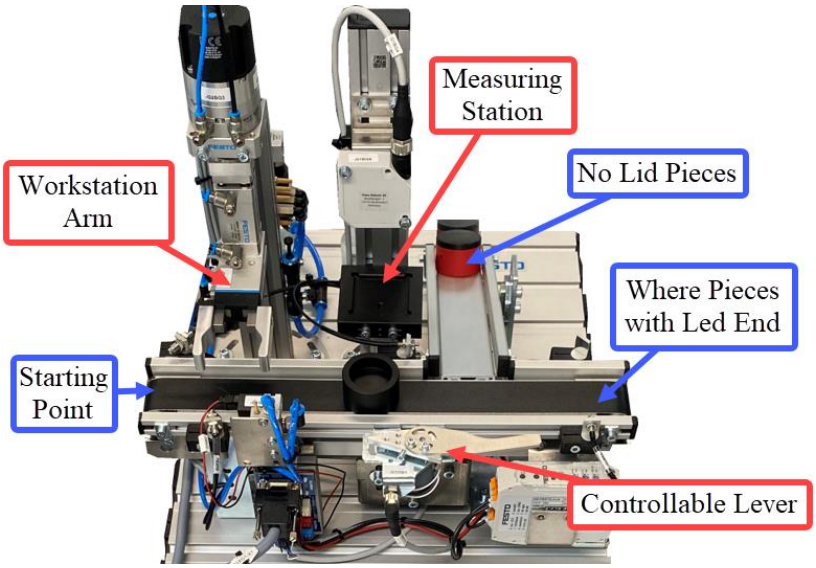


Figure 1: FESTO Workstation

The operation of the workstation is as follows. A workpiece is set on the left side of the conveyor belt, marked starting point on figure 1. Once detected by an initial sensor, the workstation arm will pick up the workpiece, rotate and place the piece on the measuring station. The measuring station detects if the workpiece has a lid or not depending on the measured distance from sensor to piece. The workstation arm will then pick the piece back up and place it back on the conveyor belt. The conveyor will carry the piece right, then if there is no lid a controllable lever will extend and push the workpiece down a ramp. Otherwise, if the piece has a lid the lever will not move, and the piece will move to the end of the workstation. The purpose of this automated task is to divert pieces that do not have a lid, presumably indicating a workpiece that missed a step in the line of production. The workstation is controlled by a Siemens PLC which is what the node will relay all information to for controls.

Using this simple workstation, we can come up with four interesting cases that a sensor node could listen for. The most obvious three are for a workpiece with no lid, a workpiece with a lid, and nothing on the conveyor belt. The final case we decided to classify for is the case of a jammed workstation arm. Through testing we found that two consecutive workpieces placed one after another tended to jam the workstation arm in the down position, effectively stopping the entire process. The jam requires human intervention to remove the workpiece or to manually retract the workstation arm.

To differentiate between these cases, we opted to use onboard machine learning for each sensor node. This means each sensor will read in audio data, classify it, then transmit the data back to the PLC. Machine learning may not be needed for such a simple classification scheme, but the ultimate purpose of a sensor node is to be implemented in environments that are potentially complex. In these environments, machine learning is essentially required to produce effective classifications. Additionally, using machine learning allows the sensor nodes to be easily implemented if produced. That is, all that would be needed to introduce a sensor node to a manufacturing facility would be to collect sufficient training data, then simply train the classifier with the data. This makes the sensor network extremely portable since the groundwork for the machine learning network is already in place and therefore implementation should be seamless.

With this general proof of concept knowledge in mind, we can continue to discuss the solution that we developed.

Machine Learning Classifier

Seeing as this project relies on robustness and accuracy of classification, it is key that a strong machine learning (ML) classifier is developed to meet the demands of a manufacturing plant. Should the classifier be too inaccurate, it would entirely defeat the purpose of the sensor. This is because the sensor is meant to be complimentary and hassle free, providing an easy to implement alternative to traditional wired or integrated sensors. False classifications that result in process delay are costly and thus the classifier should be over 99% accurate according to the initial prototype specifications. This number should be much higher in an actual manufacturing facility where 1 time in 100 is far too often for an error to occur. With this accuracy specification in mind, we can proceed and discuss the ML design process.

Before recording, the microphone is set to record at its designed operating frequency of 48kHz. A higher sampling rate is desirable as it allows us to detect higher order frequencies in the sample data. These higher order frequencies may be of interest depending on the events examined later. Being we are using a microelectromechanical microphone, the power draw is very low for a digital microphone. This means the tradeoff between high frequency operation and power draw is worth making due to the small relative size.

Before any classifier is developed, the sensor data must be translated to meaningful features which can be used for classification. To effectively communicate the design process, images of the feature data is shown as it is transformed to the final meaningful data. A sample audio signal for the initial raw data is shown in figure 2 below. This clip shows an 8-second audio sample of a workpiece traveling through the workstation without a lid.

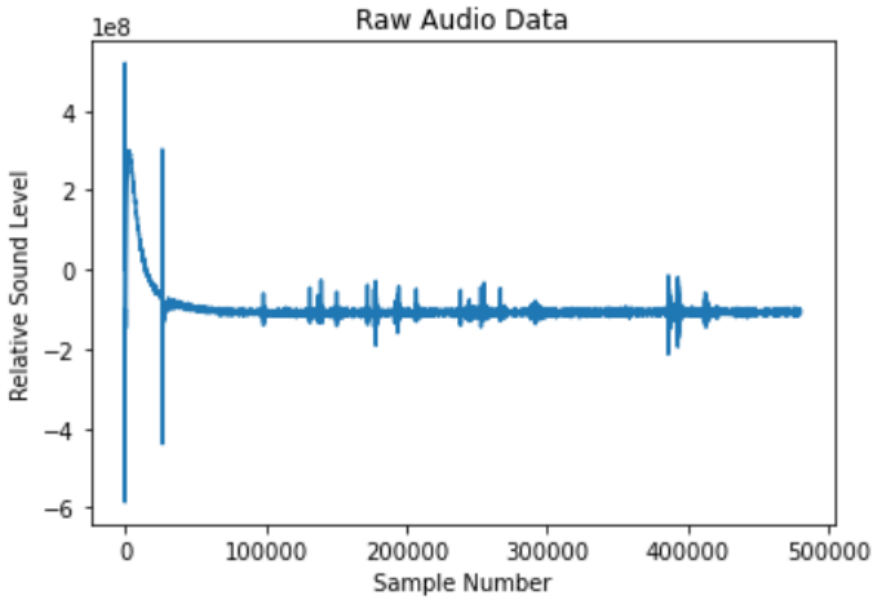


Figure 2: Raw Audio Data Plot

There are a few initial issues with the raw data. First, there is a very large high frequency spike when the microphone begins to record. This was explained upon referencing the microphones datasheet and noticing that the microphone is not designed to detect frequencies below 50Hz [6]. Being that this issue only occurs during the initial recording of the microphone we decided it was best to ignore the first 60,000 (~1.2 seconds) that the microphone records. Doing so we get the audio plot shown in figure 3.

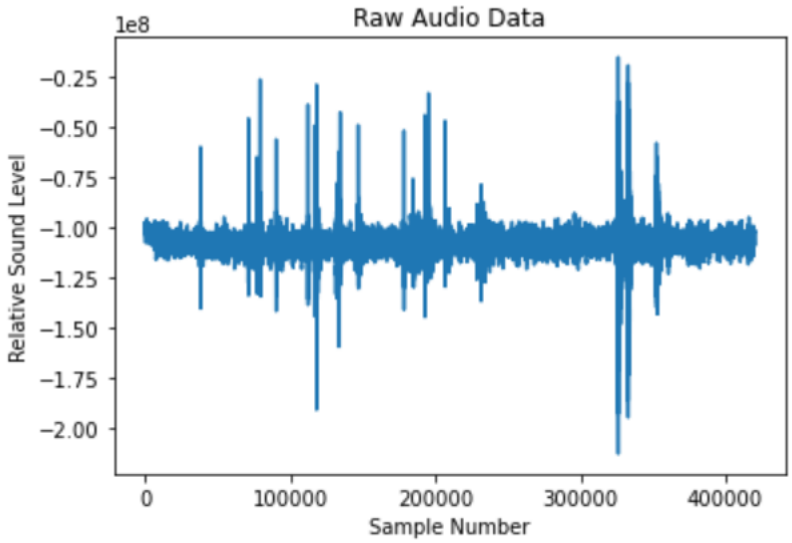


Figure 3: Raw Audio Data Trimmed

The first point of interest regarding the audio data in figure 3 above is that an event can occur anywhere within a sample window. This means that the raw sensor data cannot be used for classification since the meaningful data can be anywhere within the recorded audio window and a model would tune a classifier parameter inconsistently. This leads us down two possible design paths, either to use advanced ML algorithms, or to perform extensive feature engineering to extract meaningful data from the audio sample.

