University of British Columbia

Social Ecological Economic Development Studies (SEEDS) Sustainability Program

Student Research Report

Capstone Project 68: Designing Window Sensors to Advance Bird- Friendly and Energy Saving Building Design Strategies on UBC Vancouver Campus

Prepared by: Huawen Li, Gengran Li, Ryotaro Hokao, Benjamin Powell, Mohamed Salah

Prepared for: UBC Campus and Community Planning

Course Code: ELEC 491

University of British Columbia

Date: 27 April 2023

Executive Summary

Bird collisions with windows represent a growing environmental concern, as one of the leading causes of human-related bird mortality. The consequences of these collisions extend beyond the immediate loss of individual birds; they also pose a threat to overall biodiversity and ecosystem health. To address this issue, our engineering capstone project aims to develop a comprehensive system that detects bird collisions while also calculating the heat flow rate of windows. This project builds upon the past capstone project design that performs the bird collision detection using an accelerometer and Arduino.

The motivations behind this project stem from the need to better understand and mitigate bird collisions, as well as to improve the energy efficiency of buildings. By providing researchers and building managers with valuable data on the frequency and circumstances of bird collisions, we aim to contribute to the development of targeted strategies and solutions that can reduce bird mortality rates and protect biodiversity. Additionally, by analyzing the energy efficiency of windows via heat flow rate, our project can inform decisions about building design and materials, potentially leading to more energy-efficient and environmentally friendly construction practices.

Our proposed monitoring system consists of:

1. **Collision Detection:** accelerometer attached to the window to measure the window vibration. Data is sent to the Arduino for analysis.
2. **Temperature:** two Dragino LHT65 temperature sensors to measure the inside/outside surface temperatures of a window. Surface temperatures are transmitted to ThingSpeak via LoRaWAN, and used for heat flow rate calculation.
3. **Microcontroller:** Arduino MKR WiFi 1010 to analyze and infer a collision from the accelerometer data. It also monitors whether the accelerometer has fallen from the window. Once a collision is detected, or when the accelerometer falls, relevant data is transmitted to ThingSpeak via WiFi. Multiple tasks are managed by FreeRTOS.
4. **ThingSpeak:** a cloud-based database and user-configurable dashboard to monitor relevant data. Data received from Arduino and Dragino sensors are processed and automatically displayed. In case of accelerometer falling, ThingSpeak sends an email

notification. Users can also input the U-Value and the dimension of the window, allowing the system to compute the heat flow rate.

5. **Hardware:** two 3D printed enclosures. One of them is designed to be weather resistant, and is for outdoor use.

Researchers and other users can view not only the collision data and heat flow rate, but also other environmental data such as outside air temperature and the time of the collision on ThingSpeak. This information can be invaluable for understanding the factors that contribute to collisions and developing effective mitigation strategies, to potentially identify patterns and correlations with bird collisions.

This iteration of the project adds more functionalities to the previous design, with increased cost and size to the previous design. Our design, however, takes a modular design approach; depending on the users' needs (i.e., monitoring bird collision, or measuring heat flow rate), they can choose to purchase only the components required for their use case. Throughout this document, the alternative approaches and recommendations are discussed for future improvements.

Table of Contents

Requirements

1.0 Background and Outcome

Through the release of the Green Building Action Plan (GBAP) in 2018 and climate action plan 2030 that addresses climate change and biodiversity, University of British Columbia (UBC) is continually working on building a green sustainable environment among the campus. The goal of these plans is that by 2035, all newly constructed and rebuilt buildings on the UBC campus will achieve high performance through meeting low cost, low energy consumption, and low environmental emissions requirements [1]. Most importantly, those buildings should also avoid and cut back on environmental control and costs related to natural health deterioration. For this ambitious goal to be successful in safeguarding campus ecology, and encouraging energy balance, bird-friendly building design is a crucial element. According to a study by UBC [1], more than 10,000 birds perish each year by colliding windows because they cannot perceive translucent glass as solid [2]. In response to the GBAPs request as well as to maintain bird species number, UBC Social Ecological Environment Development Studies (SEEDS) and Green Building Manager have begun looking into solutions to lessen the number of fatalities caused by bird impacts on windows. In 2019, a student team in UBC had already thought about and created a basic model for this issue. However, the previous bird strike monitor model which can only keep track of the time and number of impacts is no longer an efficient way to monitor the data. In this project, the bird strike monitor will be redesigned to allow UBC green building researchers to better collect and visualize bird strike data and analyze sustainable factors such as heat transfer and insulation of windows.

The final design can accurately count collisions and provide associated details of each incident, including season, location, temperature inside and outside, time and date of collision occurrence. Data mentioned above enables UBC green building researchers to identify potential causes behind bird strikes and examine the efficacy of various window treatments when designing bird friendly window treatments or choosing glazing materials. Furthermore, the heat transfer through a glass window is calculated from temperature data collected, which assists green building researchers to assess the thermal insulation of various glazing materials [Different window glazing has different factors that influence the heat mitigation measurement] for better energy

savings. UBC green building researchers therefore could create more thorough strategies for the protection and conservation of bird populations. Improved bird friendly strategies enlighten green building managers and UBC building developers for challenges arising from compliance with the GBAP, the UBC Climate Action Plan 2030 (CAP 2030) and Bird Friendly Building Design Guideline. Parts of these strategies such as examination results of bird friendly window treatments and the thermal insulation of various glazing materials offers UBC building developers practical guidance window design and glazing materials selection. Moreover, an online database with a researcher-friendly interface is provided for multiple researchers to operate and export data without assistance. In short, this design will not only provide reliable and accurate data for researchers to build better bird-friendly strategies, but also helps provide practical guidelines for building developers and managers.

2.0 System Process Diagram

This system's flowchart clearly demonstrates how the complete bird strike monitor model operates and how it interacts with researchers.

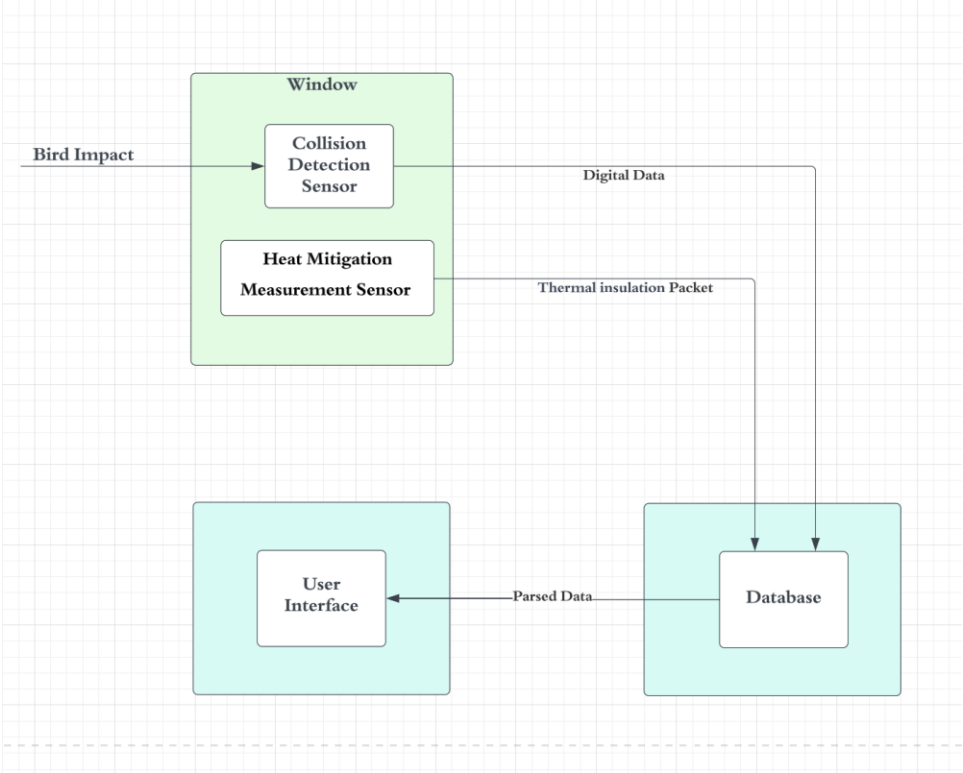


Figure 1: System Operation Flowchart.

The system collects data from various sensors placed on the windows and processes it uses a central processor. The processed data, including bird impact-related information, as well as window insulation performance data, is then sent to the backend of the user interface. The backend further processes the data with accuracy and transmits it to the frontend for display to the user. This enables users to conveniently and efficiently observe and record data, facilitating further research and implementation of GBAP measures for bird conservation.

3.0 Functional Requirement

Functional requirements which include the services, capabilities or functions delivered by the product, are identified as follows:

F1: The system must detect collisions with windows and infer whether it is a false-positive collision based on calculated force and frequency metrics of the collision.

F2: The system must record the cumulative number of birds strikes along with the time of day, season, and date at which the strikes occurred[1] [3].

F3: The system must record the latitude and longitude of the building that window placed.

F4: The system must record the data packet for the fall detection.

F5: The system must record the surface temperature of the window inside and outside.

F6: The system must record the air temperature outside of the building.

F7: The device must have the ability to secure itself to the window and no part of the device can be left unattached to, or separated from, the window under normal operation and avoid dropping in adverse weather conditions.

F8: The user interface must graphically display stored data.

F9: The user interface must export historical data from the database to the user's computer as a file.

F10: The system must alert the user in the case of the accelerometer falling from its attached position on the selected window.

4.0 Non-Functional Requirement

Non-functional requirements which include the quality attributes the product needs to exhibit (the "ilities": reliability, security, portability, interoperability) are identified as follows:

NF1: The accuracy of bird strike detection must be higher than 95%. (Using simulated bird strikes.)

NF2: The calculation error of the heat flux[2] should be within $\pm 0.001\ Watts/m^2$

[1] According to the UBC science faculty research, 19 bird strike fatalities occur every 225 days in one building on the Vancouver campus.

[2] $|Internal\ Temperature\ -\ External\ Temperature|\ /\ R\ value$

NF3: The measurement error of the window's surface temperature and air temperature must be within ±2°C.

NF4: The data recording must be fully functional for a minimum of 30 days before requiring maintenance.

NF5: The outside part of the device must be waterproof to avoid water ingress in the event of rain. Reach the IP standard IP33 [4].

NF6: The device must not have color or shape features that attract bird species and/or other animals [5].

NF7: The database must have sufficient storage size to store at least 10 bird impact events per day.

NF8: The temperature sensors should measure the temperature per hour.

5.0 Constraints

Constraints which identify conditions the product must meet in terms of how it is implemented are identified as follows:

C1: The whole project cost (including hardware, software, and auxiliary supplies) should be under $180[3].

C2: Materials chosen for the device must withstand the operating temperature of between -20°C ~ 40°C. (±6°C based on maximum and minimum temperatures of the past year) [6].

C3: The device must operate on a wireless network.

C4: The device should support both battery charging and wall outlet 120VAC charging methods.

[3] Because many of these devices may be required by researchers in the future. For several of the model to be used for bird strike monitoring, the cost option needs to pick the most affordable one in order to maintain the overall cost within the researcher's means. [Database must be free for the user.]

C5: The external and internal hardware design should be within 15cm x 6cm x 6cm dimensions to achieve a compact design.

C6: The weight of the entire hardware design (Both external and internal) should be within the 200 g.

C7: The system must function on a window with dimension 80cm x 50cm. Bird collision detection model must work on typical windows on the UBC campus.

Deliverables

The Table below lists the deliverables that will be submitted at the end of the project, agreed upon between the clients and the project group.

Table 1: List of Deliverables: Hardware, Software, and Documents.

Item	Description
Hardware	

Device	• Detect collisions with windows
	• Record the collision vibration metrics via an accelerometer
	• Infer whether the collision is a bird strike or a false-positive
	• Detect accelerometer falling from position on window
	• Record:
	  ○ the location of the bird strikes (by location of the placed hardware component)
	  ○ the time, day, and date of the bird strikes
	  ○ the temperature:
	  ▪ inside surface temperature of the window being monitored
	  ▪ outside surface temperature of the window being monitored
	• Periodically upload recorded data to cloud repository, through UBC's visitor WiFi
	• Immediately upload accelerometer-fall and notify the user

Software	
Cloud Repository	• Display all constituents of received data (time of day, temperatures, etc.) • Retrieve outside ambient temperature from weather website • Allow U-Value and Window dimension input • Calculate Heat Flow Rate • Allow the user to aggregate selected groups of recorded data into a covarying table representation • Visually represented the data, in the form of co-varying histograms (i.e., strikes vs time of day). • Allow the user to export all data, formatted, off the cloud, to the working computer that is accessing the repository.
Documents	

Key documents	• Verification and Validation Document • Requirements Document • Design Document • Presentation Slides
User Manual	• Installation of hardware component • Connecting hardware component to cloud repository • Traversal of cloud repository features
CAD files	• Computer aided design files for designs of the 3D printed hardware model encapsulation

Blueprints	• Engineering drawings of entire device assembly
	  ○ Arduino
	  ○ Temperature Sensors
	  ○ Vibration Sensors and attachment to window
	  ○ Power Supply
	  ○ 3D-printed case attachment to windowsill
	  ○ 3D-printed case enclosure

Bill of materials	• Parts list • Manufacturer • Quantity of each part • Price of eat part
Capstone Video	• Video explaining project/problem and our solution

Design

1.0 System Architecture

1.1 Hardware

1.1.1 Introduction

In this section, we will discuss the various hardware components used in the bird detection system, their roles, and their integration to form a fully functioning system. The system is comprised of an Arduino MKR WiFi 1010 microcontroller, an ADXL343 accelerometer, two Dragino LHT65 sensors, two DS18B20 thermocouples, two 3D printed hardware enclosures, and a power cable (or lithium-ion battery). Figure 2 illustrates the setup of the system.

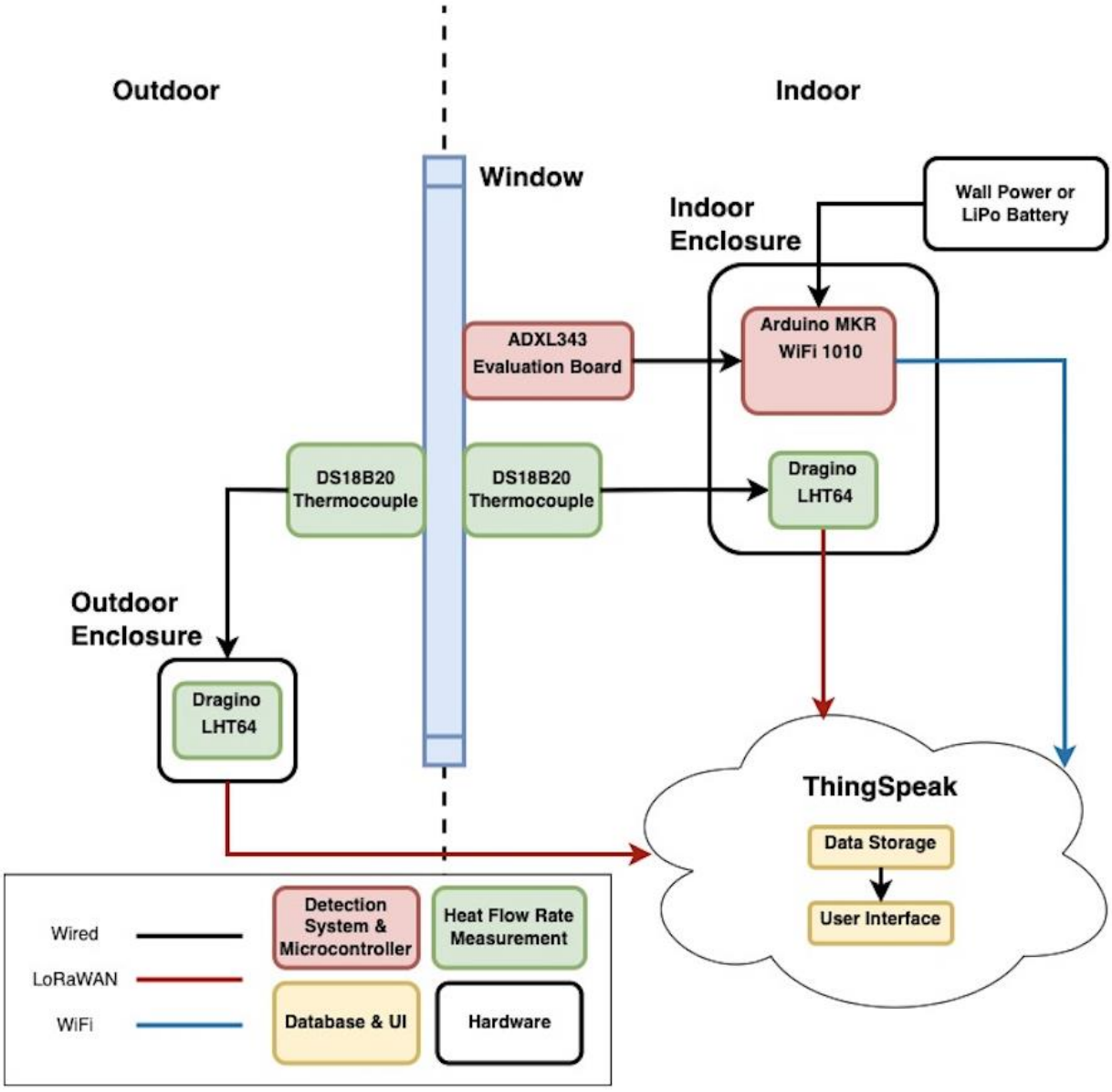


Figure 2: Hardware System Setup

1.1.2 Microcontroller: Arduino MKR WiFi 1010

The Arduino MKR WiFi 1010 serves as the central processing unit for the bird detection system. It is responsible for processing data from the ADXL343 accelerometer, as well as handling communication with the ThingSpeak platform for bird collision uploading.

1.1.3 Accelerometer: ADXL343

The ADXL343 accelerometer is responsible for taking samples of vibration signals collected from the window it is attached to. The ADXL343 is a digital accelerometer, handling the analog-to-digital conversion itself. The ADXL343 communicates with the Arduino MKR WiFi 1010, sending vibrational data for further processing.

1.1.4 Sensors: Dragino LHT65 and DS18B20 Thermocouples

The system uses two Dragino LHT65 sensors and two DS18B20 thermocouples to measure the surface temperatures of the windows. These sensors communicate this data to ThingSpeak through their built-in LoRaWAN modules, providing the necessary data for calculating heat flow rates across windows.

1.1.5 Power Options: Wall Outlet and Battery

The bird detection system can be powered in two ways: through a wall outlet or a battery. When powered by a wall outlet, the system has a continuous power supply, ensuring uninterrupted operation. Alternatively, the system can be powered by a 3.7V 3200mAh lithium-ion battery, allowing for up to four days of operation without recharging. This versatility allows for the system to be deployed in various locations without being constrained by the availability of power outlets.

1.1.6 Hardware Enclosure

The system has two 3D printed enclosures. They are printed using FDM technology to ensure robustness and being able to withstand pressure.

1.2 Communications

In this section, we will discuss the communication aspects of the bird collision detection system and heat flow monitoring system within the context of the System Architecture (see section 9.0 for a more detailed overview). Efficient and reliable communication between the various hardware components and the cloud platform (ThingSpeak) is essential for meeting functional requirements ***F2*** and ***F3,*** and constraint ***C3.*** The system utilizes different communication technologies and protocols to achieve this goal:

1.2.1 I2C Communication

The connection between the Arduino MKR WiFi 1010 and the accelerometer relies on the I2C communication protocol. This protocol enables a simple, two-wire interface for exchanging data between the devices, allowing the Arduino to receive vibration data from the accelerometer.

1.2.2 WiFi Communication

The Arduino MKR WiFi 1010 is equipped with built-in WiFi capabilities, enabling wireless communication with the ThingSpeak platform. The WiFi communication allows the Arduino to transmit bird collision data to ThingSpeak for further analysis and visualization.

1.2.3 LoRaWAN Communication

The Dragino LHT65 temperature sensors utilize LoRaWAN technology for long-range, low-power communication. This technology allows the temperature sensors to transmit data over a wide area while consuming minimal power. The transmitted data is received by a LoRaWAN gateway, which then forwards the data to The Things Network (TTN). TTN processes the data and forwards it to ThingSpeak for further analysis and visualization.

1.2.4 Serial Communication

The Arduino communicates with the user's computer via a USB connection, employing serial communication for debugging and configuration purposes. This connection enables the user to monitor the system's operation, upload new firmware, or modify existing configurations as needed.

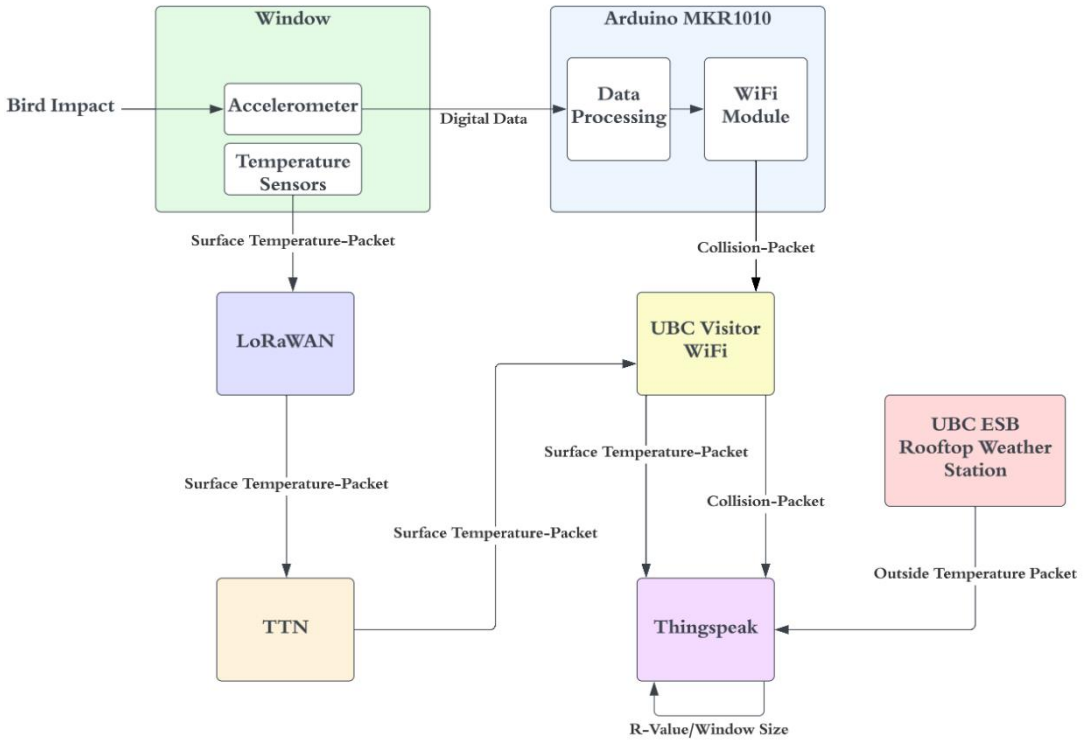


Figure 3: System Data Pathway Flowchart

Figure 3 illustrates the flow of data between various components for both the Arduino MKR WiFi 1010 and the Dragino LHT65 sensors in the overall system. Each section is assigned a different color to help visually distinguish the various components of the system.

1.3 Data Pathways

In this section, we will go over the specific data pathways of the components within our system, including the Arduino, Dragino sensors, and external data sources.

1.3.5 Arduino MKR WiFi 1010 Data Pathway

The data pathway for the Arduino MKR WiFi 1010 with an attached accelerometer involves the sampling of data to detect bird collisions on the window. When a collision or fall is detected, the Arduino processes the data and sends the relevant information to ThingSpeak over WiFi for further analysis and visualization.

1.3.6 Dragino LHT65 Sensors Data Pathway

The data pathway for the two Dragino LHT65 sensors, responsible for measuring the surface temperature of the window's inside and outside surfaces, involves communication with the LoRaWAN network. The network transmits the temperature data to The Things Network (TTN), which then forwards the data to ThingSpeak.

1.3.7 Data Integration on ThingSpeak

The data pathways for the Arduino MKR WiFi 1010 and the Dragino LHT65 sensors converge at ThingSpeak, where the platform consolidates and visualizes the information. This allows users to view detected bird collisions, input window-specific parameters, and analyze the heat flow rate through the window.

1.3.8 External Temperature Data Pathway

The data pathway for the ambient air temperature outside the building is fetched from a weather report website (weather.eos.ubc.ca). A ThingSpeak MATLAB-script is used to retrieve this data, where it is then integrated with the data from the Arduino and Dragino sensors.

1.4 State Transitioning and Tasks

In the bird collision detection system, we leverage the capabilities of FreeRTOS to replicate concurrent threading, ensuring that multiple tasks run seamlessly and efficiently.
FreeRTOS is an open-source real-time operating system designed specifically for microcontrollers, such as the Arduino MKR WiFi 1010 used in this project. The operating system allows for the efficient scheduling and management of tasks, providing a framework for handling real-time events and system states. By assigning priorities to the various tasks involved in our system, we effectively manage the execution of these tasks concurrently, allowing for a seamless transition between states. High-priority tasks, such as collision sampling, are given precedence over lower-priority tasks, such as idle waiting.
In the following section, we will discuss the task scheduling model for our bird collision detection system, detailing the different tasks the system executes, their functionalities, and their priorities.

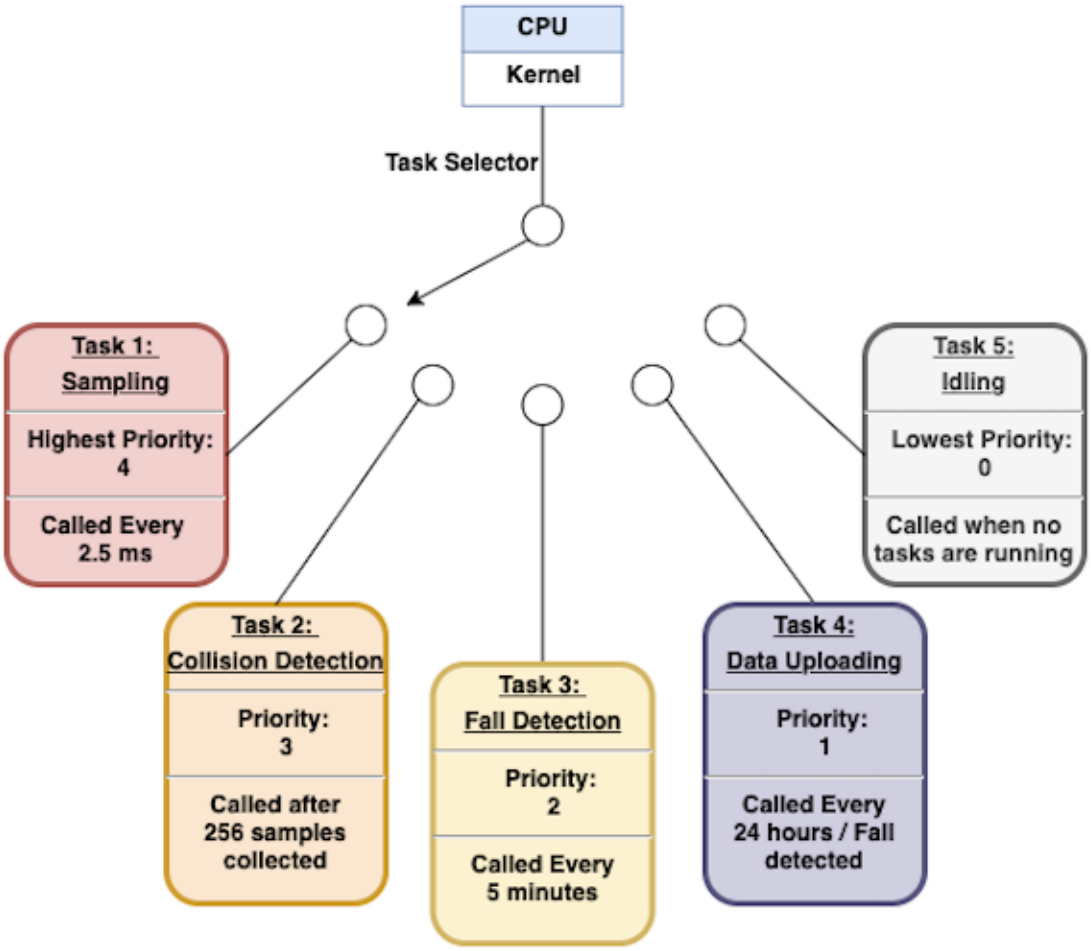


Figure 4: Kernel Task Selector.