Advanced ML algorithms such as the long short-term memory (LSTM) algorithm can interpret sequences of data, as opposed to single data points [7]. This allows it to interpret time-based datasets, effectively eliminating the problem relating to identifying where an event occurs within a dataset. This initially sounds great, but there are significant issues associated with this technique. Firstly, algorithms such as LSTM are extremely technical and require advanced libraries to implement. A few students within our group are proficient in ML, but not masters. This makes implementing advanced classifiers hard due to the lack of background knowledge in the field of deep learning. As a group, we feared that delving into an advanced topic within a field in which we have no background would lead to a solution we were incapable of understanding fully. Additionally, advanced Python libraries are great for Raspberry PI implementation but add additional computational requirements to the sensor nodes. As well, standalone processors often run C/C++ which has a less broad library network for potential ML implementation.

With the pro/cons of advanced ML algorithms in mind, we opted to implement a more traditional classifier that reps less computational costs and a more transparent implementation. To do this, an effective feature engineering algorithm must be developed to compensate for the variations in the time domain start point. With this feature engineered vector, we can train a logistic regression classifier. A logistic regression model can be trained on a plant computer using training data, validated using testing data, then the entire model can be ported to the microprocessor simply by importing the model's coefficients using a thumb drive or even through the internet using something like FTP. The logistic regression has the advantage of being a simple combination of multiplications and additions which makes it easy to implement without additional bulky libraries or undue computations. Importantly, the decision to implement a logistic regression means potentially thousands of multiplications must occur for each classification depending on the microphone sample rate and length of classification window. Thus, it is vital that the chosen microprocessor/processor has an onboard floating-point unit (FPU) block on board. FPUs are dedicated chips for addition/multiplication with floating point numbers (non-integers) which drastically decrease the time taken to compute the class for a data sample.

Notice that from figure 3, there is significant ambient noise overlayed on top of the meaningful peaks within the sample. These noise-generated peaks introduce a problem when determining where an event starts within a data set, as they may falsely indicate the start of an event. Thus, the next logical step is to implement a filter to reduce the noise. Examining a data sample involving only the background lab noise we found that the background noise is concentrated below 130Hz. Thus, a high-pass filter (HPF) is tested and implemented to reduce the undue lower order harmonics.

Filter Design

By experimentation, we found that a Butterworth filter with an order of 6 and a cutoff of 150Hz resulted in the best filter design. While an order of 6 means a relatively dull cutoff, it is an optimal balance between computational constraints and filter effectiveness. The frequency response of the designed HPF is shown in figure 4 below.

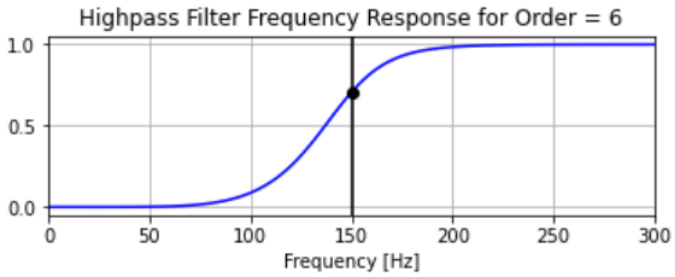


Figure 4: High Pass Filter Frequency Response

By convolving the filter designed with the audio signal we can get a filtered audio signal. This signal has clear peak values and being that each peak correlates to an audible event we can classify the data easier. Before filtering is done, the raw data is mean centered then normalized by the standard deviation. This technique allows for the audio data to look similar when the sensor position is changed nominally. For implementation in a real sensor node, the mean centering and normalization can be overlooked. The centered, filtered audio signal is shown in figure 5 below.

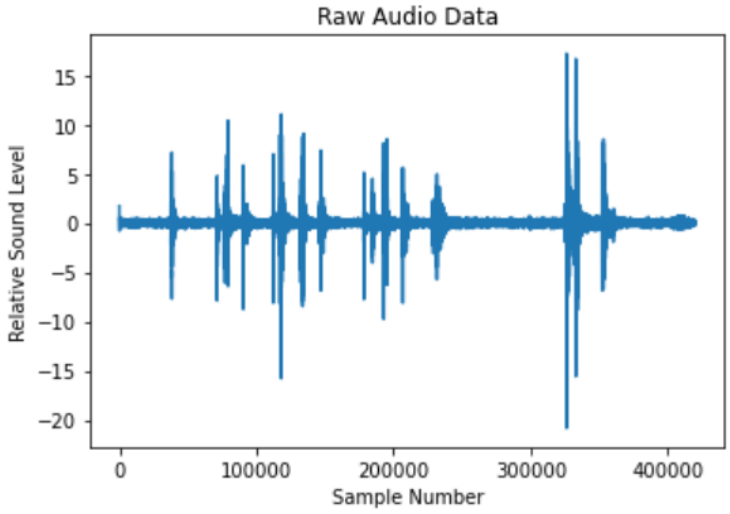


Figure 5: Centered and Filtered Audio Signal

Feature Engineering

Upon examining figure 5 above, it is clear that there exist both positive and negative signal values throughout. To reduce variability within the data, we opted to take the absolute value of the signal shown in figure 5, this results in the figure shown in figure 6 below.

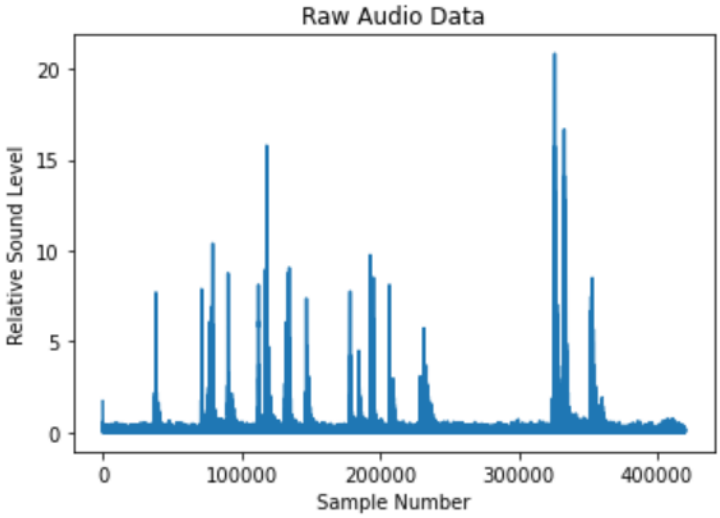


Figure 6: Absolute Value of Previous Audio Signal

Notice in figure 6, there are over 400,000 samples which means there would be over a million multiplications required in order to classify the sample. To rectify this issue it was found that down sampling the data resulted in a smaller sample size which resulted in faster classification speed. Down sampling is done by grouping together 1000 samples then taking the average between them and reducing the 1000 samples to a single new sample that is the average value between them. This scales down the dimensionality 1000 fold, hence saving precious time and energy requirements. The down sampled audio sample is shown in figure 7 below.

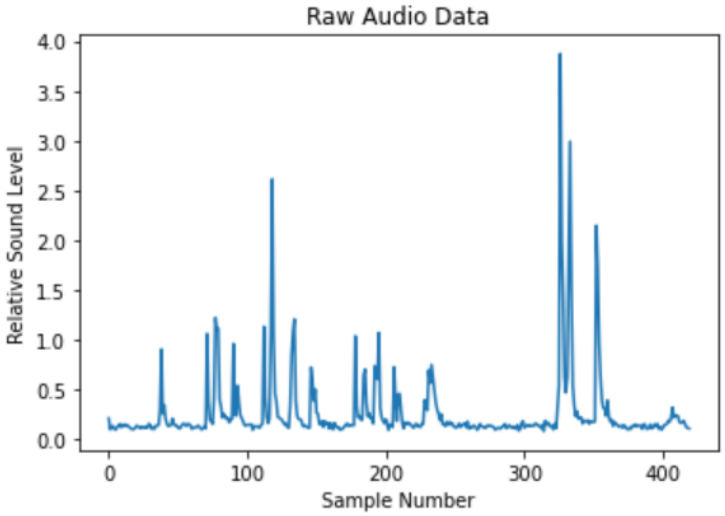


Figure 7: Down Sampled Audio Sample

Now, the feature engineering is nearly complete; the final aspect is to align the window of data with the start of the event. After experimenting with an appropriate threshold size, we found that a spike with magnitude greater than 0.35 indicates true start of an audio event. If there is no peak

above 0.35, we can reduce the data sample to zero and deduce that it is simply ambient noise. Additionally, all events for the proof of concept only take 8 seconds which correlate to approximately 350 down sampled samples. Thus, we can align the event within the classification window by aligning the start of the window with the first occurrence of a peak greater than 0.35, then including the following 350 samples following. The resulting shifted audio simple is shown in figure 8 below.