Figure 4 illustrates the interaction between the CPU, the kernel, and the five tasks in our bird collision detection system. The diagram aims to provide an overview of how tasks are selected and executed by the CPU, with the help of the kernel.

1.4.1 Idle Task

The Idle Task is the lowest priority task that runs when no other tasks are being executed. It is executed at the beginning of the system start-up, before any sampling starts, and serves as a default state during the running of the system when other tasks are not active.

1.4.2 Sampling Task

The Sampling Task is the highest-priority task that samples the accelerometer at a fixed sampling frequency. It is designed to run concurrently with other tasks, allowing the system to maintain data collection even during the execution of other tasks.

1.4.3 Collision Detected Task

This task is called every 256 samples and is responsible for checking the collected samples for possible collisions using the collision detection algorithm. If a collision is detected, the task updates the collision-array with the time of the collision and then terminates. It runs concurrently with the Sampling Task, ensuring continuous data collection even while processing collision events.

1.4.4 Accelerometer Fall Detection Task

The kernel executes this task every 5 minutes, checking for a fall. The y-axis of the accelerometer is checked against a threshold. If the y-axis value surpasses the threshold, the system suspends all other tasks and calls the Uploading Task to email the client about the fallen accelerometer. It then waits for the reset button to be pressed before allowing other tasks to resume. If no fall is detected, this task terminates, until called again.

1.4.5 Data Upload Task

The Uploading Task is called every 24 hours to upload the collision data stored in the collision array. It is also called when the Fall Detection Task detects a fallen accelerometer; emailing the client that the accelerometer has fallen from the window.

1.4.6 CPU and Kernel

At the core of the system is the CPU, responsible for executing tasks, and the kernel, which manages the tasks and their scheduling. The kernel plays a crucial role in coordinating the operation of all tasks, ensuring that they are executed in the correct order and with the right priority.

1.4.7 Task Selection

The kernel determines which task should be executed next. The task scheduler considers the priority and state of each task and selects the highest-priority task that is ready to run. This ensures that high-priority tasks, such as the sampling task, are executed at the correct sampling frequency and with minimal delay.

1.4.8 Task Execution

Once a task has been selected, the CPU executes the task according to its corresponding instructions. During the execution, the task may transition between different states, such as running, waiting, or suspended, depending on the task requirements.

1.4.9 Task Interaction

Figure 4 also highlights the relationships and dependencies between the five tasks: idle, sampling, collision detection, fall detection, and uploading. These tasks interact with each other through shared resources, inter-task communication, and synchronization mechanisms provided by the kernel.

2.0 Detection System

Bird collision detection subsystem is comprised of two components: sensor and detection algorithm. When a bird-window collision occurs, the impact force imparted by the bird causes a vibration across the window panel. As shown in Figure 2, the sensor picks up the vibration, and transmits signal to the microcontroller. On the microcontroller, the received signal is processed through the detection algorithm to check if the impact is indeed a bird collision, or if it is a false positive (e.g., tree branches, human, or random objects hitting a window). Functional requirement ***F1***, and non-functional requirement ***NF1*** capture the goal of bird collision detection system. See Requirements section for details.

Section 2.2 summarizes the accelerometer sensor used for collision detection. The following section 3.0 summarizes the developed detection algorithm that utilizes the wavelet denoising and a simple score-based detection logic.

2.1 Summary of Previous Capstone Design

As a starting point, the detection system design used by the previous Capstone team is reviewed [7]. Table 2 summarizes the key design choices made by the previous Capstone team.

Table 2: Summary of bird collision detection system by the previous Capstone team.

Sensor	
Sensor Type	3-axis analog accelerometer (only 1 axis data used)
Sensor Cost	$16
Sensor Placement	Attached to a window with duct putty
Sensor Location	Minimum 7.26cm (3inches) away from window frames
Detection Window Size	Undefined or undocumented
Detection algorithm	
Data Processing Methodology	Voltage thresholding, Frequency peak check, and impact duration check

2.2 Accelerometer

This section summarizes the sensor selected for collision detection.

2.2.1 Sensor Specification

The following table summarizes the minimum design specification identified based on preliminary research and testing, and the finalized sensor specification. Details and justifications are provided below.

Table 3: Specification for a sensor used in bird collision detection.

Criteria	**Design specification**	**Adafruit Industries LLC, ADXL343 Evaluation Board**
Sensor Type	3-axis accelerometer	3-axis MEMS accelerometer (only 1 axis data used for collision detection)
Sensor Cost		$8.72
Measurement Range (g-force)	Minimum $\pm 15g$	$\pm 16g$

Detection Window Size	TBD	Tested on 180cm by 80cm window
Sensitivity	TBD	$3.9mg/LSB$
Bandwidth	Cover 0-250Hz range	200Hz (up to 1.6kHz)
Operating Voltage	3.3V or 5V	3.3V
Operating Temperature	-20 to 40 degrees Celsius	-40 to 85 degrees Celsius
Interface	Analog or digital	Digital (Connected via I2C)

2.2.2 Sensor Type

Accelerometers are chosen over other types of sensor technology (refer to section 2.5 for other sensors considered). Accelerometers are relatively low cost compared to other sensor types and are often deployed in collision detection applications [8].

Just like the previous Capstone design, one axis data is used for collision detection (see section 3.0 for more details on detection algorithm). Additionally, another axis data is used for fall detection of the accelerometer as described in section 4.0. Although this still leaves one axis not being used, a 3-axis accelerometer is used as two-axis accelerometers are less common on the market, and cost more than a typical 3-axis accelerometers.

Among other accelerometer options, ADXL343 Evaluation Board is selected for its 1) lowest cost of CA$8.72, and 2) lowest RMS noise performance of $\sim 6.7989mg$. See section 2.4 for other accelerometers considered. Evaluation board (or breakout board) is a small PCB that contains the accelerometer chip, and components such as resistors and capacitors required for signal conditioning and/or for communication. Many commercial evaluation boards, including ADXL343 Evaluation Board, come tested at the factory [9]. The use of evaluation board saves substantial development time and cost to design a PCB and validate the accelerometer functionalities.

2.2.3 Measurement Range

Measurement range refers to a range of acceleration a particular accelerometer is designed to reliably measure. It is expressed in g-force where 1g is equivalent to Earth's gravity ($1g = 9.81m/s^2$).

To determine the measurement range required for our application, the g-force imparted by birds upon collision is estimated. The previous Capstone team has identified Golden-crowned kinglet as a bird species affected by the window collisions and estimated the impact g-force using kinglet data [7].

According to the study conducted by De Groot et al., however, the following five bird species had the highest number of bird collisions throughout their 225-day study period, January 2015 to March 2017 [10]:

1. Varied Thrush
2. American Robin
3. Fox Sparrow
4. Darkeyed Junco
5. Golden-Crowned Kinglet

Impact g-forces are therefore estimated for these five species. See Appendix II for more details. Based on the estimation, the minimum measurement range required for our application is $\pm 3g$, which is the same as the previous Capstone design. However, enDAQ Blog suggests a margin that is 5x larger than the minimum specification, so $\pm 15g$ is used to provide margins around the estimated measurement range [11].

As shown in Table 3, ADXL343 accelerometer is capable of measuring $\pm 16g$ range.

2.2.4 Detection Window Size

The previous Capstone team did not define or document the maximum window size their design can operate. Various window sizes are identified across UBC Vancouver campus buildings. Depending on the sensitivity of the accelerometer, sensor location, and impact location, accelerometer may not pick up the impact that happens far away from the sensor. This is a possible scenario particularly on large windows.

Due to the time constraints, and the lack of data about impact vibration dissipation across a window surface of varying sizes, maximum detection window size is not finalized in our design. The challenge with determining the maximum detection window size is the impact force used for estimation. If the minimum estimated impact g-force is used, then the maximum detection window size is very small, as the smaller impacts get drowned in the noise faster. On the other hand, if the maximum estimated impact g-force is used to determine the maximum distance away

from the sensor at which the impact is not picked up by the sensor, then the maximum detection window size is very big and cannot reliably detect minimum impact g-force.

2.2.5 Sensitivity

Sensitivity depends on the estimated impact force. ADXL343 accelerometer has the sensitivity of $3.9mg/LSB$, meaning the smallest increment of g-force it can measure. This, however, is not entirely accurate in real accelerometers. Accelerometers are susceptible to thermos-mechanical noise internally, which impacts the "effective" sensitivity of the accelerometer. This is discussed further in section 2.4.

2.2.6 Bandwidth

Bandwidth refers to the frequency range that the accelerometer can accurately capture. According to the previous Capstone team, a typical bird collision results in spikes in frequencies between 0-200Hz, which is used to identify bird collisions in the previous Capstone design [7, p. 27]. In our preliminary accelerometer testing, the frequency response of simulated bird-window collisions ranged from 8-250Hz. The test is done using the same test window the previous Capstone team has used.

Depending on the sampling rate, or the output data rate (ODR), the ADXL343 accelerometer has the bandwidth of up to 1600Hz. As discussed in section 3.2, the bandwidth of the accelerometer does not need to cover the entire frequency spectrum associated with the simulated bird-window collisions for our detection algorithm to function.

2.2.7 Operating Voltage

3.3V or 5V is specified to ensure compatibility with a typical microcontroller. ADXL343 accelerometer operates at voltage between 2.0V and 3.6V [12], and therefore can be powered by the Arduino MKR WiFi 1010 (which runs on 3.3V) without extra power management circuitry. See section 7.0 for details on microcontrollers.

2.2.8 Operating Temperature

The sensor is attached onto a window surface on the inside of the building at UBC Vancouver campus. -20℃ to 40℃ is chosen as the operating temperature based on constraints ***C2***. As shown in Table 3, ADXL343 accelerometer can safely operate within this temperature range.

2.2.9 Interface

Accelerometers are interfaced with microcontrollers using either analog or digital pins. Analog accelerometers require Analog-to-Digital Converter (ADC) to digitize analog output voltage. Digital accelerometers perform analog-to-digital conversion internally on the accelerometer chip; many digital accelerometers output data via SPI and/or I2C communication protocol. ADXL343 accelerometer is a digital accelerometer. Data is read by the microcontroller via I2C communication protocol (see section 3.2 for details on sampling). This eliminates the dependency on the on-board ADC on the microcontroller, which provides flexibility with the microcontroller choice in future iterations.

2.3 Limitations and Trade-offs

Selected accelerometer, ADXL343, is unable to detect the estimated minimum impact g-force discussed in Appendix II due to the following two reasons:

1. The internal noise of the accelerometer is not low enough to resolve the minimum estimated g-force.
2. The external environmental noise is higher than the minimum estimated g-force.

Figure 5 shows a typical noise signal collected by ADXL343 at 400Hz sampling rate, $\pm 16g$ measurement range, mounted on a test window of ~180cm by 80cm.

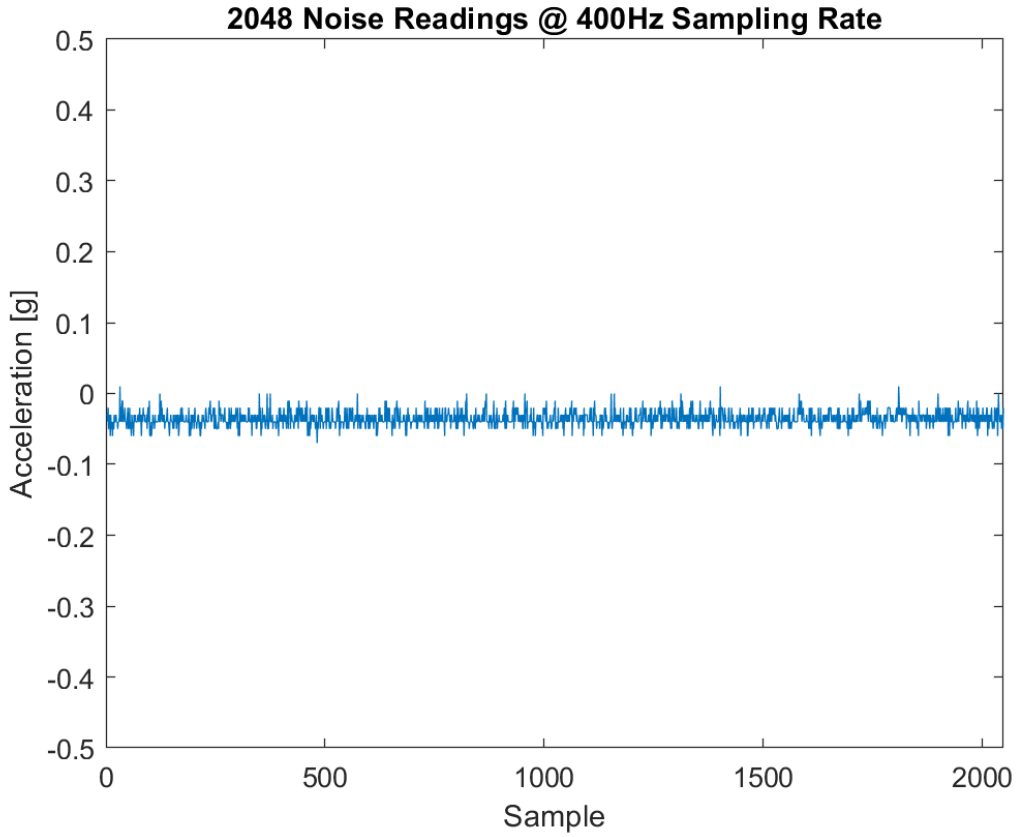


Figure 5: Typical noise readings at 400Hz sampling rate.