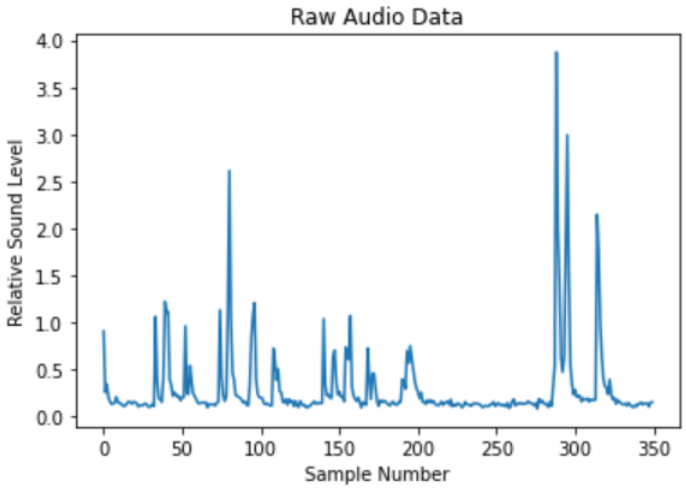


Figure 8: Shifted Audio Sample

The audio sample in figure 8 has properly windowed around the event and made clear the prevalent peaks in the data. The technique mentioned is robust and allows for the event to occur anywhere within an audio sample. The number of samples is also reasonable, meaning there are only about a few thousand computations to be completed per classification.

The final step in designing a classification algorithm is to simply train a logistic regression classifier. This was done using a Python library called *SkLearn*. *SkLearn* saved time manually training a classifier, and instead we simply saved the coefficients for the model to a CSV file which can be transferred to the microprocessor. To show the power of the feature engineering algorithm mentioned, we can plot each case for the proof of concept and overlay each sample within each case. For the proof of concept, we had four cases being lid on, lid off, jammed pieces (called consecutive pieces), and ambient noise. The training data has 10 samples per case; thus we have 40 audio samples which are plotted after having their features extracted. This plot is shown in figure 9 below. The Python script to do the feature engineering and train the classifier is shown in Appendix A.

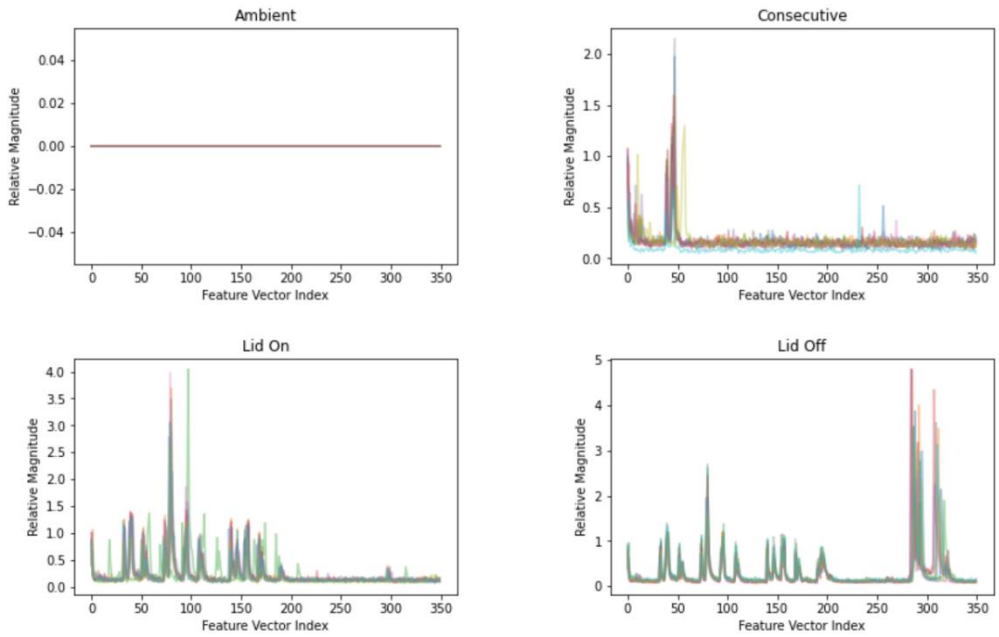


Figure 9: Feature Extracted Data Plots

Classifier Accuracy

By examining figure 9 above it is clear that the feature engineering aligned the data samples excellently. The dark lines indicate that like samples are in sync within the window, this is key to a well-behaved classifier. With the classifier trained using the data shown in figure 9, we tested the data against a similar set of testing data. The testing data was collected during the same session as the training data, thus a well performing ML classifier should classify the testing data well. By classifying each data sample, then comparing with the audio samples true test case we can gather an accuracy for the classifier. Doing so we found that the classifier we trained is 98% accurate. The data gathered is shown in a confusion matrix in figure 10 below.

Testing Data Confusion Matrix					
		Event Prediction			
		Ambient	Consecutive	Lid Off	Lid On
Actual Event	Ambient	4	0	0	0
	Consecutive	0	10	0	0
	Lid Off	0	0	10	0
	Lid On	0	1	0	9
Accuracy: 97.7%					

Figure 10: Classifier Confusion Matrix

Although our aim was to develop a classifier with 99% accuracy, we opted to settle with a 98% accuracy classifier. This is not a large compromise and is reasonable for the purpose for this

proof of concept, where the ML algorithm a single piece of the larger puzzle that is the construction of the entire sensor node.

Real-Time Classification

Classification of the audio sample is done on board the sensor node. Thus, the logistic regression model coefficients must be stored locally on the microprocessor which can be used for classification. This means the coefficients must be ported from the computer where training occurred to the target sensor. This was done using a USB drive to transfer a CSV file between the devices. With the model accessible from the sensor node, we can move forward with the real-time classification technique.

For the sensor nodes to be feasible in a manufacturing facility, they must be flexible in terms of classification periods. Meaning that two events should be able to occur in sequence with little time difference in between events. This introduces a unique challenge where events can occur anywhere within time, but the events can only be correctly identified after the sensor data has been processed. There are two issues that result by using a singular windowing technique. First, events that span two windows lose information and their classification will be missed. Secondly, two events occurring in a single window will fail as the current feature engineering solution will only detect the first event occurrence.

These issues seem to render the idea of a singular windowing method useless but examining how manufacturing plants typically operate can give insight to these problems. Manufacturing facilities are finely tuned systems, and there is often little randomness involved within the system timing. Thus, we can assume that events are periodic and predictable by nature. This allows us to construct a single window that is timed such that the event of interest is guaranteed to fall within the window. Consider the following scenario to illustrate the idea; a plant has unstable objects coming down a conveyor belt where they stop and are measured. Suppose we want to implement a sensor to listen to detect if the object tips over. If an object of interest enters the specific area every 20 seconds the sensor can listen for 15 seconds, classify the data, and send the information then begin to listen 5 seconds later. Being that the event is predictable and periodic, the sensor is tuned to always be alert when a workpiece enters, then classify during downtime between

potential events. Figure 11 below illustrates the process timeline in terms of both the event and how the system listens and performs computations.

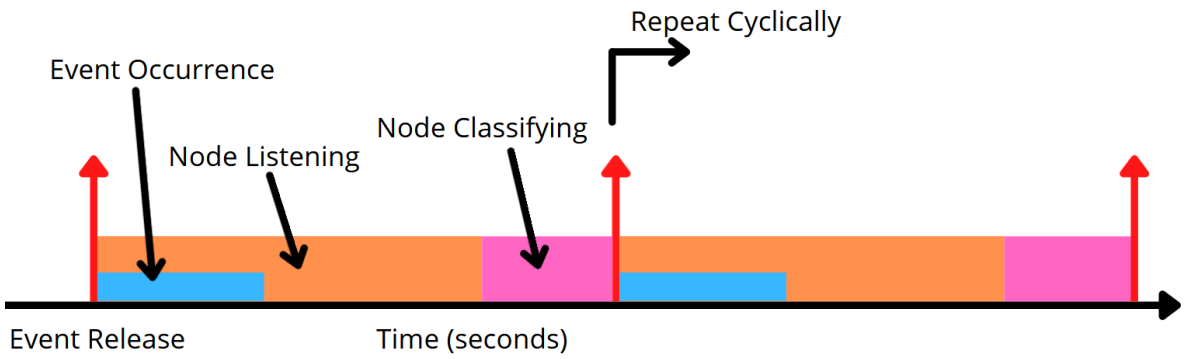


Figure 11: Real-Time Classification Windowing Technique

We can now apply the windowing technique to the specific proof of concept for this project. Considering each workpiece takes approximately 8 seconds to process through the system we assumed that the pieces arrive 25 seconds apart. We then opted to use a 20 second window, which leaves for 5 seconds of processing and sending the resulting data.