Peak-to-peak value of this signal is measured to be $80mg$, which is substantially larger than the minimum impact force estimated ($\sim 1.855mg$), meaning such impact is inevitably drowned in the noise. This limitation stems from the lack of sources on the minimum speed of birds used for minimum impact g-force estimation.

Additionally, the external environmental noise contributes to this signal. The root-mean-squared value of this noise signal is $\sim 37.2mg$, which is approximately 5x larger than the estimated (internal) RMS noise as described in the next section. This suggests the external noise is larger than the estimated minimum impact g-force, and that improving the internal noise performance of the accelerometer does not resolve this limitation.

2.4 Other Accelerometers Considered and Noise Estimation

This section briefly discusses the other accelerometer options considered.

Based on the sensor specification defined in Table 3, two other accelerometers are initially selected as shown below. These sensors are selected to cover a wide range of measurement

range, and sensitivity. See Appendix III for a full list of selected sensor candidates and their details.

Table 4: Selected accelerometers.

Name	**Measurement range [$\pm g$]**
SparkFun Electronics, STMicroelectronics H3LIS331DL Evaluation Board [13]	100,200,400 selectable
SparkFun Electronics, STMicroelectronics LIS331HH Evaluation Board [14]	6,12,24 selectable

To determine the optimal sensor for our application, the effective resolution of the accelerometers is estimated following the guidelines by NXP Semiconductors [15].
The effective resolution refers to the minimum impact g-force an accelerometer can measure without the sensor's internal noise interfering. Figure below shows the effective resolution of two accelerometers above, as well as ADXL343 accelerometer.

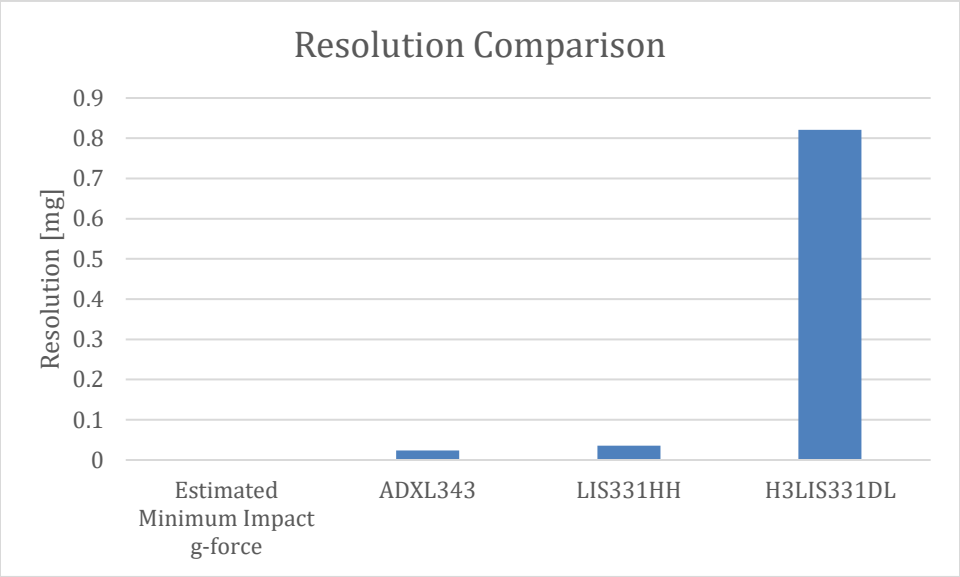


Figure 6: Comparison of effective resolutions of three accelerometer candidates and the estimated minimum impact g-force.

As the resolution of three accelerometer candidates are not sufficient to resolve the estimated minimum impact g-force, the effective resolutions of three industrial-grade accelerometers are calculated to evaluate the performance improvement.

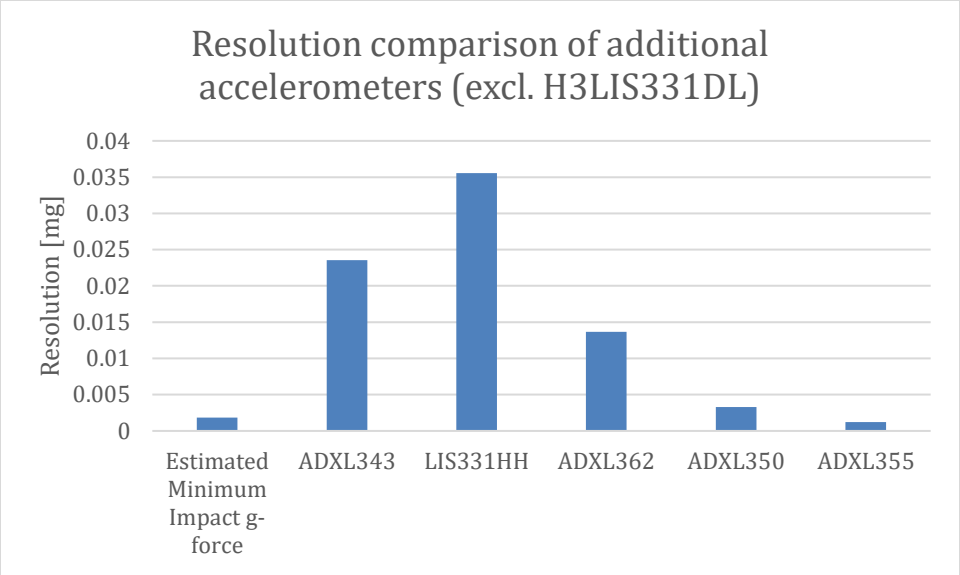


Figure 7: Comparison of effective resolutions of additional accelerometers with the estimated minimum impact g-force.

As shown in the figure above, high-performance accelerometers such as ADXL355 is theoretically capable of resolving the estimated minimum impact. However, these accelerometers are not chosen for this project for the following two reasons:

1. The estimated minimum impact is based on an educated guess of minimum flying speed, and therefore has room to make tradeoff between minimum detectable impact, and the cost.
2. The industrial-grade accelerometers are more expensive. For instance, ADXL355 accelerometer evaluation board costs $65.88 [16].

Among the initial three accelerometer options, H3LIS331DL accelerometer clearly exhibits the worst noise performance, which is expected as it is designed for high-g applications such as crash detection, and therefore is not suitable for our application. ADXL343 accelerometer has the better resolution than LIS331HH accelerometer, and thus is selected for our project.

2.5 Other Sensor Technology Explored

This section briefly discusses other sensor technology considered for collision detection. The following four additional sensor types are initially identified as potential candidates:

1. Camera: capturing collision footage
2. Microphone: capturing the air vibration (sound) of collisions
3. Force sensor: capturing the impact force

4. IR sensor: sensing presence of bird on collision surface

Candidates 3 and 4 are immediately eliminated. Force sensors are designed to sense the force applied onto them, and not around them. Since the collision can occur at any point on a given window, we cannot guarantee that the collision happens on the force sensor without covering the entire surface with force sensors. For IR sensors, they need to be placed on the outside of the window as identified by the previous Capstone team [7, p. 34]. The issue that arises from placing the IR sensor outside is that the infrared light used for sensing can be absorbed by the rainwater; therefore, we cannot always guarantee the proper operation of IR sensors on the outside.

The remaining two sensor types are compared against the accelerometer, and eliminated based on the three criteria: cost, ease of integration, and accuracy. Cost requirement is captured in the constraint ***C1***, while the accuracy of the detection system is captured in the functional requirement ***NF1***. Ease of integration refers to the relative difficulty of R&D activity required to integrate the given sensor as the centerpiece of the collision detection system.

Cost and accuracy criteria are given the equal weight of five as they are both requirements/constraints. Ease of integration, on the other hand, is given the weight of four. Each criterion is then scored out of five. The resulting decision matrix is shown in Table 5 below.

Table 5: Decision matrix for sensor type selection.

Sensor type	Cost (5)	Ease of integration (4)	Accuracy (5)	Score
Camera	4	2	5	53
Microphone	5	4	4	61
Accelerometer	5	5	4	65

Camera is scored four out of five for cost, as camera modules tend to be slightly more expensive than the other two sensor types (ranging $10 - $70) [17]. Although both microphones and accelerometers can be as expensive as camera modules, they can be had for less than $10.

For ease of integration, camera and microphone scored lower than accelerometer. The use of camera necessitates a development of image recognition algorithm. This is discussed in section 3.4 as well. Microphone is scored four due to the lack of prior works available that utilize the microphone as the main sensor in the collision detection system. Accelerometer, on the other hand, is given a score of five as there are prior works that validate the accelerometer's performance in collision detection applications, serving as resources that aid the development in our project.

Lastly, camera is given a score of five for accuracy. Assuming the adequate image recognition algorithm is developed, the use of camera improves the detection accuracy, as each collision can be verified with recorded image/video. Microphone and accelerometer are more susceptible to environmental noise, which increases the potential false detection rate; however, their accuracies are also improved with accompanying detection algorithm, granting them the score of four. Based on the decision matrix, accelerometer is selected as the sensor for detection system in this project. Future iteration of the project may choose to re-evaluate these sensor options.

3.0 Detection Algorithm

The following sections summarize the developed detection algorithm used to identify bird-window collisions.

3.1 Analysis of Previous Capstone Design

Figure 8 below shows the algorithm used in the previous Capstone design.

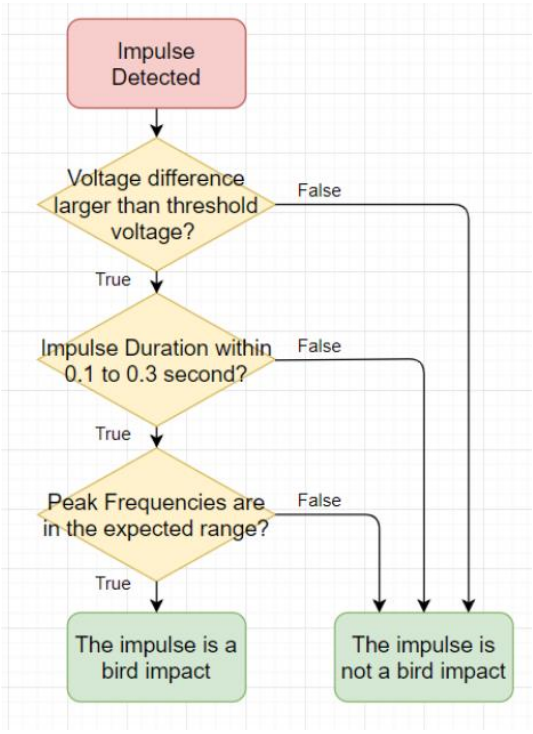


Figure 8: Detection algorithm used in the previous Capstone design.

This algorithm utilizes a combination of 1) voltage thresholding, 2) signal duration check, and 3) frequency peak check using Fast Fourier Transform [7]. When all three characteristics of an

impact signal matches the expected values, the algorithm identifies that the bird-window collision has occurred. Strength and weaknesses of this algorithm are identified below:

Table 6: Strength and weaknesses of detection algorithm in previous Capstone design.

Strength
Checking for signal duration may allow the algorithm to distinguish shorter impulse event (such as noise) from actual collisions.
Weakness
Thresholding is done without filtering/denoising. Lower energy impact may be drowned in the noise.
Frequency peaks may vary based on window size.

3.2 Detection Algorithm Design

Based on the potential for improvement in previous Capstone design, a new collision detection algorithm is devised. Figure 9 shows the high-level flowchart of the detection algorithm. The algorithm is broken up into four stages. Each stage is described in detail in the following sections.

1. **Sampling:** collecting vibration signal from accelerometer.
2. **Pre-processing:** denoising signal using wavelet denoising.
3. **Processing:** a score-based detection logic that checks for nearly consecutive data points above/below the threshold, and infers a collision based on the number of such data points.
4. **Output:** if collision is detected, fetch current time as the time of impact.

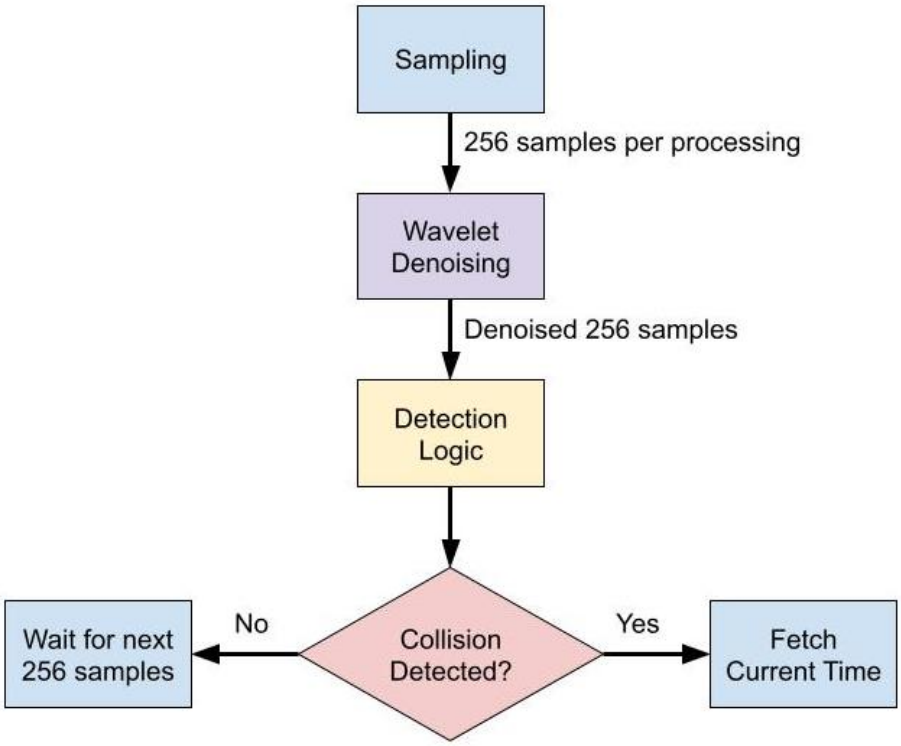


Figure 9: Flowchart of detection algorithm.