With a 20 second window, the ML classifier was able to run from start to finish in under one second. The method was validated through testing where the classifier operated as intended, both classifying correctly and working regardless of event location within the time sample. The communication strategy is the focus of the next section of the report, but we were able to achieve under 1 second transmission speed from sensor node to PLC. This means the total computation plus transmission time is under 2 seconds which places the node well within its allotted 5 seconds time frame for computation and communication.

Communications

This section relates to the communication used to transmit data from the wireless sensor node to the PLC. This includes a few key components which are discussed below.

LoRaWAN

The main communication method in this project is the use of Long-Range Wide Area Network (LoRaWAN) technology for transmitting sensor state values to the gateway. Our transmitter was setup for communication using the North American specification. This transmitter sends

hexadecimal byte data across a LoRaWAN network, the bytes are then received at an IoT gateway. LoRaWAN allows for network versatility as many device parameters can be adjusted to fit the application. This transmitter uses Activation by Personalization (ABP) and operates under the North American frequency band of 902-928 MHz [1]. Example python code for joining the ABP network is shown in Appendix B. Figure 12 demonstrates the correlation between data rate (DR), spreading factor (SF), bandwidth (BW), bitrate (BR), and range configurations for the lora module. Our transmitter was configured with high a DR of 4, and SF8 as our proof-of-concept was not configured for long-range connectivity. These settings allowed the maximization of BR without compromising any range capabilities. This is important as real-time applications rely on speed to meet critical deadlines. Table 1 below summarizes these settings.

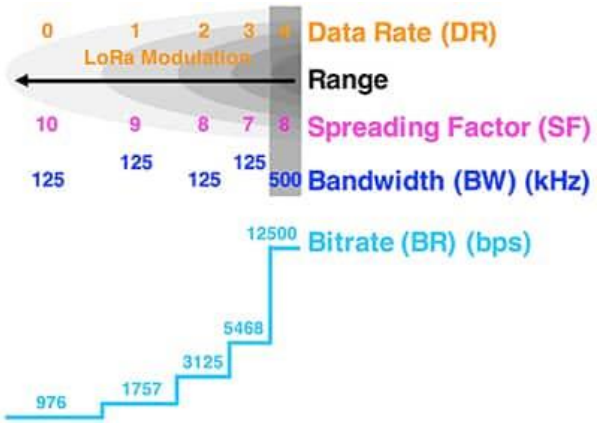


Figure 12: Common LoRaWAN data range, spreading factor, bandwidth, and bitrate correlation.

Activation	ABP
Data Rate (DR)	4
Spreading Factor (SF)	8
Bandwidth (BW) (kHz)	500
Bitrate (BR) (bps)	12500
Tx Frequency (MHz)	902-928

Table 1: Summary of LoRaWAN configurations.

Once the state of operation is determined by the machine learning algorithm, the data is transmitted by the LoRa device to the receiving network gateway using a hexadecimal key. This value is sent to the gateway with a payload that is base64 encrypted along with information about the device such as its identification number.

MQTT

Once messages are received by the Tektelic gateway, the data is stored on a Tektelic backend server which can be accessed via a web browser using login credentials. Using the backend

server, we were able to configure and push Message Queueing Telemetry Transport (MQTT) messages to an MQTT broker. MQTT is an industry standard messaging protocol which uses a publish and subscribe architecture to send messages between remote devices while minimizing bandwidth [8]. We chose to use the HiveMQ broker hosted at 'broker.hiveMQ.com' where we configured a topic 'v2/pushCapstone' that all the base64 encoded transmitter messages would be pushed to. Figure 13 displays the flow between the MQTT client and broker with the respective subscription topic. From there, a local python script subscribes to the topic and pulls the messages, as shown in Appendix C, where they are decoded and pushed to the OPC-UA server.

Figure 13: MQTT Topic Flow

OPC-UA

Open Platform Communication Unified Architecture (OPC-UA) is a widely used communication protocol between machines in industrial automation. A server was established on the stations PLC to provide the source of information, in our case, variables in a data block to update the station state. Using a local computer as a client to connect to the server through URL our group established a connection to read and write onto the PLC data. Through this connection our group was able to update in real time the station state variable from MQTT Client to PLC. With the state now updated our PLC was able to display the changes on the HMI to accurately display the monitored event. Figure 14 below shows a screenshot of the HMI in default state, and once it receives from the microprocessor that ambient noise is detected.

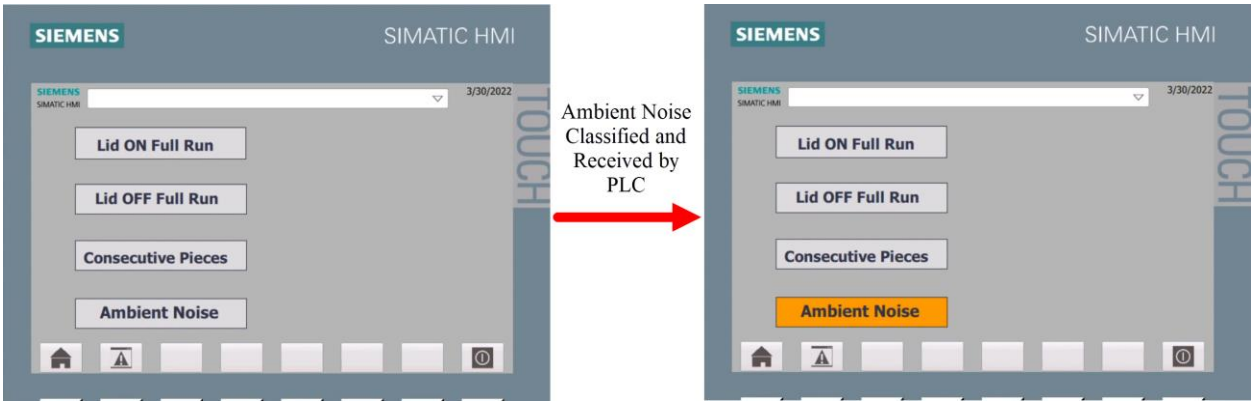


Figure 14: HMI Display Changing States

The flow of information within the proof of concept is shown in figure 15 below; it captures the entire lifecycle from audio emitting event to PLC input.

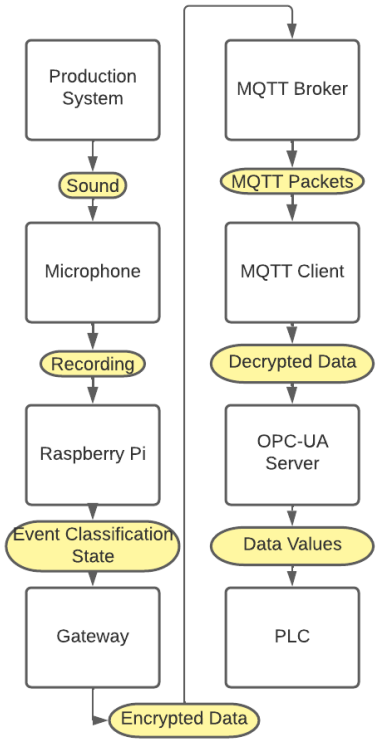


Figure 15: Proof of Concept Information Flowchart

Proof of Concept Hardware

One of the most important aspects of this project is the hardware chosen for the sensor nodes. Each node is designed to run for 10 years according to the project specifications; thus, the

hardware should be very low power devices accompanied by large batteries. Regardless of the hardware chosen there are three crucial aspects of the sensor node, these being the microprocessor/processor, microphone, and transmitter-gateway pair. The microphone will read in audio data, then the microprocessor will analyze the data where it will be sent to a transmitter which relays the data to a gateway. The gateway uploads the data to the internet, finally the PLC can pull the data and act accordingly.