3.2.1 Sampling

First, the vibration signal is collected using the accelerometer. Since ADXL343 is a digital accelerometer, the accelerometer itself handles the analog-to-digital conversion. Three design decisions are involved in this stage: 1) sampling rate of the signal, 2) buffer configuration to store accelerometer data, and 3) overlapping samples to avoid loss of information in pre-processing stage.

Sampling Rate

The accelerometer is sampled at 400Hz (every 2.5ms), and new data point is stored in the data buffer. Nyquist-Shannon sampling theorem states that the sampling rate needs to be at least 2x the highest frequency component of the signal to accurately capture a signal [18]. As mentioned in section 2.2, the highest frequency component of typical impact signal is ~250Hz; therefore, sampling rate of 500Hz is theoretically required at minimum to ensure that the impact signal is accurately captured. 400Hz sampling rate is technically smaller than the Nyquist rate of ~500Hz, resulting in undersampling.

The justification for this trade-off is that:

- Our detection logic does not perform frequency analysis, and instead uses g-force magnitude information to infer an impact. As long as the accelerometer reading is sufficiently above the typical noise floor upon impact, the detection algorithm is able to detect a collision even when some frequency components are lost due to aliasing. As shown in Figure 10, sampling rate of 400Hz still captures the characteristics of impact signal (i.e., decaying oscillation).
- Undersampling at 400Hz results in less noisy data. As shown in Figure 10, the signal sampled at 400Hz contain smaller noise variation when no collision is happening.

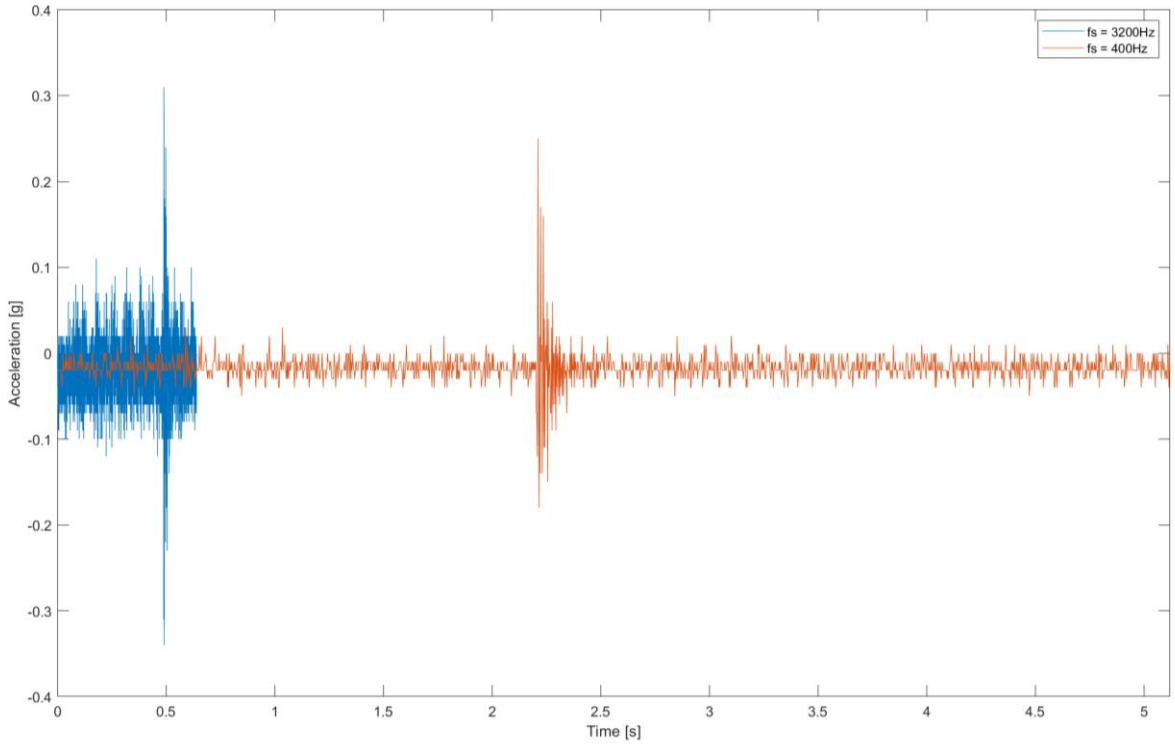


Figure 10: Comparison of a simulated impact signal (dummy weight = 50 grams) sampled at 3200Hz and 400Hz. 2048 data points are collected at each sampling rate. The signal sampled at 3200Hz is shorter because it takes less time to collect 2048 samples. Note that the signal sampled at 400Hz still captures a clear decaying oscillation signature of simulated bird-window collision despite not meeting the Nyquist-Shannon sampling theorem.

Buffer Configuration

Detection algorithm utilizes two 256-sample buffer arrays. At any given point in time, one buffer is serving as a "data buffer" where the latest accelerometer readings are actively stored at 400Hz. At the same time, another buffer is serving as a "processing buffer" where the microcontroller is

actively processing its content through wavelet denoising, and detection logic described in the following section [19]. Figure 11 shows the flowchart of this process.

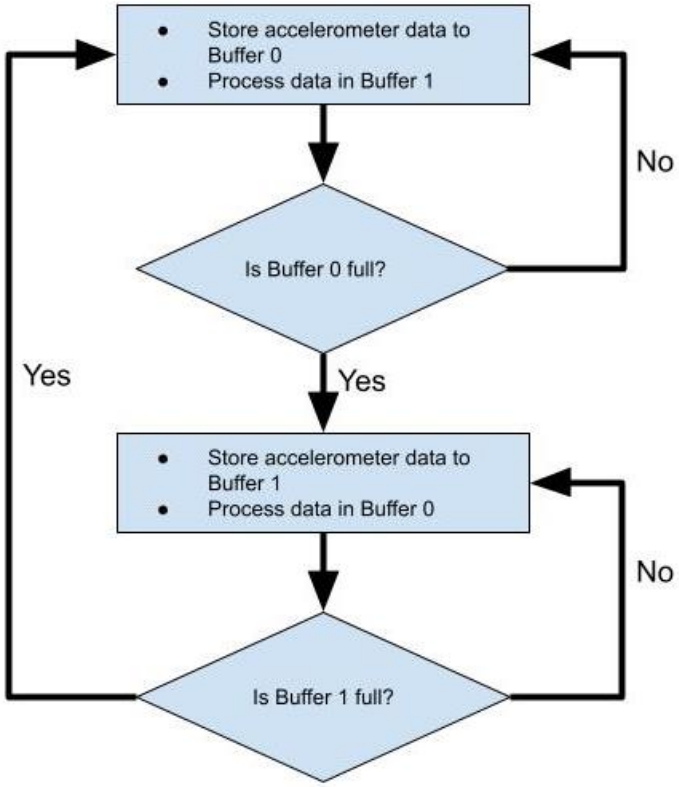


Figure 11: Flowchart depicting how dual-buffer configuration works. At any point in time, one buffer is used as a data buffer, while data in the other buffer is processed by the detection algorithm.

This "dual buffer" configuration is necessary to avoid overwriting part of the data that is actively being processed by the detection algorithm. When 256 data points are collected, without the second buffer, the sampling task described in section 8.2 keeps sampling and starts overwriting the array while the data is still being processed.

The main advantages of dual-buffer approach are:

- Detection algorithm can have a more relaxed timing constraint. Without the second buffer, detection algorithm is required to start processing after 256 samples are collected, and finish processing before the next sample (257^{th} sample) arrives (i.e., timing budget of 2.5ms). Dual-buffer configuration gives the detection algorithm $640ms$ ($2.5ms \times 256\ samples$) to finish processing and enables task scheduling using FreeRTOS described in section 8.0.

- Storage use on microcontroller is more predictable, as both buffers are allocated at compile, with no dynamic memory allocation involved.

Overlapping Samples

Wavelet denoising described in the next section is implemented using filters, and therefore inevitably introduce filter delay to the denoised signal. This results in denoised signal delayed by the total filter delay compared to the original accelerometer data. Since the total number of samples are kept constant (i.e., 256), there are certain number of signals lost at the end of each frame ("pushed" out of the frame by the filter delay).

The discontinuity is particularly not desirable when the impact signal happens across two (or more) frames, which could skew the result of the score-based detection logic. Fortunately, the filter bank is designed in such a way that the filter delay is easy to compute (total filter delay is 15 samples). To counteract the border effect, the consecutive frames are overlapped by the number of total delay samples as shown below:

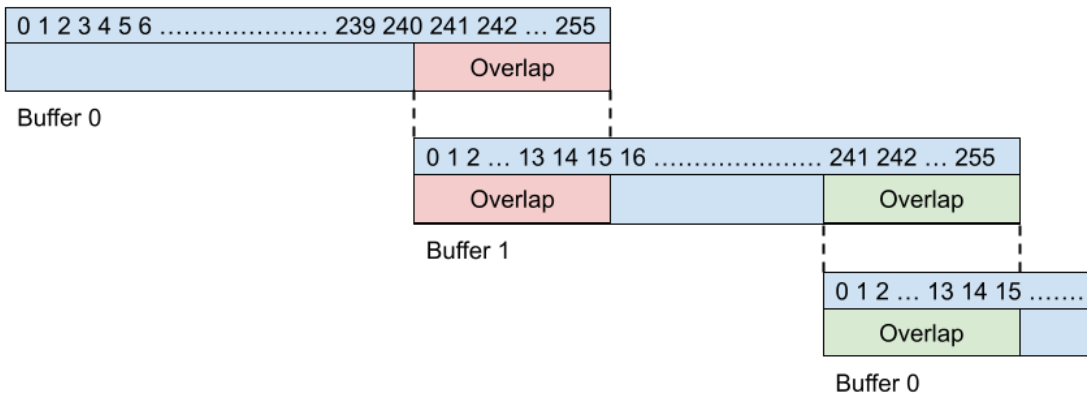


Figure 12: Overlapped sampling. The last 15 samples of current data buffer, and the first 15 samples of the next buffer are overlapped to avoid loss of information.

3.2.2 Pre-processing

Sampled signal may contain an impact signal indicating a bird impact has occurred, but also contains noise due to different factors. Preliminary research shows that the vibration signal from accelerometer is contaminated by pulse noise and white noise [20]. Accelerometers also naturally fluctuate slightly when no acceleration is applied to it (known as "zero g offset") [21].

Noisy signals pose a challenge when detecting a lower energy impact; therefore, noise component should be removed as much as possible before further processing is performed. Wavelet denoising is a technique that makes use of wavelet transform, which has been shown to handle nonstationary signals well [22]. Nonstationary signal is a type of signal whose frequency response changes over time. This applies to our signal of interest, where the bird-window impact is an infrequent event and persists for a very short period of time compared to the total number of hours the device is operational.

Wavelet denoising filter bank is implemented by utilizing FIR Decimator and FIR Interpolator functions in ARM CMSIS DSP library [23]. CMSIS library is an open-source library with a set of functions and APIs developed for ARM Cortex-M microcontrollers [24]. Arduino used in our project, Arduino MKR WiFi 1010, uses SAMD21 Cortex-M0+ microcontroller, and therefore is compatible with CMSIS library.

The following table summarizes the key wavelet denoising design choices and justifications:

Table 7: Summary of important wavelet denoiser parameters and justifications.

Parameters	**Design Choice**	**Justification**
Wavelet Basis Type	Bior 2.2	Filter bank designed using biorthogonal wavelet has a linear phase delay, which is needed to compensate for delays in real-time applications [25].
Number of denoising levels	2	Theoretical limit is $floor\big(log2(N)\big) = 8$ where $N = 256$. More levels result in smoother denoised signal, but also increase execution time per processing, and introduce longer filter delay. Moreover, the improvement in denoising for level > 2 is minimal and hard to justify.
Thresholding method for noise coefficients	Hard thresholding	Hard thresholding result in better denoising on the noise portion of the signal, while preserving the majority of the impact signal component unmodified.

3.2.3 Processing

Once pre-processing is complete, the denoised signal is checked for potential bird-window collision. The detection logic consists of 1) thresholding and 2) a score-based scheme that checks for nearly consecutive data points above threshold.

Thresholding is the simplest way to determine if a collision has occurred based on the accelerometer readings, since a collision results in irregular (i.e., larger reading than typical) data points. Thresholding alone works well in simple cases; however, it can falsely recognize random spikes in sensor reading (which can be due to random noise spikes, or someone tapping the window) as collision. A typical simulated bird-window collision results in a signal with multiple (nearly) consecutive data points that are above threshold, as opposed to just one random spike. Those (nearly) consecutive data points eventually decay out and sensor readings return to typical noise level.

A score-based scheme is implemented to take advantage of this nearly consecutive data points:

1. First, it initiates a detection logic when a data point is above the detection threshold for the first time.
2. It then counts the number of nearly consecutive data points that are above threshold.
3. The detection logic keeps counting (i.e., scoring) until there are certain number of exactly consecutive data points that are below threshold. This ensures that the impact signal has completely decayed out, and that the readings are back to the typical noise level.
4. If the score is above a certain value, that means that there were multiple nearly consecutive data points that were above the threshold, which likely means that the impact has occurred. Otherwise, it likely was a random noise spike, and can be disregarded.
5. Once the output of the detection logic is picked up by the appropriate process, the detection logic resets the score. It will wait for the next instance where the data point is above the detection threshold, and repeat.

This detection logic design ensures that 1) the random spikes in accelerometer readings are not falsely identified as collisions, and 2) nearly consecutive data points above the g-force threshold is considered as one collision instead of individual collisions.

3.2.4 Output

If the detection logic determines that the bird-window collision has occurred, the current time is fetched and used as the time of collision. This time data is eventually sent to ThingSpeak. See section 8.2 for more details.

Note that the time fetched in this stage is not the actual time of the impact. There is a delay between the time the collision data is sampled, and the time that chunk of data gets processed (as discussed earlier, the detection system operates on 256-sample frames per processing). This delay, however, has been measured to be approximately 7-8 sampling intervals, or ~17.5 – 20ms (sampling rate is 400Hz, so one sampling interval is 2.5ms), which is a negligible difference.