For this project, we opted to demonstrate a proof of concept of this process. Given the time and monetary constraints of the project we decided to focus on the functionality of a sensor node. This included getting a prototype working, but not abiding by all the specifications already determined in the initial project definition report. As a result, completion of this proof of concept will demonstrate the feasibility of the idea; but the model will not be directly implementable in a manufacturing facility.

With this in mind, we decided to choose hardware for the project that is popular and easy to use. This prevents time spend debugging issues that may have little to no online support. Additionally, we opted to use hardware provided to us by our project supervisor and faculty advisor Dr. Dean Richert. Given Dr. Richert's background in wireless sensor nodes, it is logical to use equipment he is familiar with as it allows us to have additional avenues for support through him.

Microprocessor Choice

We opted to use a Raspberry PI 3B+ as a microprocessor. PIs are essentially small configurable desktop computers, they have an onboard processor, RAM, wireless Wi-Fi, four USB 2.0 ports, an HDMI port, and a GPIO header [9]. The PI also has an onboard floating-point unit which makes it optimal for doing consecutive numeric calculations. Raspberry PIs are very popular and user friendly which cuts down on development time. The drawback with the PI is that it draws 12.5W power at maximum load. This is many orders of magnitude off of what a low power device consume, thus it cannot be implemented in a long-lasting sensor node. Being said, anything implemented on the PI can be done on a less-user-friendly processor that has more desirable power draw. Thus, the PI is the perfect choice for the proof of concept.

Microphone Choice

The onboard microphone decision has a few different angles. First, the microphone should have a high sampling frequency. According to Nyquist's sampling theorem, in order to detect a given frequency we must sample at twice the given frequency [10]. Accordingly, a high frequency microphone will be able to pick up detect additional frequencies which may be crucial depending on the events frequency domain characteristics. The next, and most important microphone property is the devices power consumption. Power draw is critical for sensor nodes battery lives as mentioned earlier. The final, and least important aspect of the chosen microphone is its output data format; being either analog or digital. All microphones are analog by nature but ensuring the microphones output data format aligns with the microprocessor capabilities is required for them to communicate.

With all these aspects in mind we opted to choose the Adafruit I2S MEMs microphone. The I2S is a low power, digital output microphone, that is designed to operate at 48kHz. These properties designate it as a great choice for both the proof of concept, and for the final sensor node construction.

Transmitter-Gateway Choice

The last consequential hardware to select is the onboard transmitter, and the gateway it connects to. These pieces of hardware should provide reliable communication, data encryption, and be able to transmit the required 2-bits representing the classified event every ~20 seconds to be consistent with the designed classification technique. Lastly, the pair should communicate using LoRaWAN and consume a low amount of power. These requirements are specifically for the transmitter, the gateway only must be reliable and capable of communicating with the transmitter. We opted to choose the RN2903 LORA(R) MOTE transceiver module as it fulfills all requirements demanded of the transmitter. The only pitfall of this transceiver module is that it is a few years old and draws more power than newer versions. The corresponding gateway used is the Tektelic Kona Pico which is now obsolete to its successor which has upgraded features such as a four hour backup battery and an integrated 3G/4G cellular modem [13].

All the components used for the proof of concept, as well as their power draws are shown in table 2 below.

Proof of Concept Hardware Choosen	
Component	**Power Draw**
Raspberry PI 3B+ Microprocessor	12.5 W
Adafruit I2S Microphone	1 mW
RN2903 Transmitter	10 mW

Table 2: Proof of Concept Hardware Components

Proof of Concept Performance

The final proof of concept electrical hardware setup is shown in figure 16 below. Note that the processor-transmitter-microphone bundle are the components of the sensor node, the gateway is physically separate from the node.

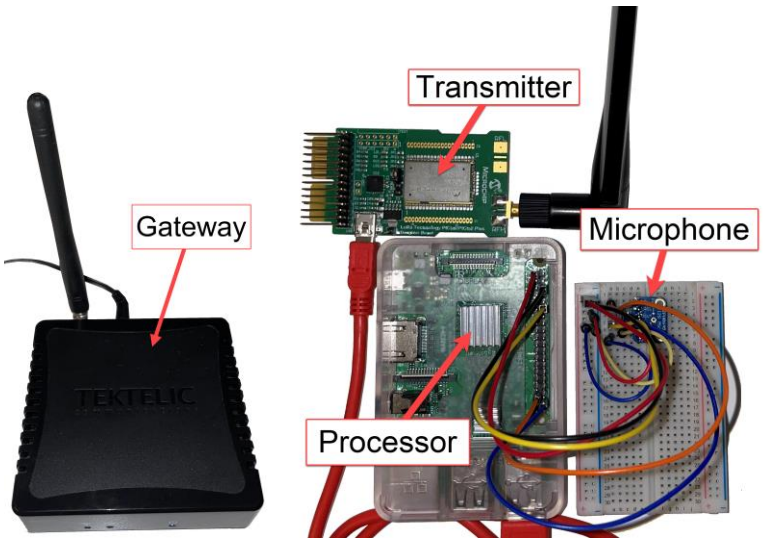


Figure 16: Proof of Concept Electrical Hardware

The resulting sensor node proved to be extremely effective. We measured the sensor to be over 95% accurate under ideal lab conditions. However, any “unknown” lab sounds, such as people talking triggers the ML algorithm incorrectly and the results become unreliable. Furthermore, any small changes in the actual workstation result in completely unreliable results. For instance, should the workpiece be placed awkwardly on the belt and take an extra few tenths of a second to slide down the conveyor belt, the resulting classification is unreliable. The final Python script used in the proof of concept is shown in Appendix D.

The largest issue associated with the proof of concept is the communication between the onboard transmitter and the gateway. The issue was Over the Air Activation (OTAA) not properly

connecting therefore we were required to implement Activation By Personalization (ABP). ABP uses a fixed device address, fixed security, and fixed network parameters which require manual configuration and may cause uplinks and downlinks to not function properly [17]. This resulted in the transmission speed between sensor node and gateway to take upwards of 3 seconds. Though we could not fix this issue, we suspect the error is a result of the proof of concept using relatively old hardware and a new transmitter would rectify the issue. Another potential source of error is the distance between transmitter and gateway. LoRaWAN is meant for long range transmission, but in this proof of concept the pair were less than five meters away from one another. Increasing the distance between transmitter and gateway would place then within optimal communication distance and allow for my effective transmission. Ultimately, this error is minor and can be rectified easily through additional testing which requires additional time we do not have.

The proof of concept developed demonstrates the concept of the project and does so effectively. It acts to show the viability of sensor nodes in manufacturing plants. The next section is dedicated to discussing future improvements to the sensor node to make it widely implementable.

Future Improvements

While there are many manufacturing facilities that have implemented Industry 4.0 sensor technologies, there is still a sizeable group that is not as modernized that would benefit from implementing something like a network of the sensor nodes proposed in this document. For these production systems our sensor nodes could be implemented seamlessly in addition to their existing layout. Our solution would be cost effective and easy to integrate, making it a more appealing option for companies that do not want to invest into a more costly revamp of their systems. The LPWAN network can be easily established with a few gateways to connect all sensor nodes to the network.

Machine Learning Algorithm Improvements

Having discussed the detailed algorithm in the previous section leads us naturally to the potential improvements. Potential improvements in the machine learning section are merely observational and further testing is required to validate any speculations.

The first method to increase the performance of the ML implemented would be to examine the classifier used. The logistic regression model implemented is simple and not robust, meaning any small changes to the time domain signal results in inaccurate classifications. Adding more complex deep learning algorithms has the potential to increase reliability of the classifier and thus make it more likely to be implemented in industry. Potential classifiers to examine next include the LSTM mentioned prior, or premade libraries such as tensor flow. As mentioned prior, these libraries and advanced learning methods may come with the caveat of increasing processing requirements which may push computation times beyond the desired time frame.

Should a more sophisticated classifier be implemented, it would be of interest to look at scaling back the feature engineering done. Feature engineering was required to make up for the fact we implemented a rather dull classifier, thus a more advanced model may need less help in order to pull out the desired features from the audio data. Performing less feature engineering may free up additional time for the classifier to work, thus a tradeoff may be possible between a more advanced training model and less pre-classification computation.

Real-Time Classification Improvements

The initial classification technique assumes that there is time for computation between events occurring. But what if the events are back-to-back, and there is not time for the node to perform computation between events occurring? This introduces the need for parallelization of events. The most effective improvement to the classification method is to introduce multithreading. Multithreading allows specific cores on the processor to handle specific tasks, and thus eliminate the series occurrence of recording, then classifying. With multithreading, a singular core could read in sensor data then pass the data off to a separate core for classification and transmission. Provided the separate thread could perform its action quicker than the window length, the microprocessor will work as intended. Multithreading is the easiest technique to guarantee correct timing for periodic events.