3.3 Detection Algorithm Performance

The detection algorithm including wavelet denoising using FIR Decimators and Interpolators are first developed on MATLAB for easier troubleshooting and visualization. Figure 13 shows the demonstration of detection algorithm. See section 2.0 in Verification & Validation for demonstration of detection algorithm deployed on Arduino.

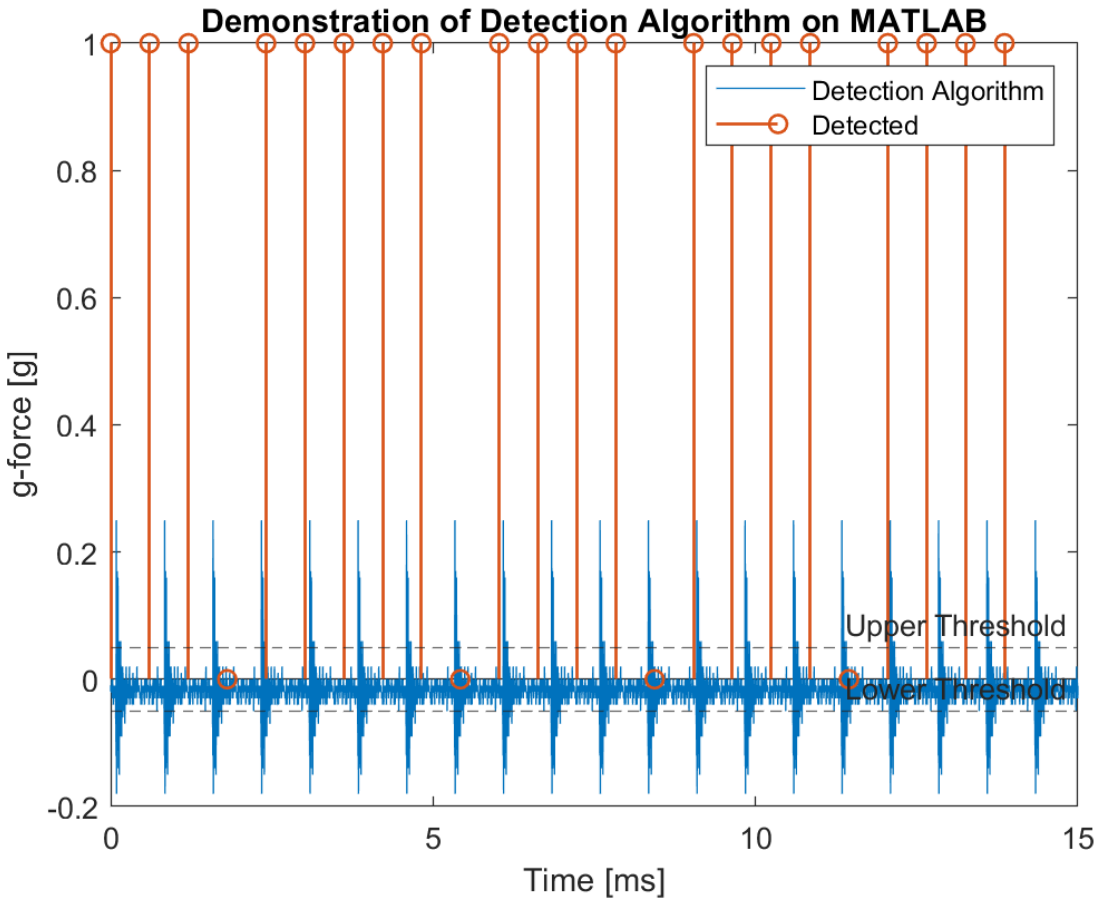


Figure 13: Demonstration of detection algorithm on MATLAB model. The orange spike shows that the collision is detected in the time between the spike and the next orange point.

3.4 Limitations and Other Designs Explored

The main limitation of our detection system design is that it only uses vibration information from the accelerometer. This means that the detection algorithm identifies a collision as long as the expected signature (i.e., decaying oscillation) is present in the signal. That is, our detection system identifies any object with enough mass to cause such vibration as a bird-window impact. The fundamental assumption of our detection system is that the detection window (window used for collision monitoring) is free from people or object accidentally hitting the window.

This is assumption is necessary because the detection system need to be tested by the simulated bird-window collision setup, which consists of a ball made from cloth and rice as a filling. By assuming such object represents a bird-window impact and can be used verify the functionality

of detection system, our detection system also assumes that anything hitting a window is a collision.

An alternative design that has the potential to alleviate this limitation is the use of camera and image recognition to verify if the impact is indeed caused by a bird. This design, however, increases cost and the complexity of the detection system as mentioned in section 2.4.

Another design explored utilizes a neural network model to detect anomaly (i.e., collision) in sampled signals. Collision detection system with neural network classifier developed for wind turbines demonstrates an improvement over a simple thresholding system [8].

Despite its advantages, machine learning approach is removed from consideration due to high R&D overhead. Future improvements to our design could be made by considering machine learning algorithm from the earlier stages of design process.

Lastly, the detection algorithm that use data from all three axes of the accelerometer was considered, inspired by design described in [8].

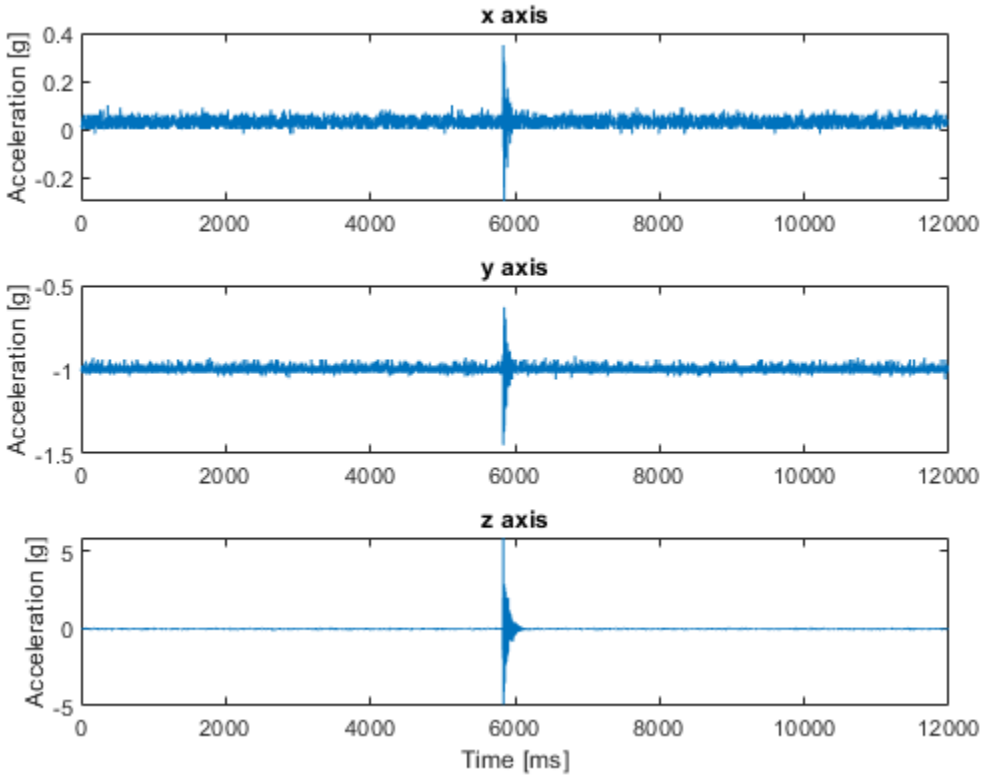


Figure 14: Plots of vibration signals from three axes. Z-axis is orthogonal to the windowpane and therefore has captured the largest magnitude of impact signals.

Figure 14 shows the impact signals from each axis. The axis orthogonal to the windowpane, z-axis, has captured the largest impact signals, while other two axes still contain the signature of small bird-window impact. Signal from other axes may contain a useful impact signal especially when the collision happens at an angle, where the impact vector is more evenly split across three axes. Due to the time constraints, and the fact that the x and y axis impact is significantly smaller than the z axis impact (approximately 1/10x), x and y axis information are not used in our final design.

4.0 Accelerometer Fall Detection Algorithm

Fall detection is a crucial feature for meeting the functional requirement ***F10***, as it ensures the reliability of the accelerometer and the overall system. The fall detection algorithm is designed to identify if the accelerometer has fallen off the window surface, which causes inaccurate measurements.

4.1 Algorithm and Functionality

The fall detection algorithm is designed to be power-efficient and ensure the reliability of the accelerometer and the overall system. It works by periodically monitoring the y-axis accelerometer readings every 5 minutes. This interval was chosen to minimize power consumption and reduce the computational load on the microcontroller.

The algorithm uses a two-step process to detect a fall. In the first step, when the fall detection task is called, it checks the y-axis accelerometer value and sets a flag if the value indicates a possible fall. At this point, fall detection task finishes executing with the flag set. This flag serves as a preliminary indication of a fall and is used in the second step to further confirm the event.

In the second step, when the fall detection task is called again after the next 5-minute interval, it checks the y-axis value again. If the value still indicates a possible fall and the flag set in the first step is still active, the algorithm confirms that a fall has occurred. This two-step process helps to reduce false positives by ensuring that the accelerometer readings consistently indicate a fall over two consecutive checks.

4.2 Design Decisions

In this section, we explore the fall detection design decisions, focusing on power efficiency, reliability, and computational load. We will discuss the periodic execution of the task, the two-step process for improved accuracy, and the reduction of the microcontroller's computational load.

4.2.1 Design Considerations

Power efficiency: As the system can be battery-powered, minimizing power consumption is essential. The chosen design periodically executes the fall detection task, reducing power usage and computational load (see section 11.2, for more details on battery operation).

Reliability: The two-step process improves the reliability of fall detection by minimizing false positives, ensuring that the system only acts when a fall is detected.

Computational load: By executing the fall detection task at 5-minute intervals, the computational load on the microcontroller is reduced, allowing it to perform other tasks efficiently.

4.3 Advantages and Disadvantages

4.3.1 Advantages

Power-efficient: The periodic execution of the fall detection task reduces power consumption.
Reliable: The two-step process minimizes false positives, ensuring accurate fall detection.
Reduced computational load: The 5-minute interval between task executions allows the microcontroller to perform other tasks more efficiently.

4.3.2 Disadvantages

Potential delay in fall detection: The 5-minute interval between task executions may cause a delay in detecting a fall.
Reduced sensitivity: The two-step process might not detect falls with less impact on the y-axis value.

4.4 Alternative Approaches

Several alternative approaches to fall detection were considered during the design process:
Continuous monitoring: This approach involves constantly monitoring the accelerometer readings to detect a fall. While this approach may provide faster fall detection, it increases power consumption and computational load on the microcontroller.
Single-step detection: A single-step detection approach checks the accelerometer value only once to determine a fall. While this method may be simpler, it can result in a higher rate of false positives.
Machine learning-based detection: This approach uses machine learning algorithms to predict falls based on accelerometer data. While this can be more accurate, it requires significant computational resources and may not be suitable for a low-power microcontroller like the Arduino MKR WiFi 1010.
The chosen design balances power efficiency, detection reliability, and computational load, making it a suitable choice for this application.

5.0 Temperature

This section describes the heat detection process in our project and how the collected temperature measurements play a role in the UBC sustainable green building policies. It also describes how the temperature measurement are collected and displayed to the users for further analysis.

5.1 Heat Flow Rate Measurement

We are required to measure the efficacy of windows and how well they mitigate heat flow rate through them to quantify how well they perform against other windows. This design helps with UBC enacting sustainable green building policies. The researchers for whom this device is designed for will study the data provided by temperature sensors on the inside and outside of the surface of the windows being observed. Together with a U-value and the dimensions of the window (that will be provided by the researchers), the heat flow rate is calculated, and this value is used to measure the efficacy of windows and hence quantify the comparisons of different windows on UBC campus.

According to ***F3***, the minimum allowed range we could have for any temperature sensor chosen is between -40C to 50℃ which our temperature sensor satisfies. In addition, our temperature sensor of choice is required to be wireless, connect to internet, waterproof and easily mountable to insure user-friendliness according to ***F2***, ***F3***, ***F5***, ***NF5***.

$$\boldsymbol{Heat\ Flow\ Rate} = \frac{|\boldsymbol{Inside\ Surface\ Temperature} - \boldsymbol{Outside\ Surface\ Tempearture}|}{\boldsymbol{R\ value}} * \boldsymbol{Window\ Surface\ Area}$$

Our temperature sensor connects to ThingSpeak via LoRaWAN and will send two surface temperature packets to ThingSpeak every thirty minutes. We decided to collect the data every thirty minutes as maximum the temperature change recorded in Vancouver is ±2℃ which is a minimal change in a collision detection analysis.

A set of two thermocouple temperature sensors (DS18B20) are selected for gathering the heat mitigation data. One temperature sensor is used on the outside surface of the window one on the inside of the surface of the window. The sensors are touching the surface of the window and are attached to the window using duck puddy as a barrier between the surface of the window and the outside air to prevent the thermocouple from reading the outside air temperature. The

temperatures we chose can read a range of temperatures ranging from -40 to 200°C. This fits in perfectly with the expected temperature of the outside surface of the window.

These two thermocouples are attached to the device called Dragino LHT65 which has a built-in 2400mAh non-rechargeable battery which can be used for more than 10 years. The Dragino sensor includes a built-in SHT20 Temperature & Humidity sensor with an external sensor thermocouple temperature sensor. Moreover, the device records up to 3200 data records with date/time which can be retrieved with ThingSpeak for further analysis [26]. The LHT65 also connects directly with ThingSpeak meaning that no extra microcontroller needs to be placed on the outside the window, eliminating extra safety and technical risks. These temperature values are then be stored in the ThingSpeak database for further analysis.

With the collected surface temperature data, the heat flow rate across the window is calculated using the window surface area and the U-value provided by the user (which is a characteristic of windows) to assess the efficacy of windows mitigating heat transfer, and consequently their efficiency saving energy by keeping the heat trapped in a room. This allows researchers to know which windows to use for building green and sustainable buildings.

Table 8: Hardware/software used to measure heat flow rate.