Should the events not be periodic, more sophisticated windowing techniques are needed. This could include multiple overlapping windows that are able to capture the event, or even implementing algorithms that do not begin recording until a specific threshold or occurrence. Note that these windowing techniques likely also require multithreading to operate effectively.

Hardware Improvements

The proof of concept developed is complete and functional, but it is not fit for implementation in a wireless sensor node. This stems from a few issues. The first is the electronic hardware used in the prototype. The “microcontroller” used in the prototype is extremely power demanding and must be addressed for feasible implementation. Additionally, for a wireless sensor node we must choose corresponding batteries that can power the desired microcontroller. The next improvement lies in the transmitter used; the proof of concept uses an old and power heavy transmitter that can be refreshed to improve sensor node lifespan. We can then use these components to calculate the power draw of the entire sensor node to estimate the shelf life of the node.

Microcontroller

For industry application, it would be desirable to use the lowest-powered microcontroller available instead of the Raspberry Pi used in our implementation. After some research, it appears that the microcontroller with the best ULP (ultra-low-power) bench score is the ON semiconductors RSL 10 chip [11]. It has a bench score of 1090, which is derived from the inverse of the average power consumption in μW over 50 iterations, multiplied by 1000 [11]. This means that the processor consumed about 1μW while operating during the test. By our estimation, according to the datasheet for the processor, it would consume approximately 2.25 mW on average when in use [12]. The chosen microprocessor is skewed towards power use and does not have optimal performance specifications. For instance, the RSL10 does not have an onboard FPU which may be problematic. Accordingly, while our best estimations are that the processor is adequate, additional testing is required. An image of the chosen microcontroller is shown in figure 17 below.

Figure 17: Chosen Optimal Microprocessor [12]

Batteries

Our selected battery, the EEMB ER34615 will be deployed in packages of 4. With each having an energy capacity of 19,000mAh or 68.4Wh, each node would have a total of 273.6Wh at its disposal [13]. The battery was selected as it is a lithium thionyl chloride battery which boasts low self-discharging at about 1-2% per year [14]. Additionally, these batteries offer high energy densities which allows to pack more energy in a smaller form factor [14]. The final advantage of these batteries relevant to this project is their operable temperature range from -80° to 125° Celsius [14]. This is relevant for industrial environments where temperatures may reach

extremes. Each battery has a diameter of 34mm, and a height of 61.5mm which is quite small. This size allows us to place four batteries per sensor node cartridge. An image of the chosen batteries is shown in figure 18 below.

Figure 18: Chosen Batteries [13]

Transmitter

The RN2903 transmitter should be swapped out with the latest version of the same product line. This is the RN2483 microchip, which has the same fundamental qualities while drawing nearly half of the power as the proof-of-concept counterpart [15, 16]. From [16] we can find the average power draw of the transmitter is approximately 5mW. An image of the chosen transmitter is shown in figure 19 below.

Figure 19: Chosen Optimal Transmitter [16]

Sensor Note Battery Life

With the power consumption data from the ideal electrical hardware components, we can calculate the theoretical battery life of the node. The microcontroller has 2.25mW draw, and the transmitter consumes 5mW as mentioned previously. The final component is the microphone which consumes approximately 1mW. These component specifications are shown in table 3 below. Given the proposed battery pack has a capacity of 273.6Wh, we can calculate the battery

life of the sensor node to be 3.8 years. The calculation for the battery life span is provided in Appendix E below. This is below the target 10 years lifespan from the specifications. Depending on the desired physical size of the node, additional batteries can be added to further increase the lifespan.

Sensor Node Cost

The final aspect of the wireless sensor nodes is the cost. Manufacturing plants likely require tens to hundreds of sensor nodes to effectively monitor their process. Thus, each node must be cost effective to be at all realistic. By adding the total hardware cost for each electrical component, we can find the total cost of each sensor node is $250CAD [6, 12, 13, 15], the specific cost breakdown is shown in table 3 below. The largest costs are associated with the microcontroller, transmitter, and batteries. The price may be increased due to the current semiconductor shortage, and we suspect the price will steadily drop over the coming months/years. Additionally, buying components in bulk would considerably reduce the prices. Savings could also lie in purchasing the batteries in a larger capacity directly from a wholesaler. This alone could drop the price by ~$75CAD. That being said, $250CAD per sensor is nearly three times our initial specification of $100CAD, and this design does not consider the sensor housing construction.

Ideal Sensor Hardware Components		
Power Consuming Components		
Component	**Power Draw**	**Cost**
RSL10 Microprocessor	2.25 mW	$ 71.80
Adafruit I2S Microphone	1 mW	$ 6.99
RN2483 Transmitter	5 mW	$ 25.00
TOTAL:	8.25 mW	$ 103.79
Power Providing Components		
Component	**Energy Capacity**	**Cost**
Batteries	273.6 Whr	$ 150.00
TOTAL:	273.6 Whr	$ 150.00
Final Design		
Runtime on Battery:	3.12 years	
Total Cost:	$ 253.79	

Table 3: Ideal Sensor Design Summary

Conclusion

While there are many manufacturing facilities that have implemented Industry 4.0 sensor technologies, there are still a select group that are not as modernized that this project would fulfill a need for. For these production systems our sensor nodes could be implemented seamlessly in addition to their existing layout. Our solution would be cost effective and easy to integrate, making it a more appealing option for companies that do not want to invest into a more costly revamp of their systems. The LPWAN network can be easily established with a few gateways to connect all sensor nodes to the internet to be used for monitoring and/or data collection and analysis.

The node proposed in this report is robust as it relies only audio data, which is readily available and easy to collect in almost every setting (including non-industrial applications). Since it relies only on audio data, there is precious little work to be done to integrate a network of our sensor nodes when compared with the cost and labour associated with purchasing modern components with their own integrated sensors. Our solution is as close to ready-out-of-the-box as one could ask for.

Our sensor node is also extremely portable due to our use of machine learning. Because we have laid the groundwork for ML algorithm implementation onboard a sensor node, all that's required to tune a node to a specific application is training data so that the node can accurately classify events of interest. However, the use of ML techniques extends the utility of our node beyond simple time-domain sequential events. For instance, a node could be used to detect faults when listening to an operating motor and preemptively indicate to an operator that maintenance is required.

Our proof of concept developed illustrates the sensor nodes viability in a manufacturing facility. Over a relatively short duration of time, we developed a wireless sensor node that works in accordance with periodic tasks to classify specific events using audio data alone. This data is then used to directly tie to an existing control system where it can be used to alter the given process. While further development is required to finalize each specific sensor node, this proof of concept can serve as a launching point towards more feasible wireless sensor nodes in manufacturing facilities.

Appendices

Appendix A

```
import numpy as np
import pyaudio
import wave
import matplotlib.pyplot as plt
import struct
from scipy.io import wavfile
import os
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import linear_model
from sklearn import svm
import sklearn
import pandas as pd
from scipy.fftpack import fft,fftfreq
from scipy import signal
import math
from scipy.signal import butter, lfilter, freqz
import joblib

def butter_highpass(cutoff, fs, order=5):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='hp', analog=False)
    return b, a

def butter_highpass_filter(data, cutoff, fs, order=5):
    b, a = butter_highpass(cutoff, fs, order=order)
    y = lfilter(b, a, data)
    return y

def get_audio_data(rel_dir, en_filter = 1):
    width = 1000
    trim_value = 3750_00

    # data from audio files
    x = []
    # corresponding audio file class
    y = []
    # unique classes seen by the algorithm, for creating new classes
dynamically
    classes = {};
```

```
    for item in os.listdir(rel_dir):
        # unqique identifier for file name
        identifier = item[0:12]

        # IF statements used to determine if class has been seen before
        if identifier in classes.keys():
            # assign class key to the list of labels
            y.append(classes[identifier])

        else:
            # checks if any classes have been added yet
            if( classes):
                # add class file name identifier to the classes, assign index
one higher than maximum value present
                classes[identifier] = max(classes.values()) + 1
            else:
                classes[identifier] = 0
            # append new class to list of labels
            y.append(classes[identifier])
        samplerate, data = wavfile.read(f'{rel_dir}/{item}')
        #### ALL CODE BELOW IS DEDICATED TO FEATURE ENGINEERING
        # trim first 60k samples (shortly after clap)
        data = data[60000:]
        # mean center and scale data
        data = (data - data.mean()) / data.std()
        # filter using HPF
        if( en_filter):
            data = butter_highpass_filter(data, cutoff, fs, order);
        data = np.abs(data)
        # compresses data into bins to reduce dimentionality, works great
after filter
        data = data[:(data.size // width) * width].reshape(-1,
width).mean(axis=1)
        # If peak value < 0.36 must be ambient noise
        if ( np.max(data) < 0.36 and en_filter): # 0.36 threshold
            data[:] = 0
        # looks at the data for the first non-zero value (first peak in wave)
        first_peak = 0
        for i in range(data.size):
            if( data[i] >= 0.35 and first_peak == 0):
                first_peak = i
        #trim from first peak onwards to end (sound runs long anways)
        data = data[first_peak:]
        data = data[:350]
```

```
        if( data.shape[0] < 350):
            print(item)
            y.pop(-1)
        else:
            x.append(data)
    return x, y

x, y = get_audio_data("./training")

nn = linear_model.LogisticRegression(solver = "liblinear" )
df = pd.DataFrame(x)
df = df.apply(lambda row: row.fillna(row.mean()), axis=1)
x = df.to_numpy()
nn.fit(x,y);

df = pd.DataFrame(nn.coef_)
df.insert(0, "intercept", nn.intercept_)
display(df)
df.to_csv("ML_intercept_and_coefs.csv")
```