Function	**Selected Candidate**
Surface Temperature Data Measurement	Dragino LHT65
Heat Flow Rate Calculation	The built-in analytics tools of ThingSpeak to gather data from the UBC weather website

The section 1.0 in Verification & Validation has proven to work, and measurements taken from ThingSpeak showed that our sensors are in fact measuring the temperatures and displaying them for the user.

The section 3.0 in Verification & Validation has proven the pre-set codes for heat flow rate calculation in the MATLAB Analysis can give the accurate calculation result, which is updated and stored every 30 minutes, and displayed in the dashboard of ThingSpeak as expected. More detailed testing processes are recorded in the Verification & Validation Document.

5.2 Other Candidates

As for the other designs that we decided not to move forward with, the first option considered was a heat flux sensor. This method gives the user a U-Value which is a quantitative way of determining the efficacy of windows mitigating heat transfer. However, this method requires a specific DAC that cannot connect to the Arduino. In addition, the heat flux sensor requires a voltage resolution which the Arduino does not offer.

Moreover, one thing that was proven ineffective was using a heat flux paste to use as a middle ground between the thermocouple of the temperature sensor and the window to better conduct the temperature. However, after comparing it to a control thermocouple, both thermocouples output the same value for surface temperature which suggests that the effect that the thermal paste is at best minimal.

Table 9: Complete list of temperature sensor candidates [27], [28].

	DHT11	DHT22	LM335	BMP180	TMP36	LM75	BME280	DS18B20	LHT65
temp									
pressure									
humidity									
Communication protocol	One wire	One wire	Analog output	I2C	Analog output	I2C	i2C or SPI	One wire	ThingSpeak
Power supply range	3 to 5.5V	3 to 6v	4 to 30 v	1.8 to 3.6v for chip 3.3 to 5V for module	2.7 to 5.5 V +-1c	3 TO 5.5 V	1.7 to 3.6 for chip 3.3 to 5v for board	3-5.5 V	2400mAh non-chargeable battery
Temp range	0 to 50c +-2c	-40 TO 80c	-55 to 150C +-0.5c	0 to 65c +-0.5C	-40 to 125 c +-1C	-55 to 125 c +-2c	-40 to 85c +-0.5c	-55c to 125c +-0.5c	-40c to 80c +-0.8
Humidity range	20 to 90% +-5%	0 TO 100% +-2%							0 to 99.9% +-10%
Sampling period	1 second	2 seconds		128 samples/sec	3 seconds		0.21 seconds		
waterproof									

LEGEND

	no
	yes

	NA

6.0 Outside Air Temperature Detection

For researchers to be able to identify the conditions when collisions occur, as per ***F2***, the outside temperature is recorded at the time of impact. The data is collected from the UBC ESB Rooftop Weather Station website.

Table 10: Function and selected candidate for outside air temperature measurement.

Function	**Selected Candidate**
Outside Air Temperature Measurement	UBC ESB Rooftop Weather Station Website

Through the ThingSpeak's MATLAB Analysis function, we programmed MATLAB code to specify the web address string of the UBC ESB Rooftop Weather Station as the URL that we want to scrape the web temperature data. The web temperature data is fetched every 30 minutes, parsed to find the air temperature, and store the data into the data field specified.

The stability and accuracy of codes that used to collect the outside air temperature collection from the UBC ESB Rooftop Weather Station website is verified in the section 3.0 in Verification & Validation.

This has proven to work, and measurements taken from ThingSpeak showed that our sensors and weather website are in fact displaying the correct temperatures.

7.0 Microcontroller

This chapter mainly discusses the role of microcontrollers in the project, as well as the final selection of microcontrollers and the filtering process.

7.1 Significance and Function of Microcontrollers

The microcontroller holds a pivotal position in our project, serving as the central processing unit of our system. It efficiently gathers and processes data from sensors strategically placed on windows, allowing for real-time analysis and provide accurate collision and heat flow rate information to researchers. This data aids researchers in identifying and addressing the multiple

factors contributing to bird-window collisions, supporting the implementation of UBC Green Building Action Plan (GBAP) and the protection of birds and the natural environment.
The microcontroller will interface with an accelerometer and a reset button. The accelerometer captures window vibration data, which is processed by the microcontroller using detection algorithm as described in detection algorithm. In the event of a collision detection, the microcontroller will log the timestamp and transmit the data along with the event count to the user interface at the end of each day. (To optimizes power utilization, the microcontroller will employ a data transmission interval of 24 hours. However, users have the option to increase data frequency by connecting the system to a wall power source and upload an alternative code to make the data transfer interval more frequent, every 5 minutes.)

7.2 Microcontroller Decision

The Arduino MKR WiFi 1010 is chosen as the microcontroller for the project, following a detailed selection process.
The Arduino MKR WiFi 1010 satisfies the highest priority criteria's that are critical to the success of the project, including availability of WiFi which correspond to ***C4***, enough input digital pins, compatibility with power socket module (***C1***), ample flash memory capacity (***NF4***), and cost-effectiveness within the project budget when compared to other microcontrollers. Moreover, its compact size and weight make it ideal for meeting the requirements of ***C5*** and ***C6***. For a comprehensive comparison, please refer to the detailed comparison table provided in Appendix IV.
The stability of Arduino MKR WiFi 1010 has been rigorously tested using various methods, including testing the detection algorithm and data transmission code on the Arduino. The testing results have successfully recorded the time and data of simulated bird-window collisions, which were then transmitted to the user interface. For detailed testing results, please refer to the Verification & Validation section.

7.3 Other Microcontroller Candidates and Screening Process

Eight widely used microcontrollers are assessed using 14 criteria points based on the project constraints. The assessed microcontrollers include:

1. Arduino Uno WiFi Rev2

2. Raspberry Pi 3
3. Arduino NANO 3
4. Arduino Uno R3
5. Arduino Mega
6. ESP32 NodeMCU
7. ESP8266 Thing
8. Arduino MKR WiFi 1010

The 13 criteria points were prioritized based on their essentiality for the success of the project. The weight of each criterion in microcontroller selection was determined by its importance. To facilitate comparison, a decision table was constructed (Appendix IV). In Table 11, the minimum criteria used to filter out unsatisfactory microcontrollers are listed.

Table 11: Minimum Microcontroller Requirements.

Microcontroller Feature	Minimum Requirement
Number / type of I/O (Digital)	4
WiFi Availability	Yes
Flesh Memory	130KB
Power	Could be powered by both battery and 120 VAC wall outlet

7.3.1 Criteria 1: Digital Input Pins

The hardware model of the project requires multiple input pins on the selected microcontroller to receive digital data coming from impact force/vibration readings. As seen in Figure 2, the detection section's collision data, recorded from the vibration sensor, requires, at minimum, 2 digital input pins of the microcontroller to function. In the fall-detection system, two input pins are required for the reset button to receive data. The selection of the microcontroller considered over-designing, allowing for potential future additions or alternative solutions, such as using multiple vibration sensors for larger windows, which would require additional digital input pins. This approach enables scalability and cost reduction in the final design. A detailed comparison of data from different microcontrollers can be found in Appendix IV.

7.3.2 Criteria 2: WiFi and Database Connection

The hardware-to-software connection component of the model necessitates the secure transfer of data over a wireless network, specifically WiFi, as indicated by ***C4***. In line with section 10.0, the chosen database framework is ThingSpeak. To comply with ***NF4***, the microcontroller must consistently transfer data to the ThingSpeak database. According to the findings in Appendix IV, microcontrollers such as Arduino Uno WiFi Rev2, Raspberry Pi, ESP8266 Thing, ESP32 NodeMCU, and Arduino MKR WiFi 1010 fulfill the criteria as they possess WiFi capabilities and a direct link to the database.

7.3.3 Criteria 3: Flash Memory

While the Arduino is operating and executing code, flash memory is needed to store program code and gather data. It shows that the microcontroller used in this project needs to have a flash memory that is big enough to store and evaluate the data needed by the client. Following some experiments, it was discovered that the collision detection system needed a specific amount of flash memory from the microcontroller for data gathering, noise reduction, and processing. At the same time, the microcontroller also needs to transfer the temperature data for calculation, which leads to the microcontroller's flash memory capacity must reach 130KB (See detail calculation in Appendix V). Only microcontroller Arduino Uno WiFi Rev2 from the previous round did not match this requirement and was eliminated, leaving Raspberry Pi, ESP8266 Thing, ESP32 NodeMCU and Arduino MKR WiFi 1010 with flash memory levels that were far greater than the minimum necessary and provided the project with considerable fault tolerance.

7.3.4 Criteria 4: Price and Power Socket

In order to satisfy ***C1***[4], and ***NF4***[5], the microcontroller candidates' prices and power supply need to be carefully selected. Based on the empirical findings from research references [29], [30] and detailed calculations, it has been shows that the utilization of a 3200mAh battery, with a data

[4] For several of the model to be used for bird strike monitoring, the whole project cost (including hardware, software, and auxiliary supplies) needs to choose the most economical one within the available options and the cost of an entire system should be under $180. [Database must be free for the user.]

[5] The data recording must be fully functional for a minimum of 30 days before requiring maintenance.

transmission frequency of once a day to ThingSpeak, would provide the microcontroller with an operational lifespan of approximately 6 days. The incorporation of batteries into the system enhances its flexibility and availability, allowing users to deploy the device in locations without access to wall outlets (Comprehensive calculations and analysis can be found in Appendix XI). However, for more frequent data monitoring requirements, reliance on power outlets becomes necessary to sustain the device's power consumption. Therefore, the selected microcontroller must possess the capability to receive power from both batteries and external power outlets. Among the remaining microcontroller options from the previous rounds of screening, only Arduino MKR WiFi 1010 and Raspberry Pi 3 demonstrate the ability to accept power from both batteries and external power outlets. Among these options, Arduino MKR WiFi 1010 stands out as a more competitive choice compared to Raspberry Pi 3, considering that the latter is priced approximately four times higher than Arduino MKR WiFi 1010.

8.0 Task Scheduling Algorithm

8.1 System Context for Design and Architecture

FreeRTOS is an open-source, real-time operating system designed specifically for microcontrollers and small embedded systems. It offers a simple, efficient, and lightweight solution for managing tasks, time, and resources, which makes it a popular choice for developers working on IoT and embedded projects [31].

The purpose of using FreeRTOS is that timing constraints can be set for each task, and the whole system has a pre-determined, predictable behaviour. FreeRTOS is responsible for managing and scheduling the tasks that collect, process, and transmit data from the accelerometer, and it simplifies the development of complex systems with multiple tasks, and ensuring reliable operation and efficient utilization of CPU resources in the microcontroller. Through the utilization of FreeRTOS in our system architecture, the requirements of ***F2*** and ***NF4*** are met, enabling concurrent execution, and recording of multiple tasks while ensuring system reliability and smooth operation.

8.2 Design

In FreeRTOS, there are five different tasks assigned to our system, the timing schedule is showed in Figure 15 and Figure 16. The priority of these five tasks gradually decreases from Task 1 to Task 5.

- Task 1 and Task 2 handle collision detection. Task 1 collects data from accelerometer and wakes up Task 2 every time 256 data points are collected for further noise reduction and analysis, to determine if a bird collision event has occurred.
- Task 3 is responsible for checking the accelerometer's proper attachment to the window being tested every five minutes. If the accelerometer is detected to have fallen during the first check, Task 3 will perform a second check after 5 minutes. If the result of the second check is also a fall, Task 3 will stop other tasks, send a message to ThingSpeak to notify the user of the fallen accelerometer, and wait for the user to reattach the accelerometer and press the reset button to resume other tasks. Task 2 will be resume and finish the remaining analysis after the reset button is pressed, and Task 4 will be re-initialized as well. After a delay of five seconds, Task 1 will be restarted, giving Task 2 enough time to complete the remaining work from the previous session. See Figure 16.
- Task 4 is responsible for sending bird collision data and fall detection to ThingSpeak. Depending on user’s needs, Task 4 can run once every 24 hours or once every 5 minutes. Task 4 sends the time and count of bird collision events that occurred during the time elapsed since the last data transmission to ThingSpeak.
- Task 5 is the system's default idle task, which runs when no other tasks can run such as sampling task.

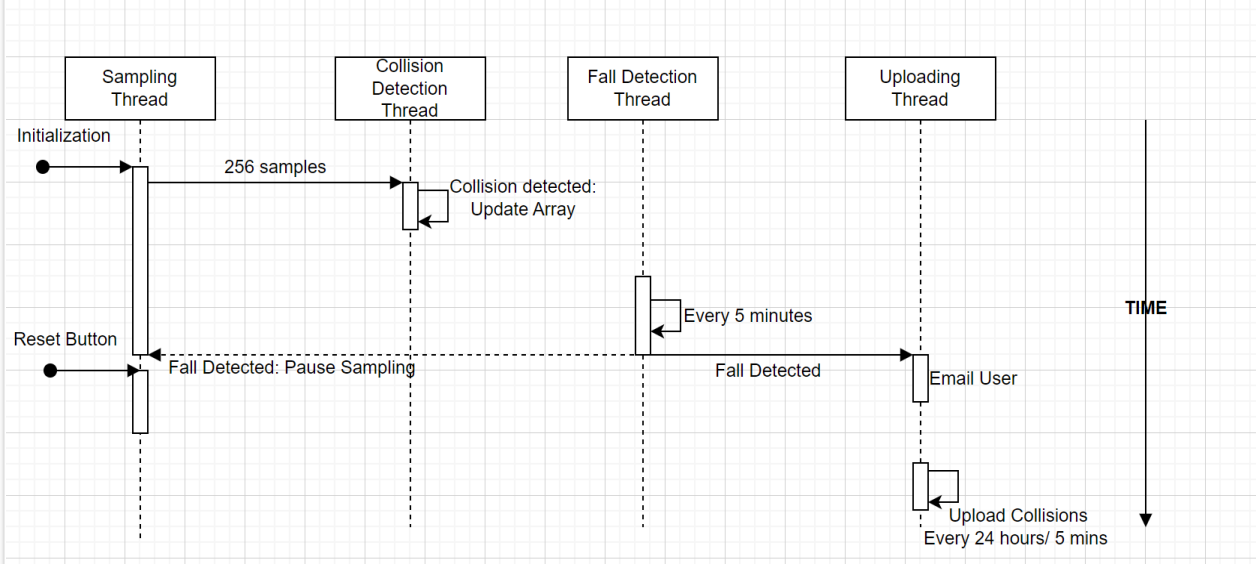


Figure 15: Time scheduling diagram depicting tasks running with FreeRTOS. Solid line means forward operation; dashed line means response.