Appendix B

LoRaWAN RN2903 Python Scripting:

```
import rn2903
import time
import serial
import sys
import glob

# list serial ports
print(rn2903.list_serial_ports())

# open serial connection
con1 = rn2903.open('/dev/serial/by-id/usb-
Microchip_Technology_Inc._LoRa_Tech._PICtail_Board-if00')

# print connection status
print(rn2903.status(con1))

devaddr = "c429fcbb"
nwkskey = "52cf3cf9982813369023cf2e716ebce7"
appskey = "f2c44bac845ad32f17ed292456d0d1e7"

rn2903.raw_command(con1, "sys reset")
time.sleep(2)

# set device address
rn2903.command_response(con1, "mac set devaddr "+str(devaddr),"ok")
print("Device Address Set")
time.sleep(4)

# set network session key
rn2903.raw_command(con1, "mac set nwkskey "+str(nwkskey))
print("Network Session Key Set")
time.sleep(4)

# set app session key
rn2903.raw_command(con1, "mac set appskey "+str(appskey))
print("App Session Key Set")
time.sleep(4)

# save mac settings
rn2903.raw_command(con1, "mac save")
time.sleep(4)

# join activation by personalization (ABP)
rn2903.raw_command(con1, "mac join ABP")
print("ABP Joined")
print(rn2903.macRecBuf(con1))
time.sleep(4)
sys.stdout.flush()
```

```
# loop forever
while 1:
        ** code for sending messages can be added here **
```

Appendix C

Python script for MQTT – OPC-UA connection:

```
'''
	Script to monitor a sensor and save MQTT messages, then upload to OPC-UA server.
	Author: Jonathan Lake
	Date: 01/03/2022
'''

'''
	CAPSTONE PROJECT PIPELINE (simple):
	RASPBERRY PI sensor information (hex) -> TEKTELIC GATEWAY (base64) -> HiveMQ Broker (base 64)
	|-> THIS script (base64 to hex conversion) -> HiveMQ (hex) and OPC-UA (hex)
'''

# python imports
from socket import timeout
import paho.mqtt.client as mqtt
import code
from opcua import Client
from opcua import ua
import time
import json
import sys
import base64

# function for decoding base64 encoded messages
def decodePhyPayload(msg):
	# extract the physical payload from the message
	# and convert to hex
	PHYPayload = base64.b64decode(msg).hex()

	return PHYPayload


# must be unique to other instances of this script that are running simulataneously
client_name = "loraNode"
# HiveMQ broker address
broker = "broker.hivemq.com"
# push topic from Tektelic gateway
topic = "v2/pushCapstone"

# create empty array for messages
msg_list = []

# mqtt 'on connect' behavior function
def on_connect(mqttc, obj, flags, rc):
	if rc == 0:
		mqttc.connected_flag = True
```

```
                print("connected ok")
        else:
                print("bad connection. returned code = ", rc)

# mqtt 'on subscribe' behavior function
def on_subscribe(mqttc, obj, mid, granted_qos):
        mqttc.subscribed_flag = True
        print("subscribed ok")

# mqtt 'on message' behavior function
def on_message(mqttc, obj, msg):
        global msg_list
        print(1)
        msg_py = json.loads(msg.payload)
        msg_py["topic"] = msg.topic
        msg_py["qos"] = msg.qos
        print(json.dumps(msg_py, sort_keys=True, indent=4, separators=(',', ':
')))
        msg_list += [msg_py]


if __name__ == '__main__':

        # configure mqtt
        mqtt.Client.connected_flag = False
        mqtt.Client.subscribed_flag = False
        mqttc = mqtt.Client(client_name)
        # bind call back functions
        mqttc.on_connect = on_connect
        mqttc.on_subscribe = on_subscribe
        mqttc.on_message = on_message
        # set username and password
        mqttc.username_pw_set('', password='')

        # connect to broker
        print("connecting to broker " + broker)
        try:
                mqttc.connect(broker, 1883, 60)  # connect to broker
        except:
                print("can't connect")
                sys.exit(1)
        mqttc.loop_start()
        while not mqttc.connected_flag:
                print("waiting for connection...")
                time.sleep(1)

        # subscribe to topic
        print("subscribing to topic " + topic)
        mqttc.subscribe(topic, 1)
        while not mqttc.subscribed_flag:
                print("waiting to subscribe...")
                time.sleep(1)
```

```
        # loop forever!
        num_msgs = 0
        try:
                while True:
                        time.sleep(1)
                        while msg_list != []:
                                try:
                                        # get lora payload from msg
                                        RawPayload =
msg_list[0]['0004A30B001A820C'][0]['values']['nsRawPayload']

                                except KeyError:
                                        # handle exception
                                        print("Key not found.")

                                # decode payload
                                DecodedPayload = decodePhyPayload(RawPayload)
                                # print decoded payload
                                print("Decoded Payload: ", DecodedPayload)
                                # publish decoded payload to HiveMQ broker
                                mqttc.publish("v1/pull", DecodedPayload, 0,
True)
                 # remove the processed message
                                msg_list = msg_list[1:]

                                # set OPC-ua client address
                                client = Client("opc.tcp://192.168.0.1:4840/")
                                try:
                                        # connect to client
                                        print("connecting to OPC-UA client...")
                                        client.connect()
                                        print("connected ok")

                                        # get node value
                                        PLCstate_node =
client.get_node('ns=3;s="OPC"."PLCstate"')
                                        PLCstate = PLCstate_node.get_value()

                                        # set node value
                                        var1_setValue =
ua.DataValue(ua.Variant(int(DecodedPayload,16),
PLCstate_node.get_data_type_as_variant_type()))
                                        PLCstate_node.set_value(var1_setValue)

                                        # print new value
                                        print("Posted Value: ",
int(DecodedPayload,16))

                                except timeout:
                                        print("connection timed out")

                                finally:
                                        # disconnect from client
```

```
                                        try:
                                                client.disconnect()
                                                print("disconnected ok")
                                        except AttributeError:
                                                print("OPC-UA client not
connected")


        except KeyboardInterrupt: # Ctrl-c to quit
                print("stopping client loop")
                mqttc.loop_stop()
                print("disconnecting from broker " + broker)
                mqttc.disconnect()
```

JSON packet received from MQTT broker:


```
{
    "0004A30B001A820C": [
        {
            "ts": 1649622813697,
            "values": {
                "bytes": "[61]",
                "nsFCount": 249,
                "nsFPort": 2,
                "nsGateway": "647fdafffe00511d",
                "nsGatewayCount": 1,
                "nsGwId": "bff4f970-aa3d-11e8-9876-a36e3fbec477",
                "nsRawPayload": "AQ==",
                "nsRssi": -117,
                "payload length": 1,
                "port": 2
            }
        }
    ],
    "qos": 1,
    "topic": "v2/pushCapstone"
}
```

Appendix D

```
import time
import serial
import sys
import rn2903
import pyaudio
import wave
import numpy as np
from scipy.signal import butter, lfilter, freqz
import joblib
from sklearn import linear_model
import pandas as pd


def butter_highpass(cutoff, fs, order=5):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='hp', analog=False)
    return b, a

def butter_highpass_filter(data, cutoff, fs, order=5):
    b, a = butter_highpass(cutoff, fs, order=order)
    y = lfilter(b, a, data)
    return y
def feature_extraction(data):
    # Filter requirements.
    order = 6
    fs = 48000     # sample rate, Hz
    cutoff = 150  # desired cutoff frequency of the filter, Hz
    width =1000
    trim_value = 375000
    print("start")
    data = np.array(data)
    print("np array")
    data = data[60000:]
    print("cutoff")
    # mean center and scale data
    data = (data - data.mean()) / data.std()

    print("mean centered")

    # filter using HPF
    data = butter_highpass_filter(data, cutoff, fs, order);
    data = np.abs(data)
```

```
    print("filtered data")

    # compresses data into bins to reduce dimentionality, works great after
filter
    data = data[:(data.size // width) * width].reshape(-1,
width).mean(axis=1)

################################################################################
#########

    if ( np.max(data) < 0.36):
        data[:] = 0

    first_peak = 0
    for i in range(data.size):
        if( data[i] >= 0.5 and first_peak == 0):
            first_peak = i
        #trim from first peak onwards to end (sound runs long anways)
    data = data[first_peak:]

        # trims from end to have normalized size, scaled by width to
accomidate different binning
#         data = data[:int(trim_value/width)]
    data = data[:350]
    if( data.shape[0] < 350):
        x = np.zeros(350)
    else:
        x = np.array(data)
    return x

print(rn2903.list_serial_ports())
#con1 = rn2903.open('/dev/ttyACM0')
con1 = rn2903.open('/dev/serial/by-id/usb-
Microchip_Technology_Inc._LoRa_Tech._PICtail_Board-if00')

print(rn2903.status(con1))

form_1 = pyaudio.paInt32 # 16-bit resolution
chans = 1 # 1 channel
samp_rate = 48000 # 44.1kHz sampling rate
CHUNK = 24000 # 2^11 samples for buffer
dev_index = 2 # device index found by p.get_device_info_by_index(ii)
RECORD_SECONDS = 500

audio = pyaudio.PyAudio() # create pyaudio instantiation
```

```
flag = True
stream = audio.open(format = form_1, rate = samp_rate, channels = chans,
input_device_index = dev_index, input = True, output = False,
frames_per_buffer = CHUNK)


print("recording")
frame1 = []

df = pd.read_csv("ML_intercept_and_coefs.csv")
df.drop(columns=df.columns[0], axis=1, inplace=True)
intercepts = df["intercept"].values
df.drop(columns=df.columns[0], axis=1, inplace=True)
coefs = df
coefs = coefs.values

channel_count = 0

for i in range(0, int(samp_rate / CHUNK * RECORD_SECONDS)):
    data = stream.read(CHUNK,exception_on_overflow = False)
    data = np.frombuffer(data, np.int32)
    frame1.append(data)
    temp = []

    timeElapsed = i / 2 +.5
    print(timeElapsed)

    if( timeElapsed % 20 == 0):
        if( i != 0):
            toClassify = frame1
            for i in toClassify:
                for j in i:
                    temp.append(j)
            print("Frame 1 length {}".format(len(toClassify)))
            frame1 = []

    if( len(temp) != 0):
        data = feature_extraction(temp)
        probs = {}
        keys = [0, 1, 2, 3]

        for key in keys:
            summer = intercepts[key]
            for i in range(len(coefs[key])):
```

```
                summer = summer + coefs[key][i] * data[i]
            probs[key] = summer
        print(probs)

        max_key = -1
        max_val = -9999999
        for key,value in probs.items():
            if( value > max_val):
                max_key = key
                max_val = value
        name_map = {
            0: "Ambient",
            1: "Consecutive",
            2: "Lid On",
            3: "Lid Off"
            }
        print(name_map[max_key], max_val)
        channel_map = {0:1, 1:5, 2:7}
        channel_selected = channel_map[channel_count % 3]
        channel_count = channel_count + 1
        print("Channel selected: " + str(channel_selected))
        print("Sent Msg Response: ", rn2903.raw_command(con1,"mac tx cnf " +
str(channel_selected) + " " + str(max_key)))
        time.sleep(3)
        while(rn2903.macRecBuf(con1) != "mac_err"):
            time.sleep(0.1)
        sys.stdout.flush()
```

Appendix E

The following is a calculation of the theoretical battery life of the sensor node. The total power draw from the sensor can be calculated by adding the power draw from each individual component. We can let *P* equal this sum. *P* is calculated as follows:

$$P = 2.25\text{mW} + 1\text{mW} + 5\text{mW} \; = \; 8.25mW$$

Given the battery has an energy rating of 273.6W·hr, we can calculate the life span as follows:

$$\text{Battery Life} = \frac{273.6Wh}{8.25mW} \times \frac{1\text{day}}{24hr} \times \frac{1yr}{365\text{days}} = 3.8yr$$

Thus, the battery life is 3.8 years.

References

[1] - “PDF.” LoRa Alliance, San Ramon, Nov-2015.https://lora-alliance.org/wp-content/uploads/2020/11/what-is-lorawan.pdf

[2] - S. Selcuk, “Predictive maintenance, its implementation and latest trends,” Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, vol. 231, no. 9, pp. 1670–1679, 2016. https://doi.org/10.1177%2F0954405415601640

[3] - SEMTECH, “LoRa Technology: Remote and Real-time Acoustic Monitoring.” 2018. https://www.semtech.com/uploads/technology/LoRa/app-briefs/Semtech-UseCase-Calgary-UrbanAlliance-web.pdf

[4] - Cihun-Siyong Alex Gong, Huang-Chang Lee, Yu-Chieh Chuang, Tien-Hua Li, Chih-Hui Simon Su, Lung-Hsien Huang, Chih-Wei Hsu, Yih-Shiou Hwang, Jiann-Der Lee, Chih-Hsiung Chang, "Design and Implementation of Acoustic Sensing System for Online Early Fault Detection in Industrial Fans", *Journal of Sensors*, vol. 2018, Article ID 4105208, 15 pages, 2018. https://doi.org/10.1155/2018/4105208

[5] - S. Zhang, *et al.*, Design of axle-temperature detect system based on wireless sensor network (in Chinese), Modern Electronics Technique (3) (2008), pp. 86-88. https://en.cnki.com.cn/Article_en/CJFDTotal-XDDJ200803030.htm

[6] - A. Industries, “Adafruit I2S MEMS microphone breakout - SPH0645LM4H,” adafruit industries blog RSS. [Online]. Available: https://www.adafruit.com/product/3421.

[7] - Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

[8] - “MQTT - The Standard for IoT Messaging”, Mqtt.org, 2022. [Online]. Available: https://mqtt.org/.

[9] - Raspberry Pi, “Buy A raspberry pi 3 model B,” *Raspberry Pi*. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/.

[10] - A. Hero, “Nyquist Sampling Theorem.” https://www.eecs.umich.edu/courses/eecs206/archive/f02/public/lec/lect20.pdf

[11] - “EEMBC,” CPU Energy Benchmark – MCU Energy Benchmark – ULPMark™ – EEMBC Embedded Microprocessor Benchmark Consortium. [Online]. Available: https://www.eembc.org/ulpmark/ulp-cp/scores.php.

[12] - “RSL10-002GEVB”, DigiKey. [Online]. Available: https://www.digikey.ca/en/products/detail/onsemi/RSL10-002GEVB/7942070

[13] – “EEMB 3.6 V D Size Lithium Battery ER34615 19000 mAh Li SOCL2 UL Certified Non Rechargeable 3.6 Volt Lithium Thionyl Chloride Battery”, Amazon Canada. [Online]. Available: https://www.amazon.ca/EEMB-Lithium-Certified-Rechargeable-Chloride/dp/B075ZPXG6M/ref=sr_1_3?crid=2HQTW1MCZXUGA&keywords=lithium%2Bthionyl%2Bchloride&qid=1649088083&s=electronics&sprefix=%2Celectronics%2C184&sr=1-3&th=1

[14] - “What are lithium thionyl chloride batteries? main advantages,” Tadiran Batteries. [Online]. Available: https://tadiranbatteries.de/lithium-thionyl-chloride/.

[15] - “RN2903A-I/RM103,” DigiKey. [Online]. Available: https://www.digikey.ca/en/products/detail/microchip-technology/RN2903A-I-RM103/8019733.

[16] - “RN2483A-I/RM105,” DigiKey. [Online]. Available: https://www.digikey.ca/en/products/detail/microchip-technology/RN2483A-I-RM105/9867251.

[17] - “ABP vs OTAA”, Thethingsindustries.com, 2022. [Online]. Available: https://www.thethingsindustries.com/docs/devices/abp-vs-otaa/.