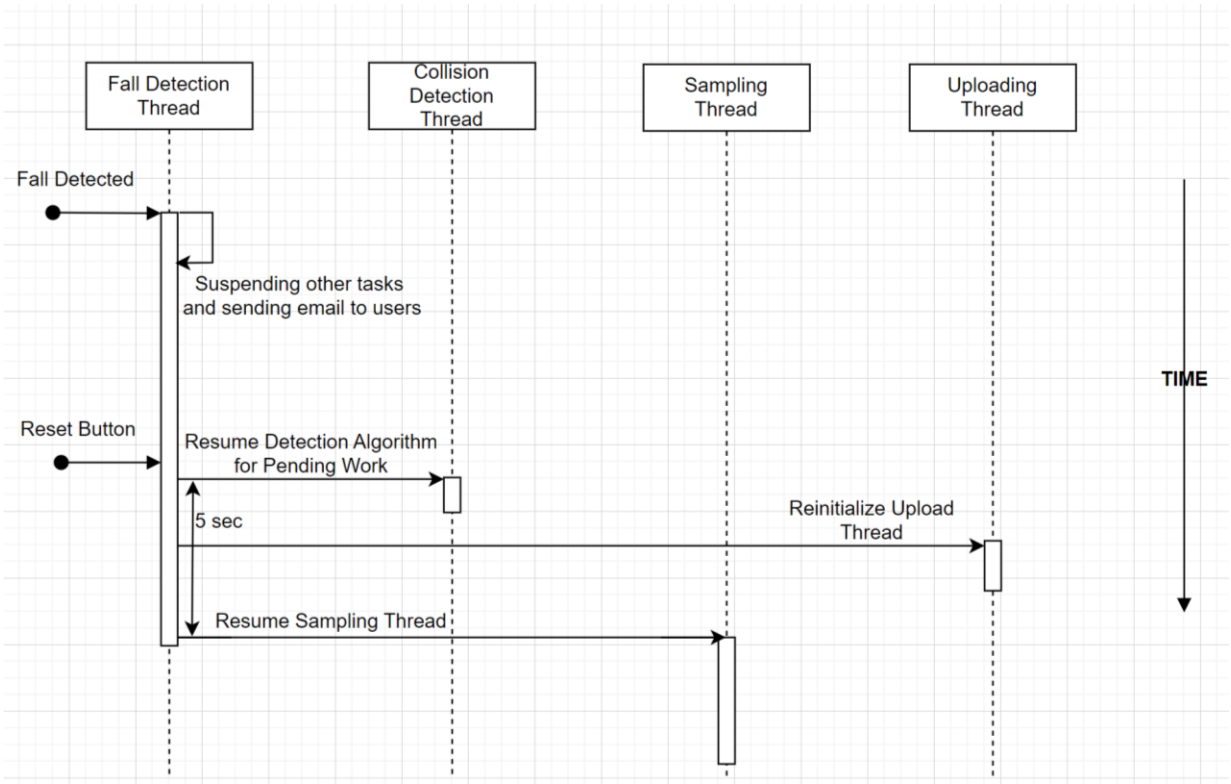


Figure 16: Fall detection detailed Time scheduling diagram.

8.3 Alternative Design for Scheduling Algorithm

The previous design of the bird collision detection system relied on a state machine, illustrated below, by Figure 17, which executed tasks sequentially based on a set of predefined states and

transitions. While this approach was functional, it had certain limitations, particularly when it came to handling multiple tasks simultaneously and efficiently managing resources. One significant drawback of the state machine model was its inability to sample in "real-time," which led to the potential for missed bird collisions if the system was in the wrong state. Likewise, fall detection could not successfully be deployed for this same reason. Further, in a task-based design, tasks are executed concurrently, allowing the processor to perform multiple operations simultaneously. As a result, the processor spends less time idling and waiting for tasks to be executed sequentially, as it would in the state machine model. This efficient use of resources results in a lower overall power consumption, which in turn extends the battery life.

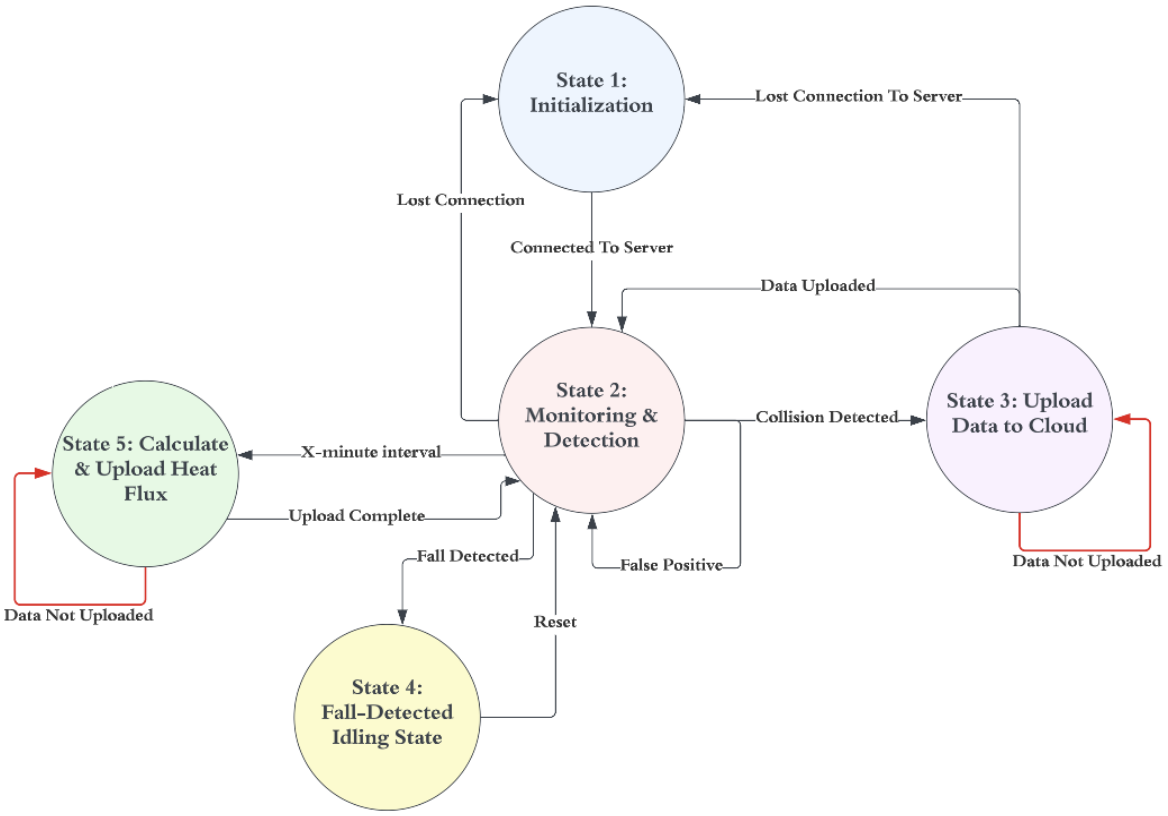


Figure 17: Previous State Machine Design.

To address these shortcomings, we decided to switch from the state machine-based design to a real-time operating system that facilitates concurrent programming. FreeRTOS enables the creation and management of multiple tasks that can run concurrently, effectively simulating parallelism in a single-core microcontroller environment.

8.4 Task Scheduling Performance

The task scheduling and coordination capabilities of FreeRTOS have been thoroughly evaluated, including task execution time, regularity, and potential latency. Test Results Demonstrate FreeRTOS as an effective tool for achieving project objectives with consistent and reliable operation. We tested the state machine functionality of FreeRTOS, as well as the compatibility of FreeRTOS with the detection algorithm and the code for sending messages. For detailed testing results, please refer to the Verification & Validation section.

8.5 Detail Explanation of FreeRTOS Operation

In this section, we will discuss how FreeRTOS works, its advantages and disadvantages, and other alternatives that can be considered for implementing real-time systems.
FreeRTOS employs a pre-emptive, priority-based scheduling algorithm that allows multiple tasks to share a single CPU. Each task is assigned a priority level, and the scheduler ensures that the highest-priority task is always running [31]. When two tasks share the same priority level, they are executed in a rotating sequence, alternating between the tasks in a fair and balanced manner. Tasks can be in different states such as running, ready, blocked, or suspended. A tick interrupt (periodic interrupt generated by a hardware timer in a microcontroller or microprocessor) is used to manage the passage of time and facilitate context switching between tasks. When the tick interrupt fires, the scheduler checks if a higher-priority task is ready to run. If so, the context of the current task is saved, and the new task is executed. This enables effective multitasking and ensures that the system remains responsive to external events.

8.5.1 Advantages of FreeRTOS

Lightweight and efficient: FreeRTOS has a small memory footprint, making it suitable for resource-constrained environments like microcontrollers. It is designed for efficiency and can be easily configured to use only the required features, further reducing memory usage.
Scalability: FreeRTOS is highly scalable, allowing developers to manage a wide range of tasks and resources on various hardware platforms.
Portability: FreeRTOS supports a wide variety of microcontrollers and processor architectures, making it easy to port applications across different hardware platforms.

Extensive documentation and community support: FreeRTOS has a large and active community, which ensures that developers have access to extensive documentation, example code, and support forums.

8.5.2 Disadvantages of FreeRTOS

Limited features: FreeRTOS lacks some features found in more comprehensive operating systems, such as file systems, networking, and advanced memory management.
Learning curve: Developers new to real-time operating systems may experience a learning curve when working with FreeRTOS, as it requires understanding concepts like tasks, priorities, and scheduling.

8.5.3 Alternative Options

ChibiOS/RT: While ChibiOS/RT is another lightweight RTOS with features comparable to FreeRTOS, it has a smaller community and less extensive documentation compared to FreeRTOS. The availability of comprehensive documentation, example code, and community support for FreeRTOS played a significant role in our decision-making process. Furthermore, the performance differences between ChibiOS/RT and FreeRTOS were not significant enough to justify choosing ChibiOS/RT over FreeRTOS for our project.
Zephyr: The Zephyr Project is a more feature-rich RTOS, offering advanced features such as networking, file systems, and security. However, these additional features come at the cost of increased complexity and memory usage. Given the resource constraints of our microcontroller-based system, the lightweight and efficient nature of FreeRTOS was more suitable for our needs. Moreover, our project did not require many of the advanced features provided by Zephyr, making it unnecessary to adopt a more complex RTOS.
RIOT: RIOT is designed specifically for IoT devices and low-power microcontrollers, offering a user-friendly API and support for various communication protocols. However, RIOT's focus on IoT devices meant that some of its features were not applicable to our project. Additionally, like ChibiOS/RT, RIOT has a smaller community and less extensive documentation compared to FreeRTOS. Given the need for robust community support and documentation, FreeRTOS was a more attractive choice for our project.

9.0 Communication System

When developing the overall system, several design decisions are made concerning communication protocols. These decisions were influenced by factors such as power consumption, data transfer speed, reliability, ease of implementation, and compatibility with existing systems. In this section, we will discuss the chosen protocols in-depth, comparing them to alternative options and discussing the advantages and disadvantages of each.

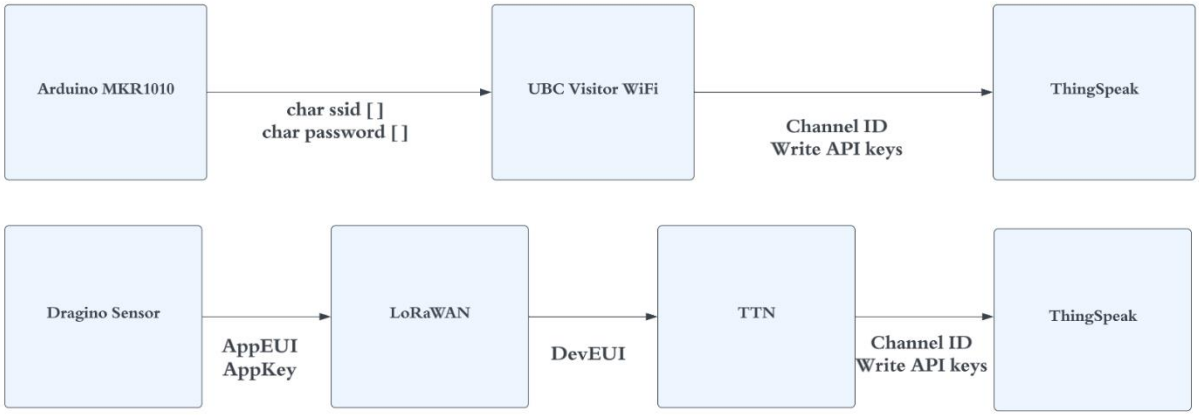


Figure 18: Communication System Initialization Parameters

Figure 18 illustrates the initialization parameters passed from the Arduino MKR WiFi 1010 and Dragino sensors to establish a connection with and to write to ThingSpeak.

9.1 WiFi for Arduino MKR WiFi 1010

WiFi is chosen as the communication protocol for the Arduino MKR WiFi 1010 because of its widespread availability, ease of use, and compatibility with the IoT platform ThingSpeak. WiFi allows for efficient and reliable data transfer with minimal latency. Although it consumes more power compared to some low-power wireless protocols like Zigbee or Bluetooth Low Energy (BLE), it provides a higher data transfer rate. Zigbee and BLE are low-power alternatives, but they lack the data transfer speed and widespread availability of WiFi. WiFi's higher bandwidth capacity and seamless integration with IoT platforms outweigh its power consumption disadvantage in this case.

By default, microcontrollers are unable to access the UBC visitor WiFi network. To resolve this, the MAC address of the microcontroller(s) is registered with UBC IT.

During the uploading task of the microcontroller, a connection protocol is executed, through the onboard WiFi-module, to establish a connection between the microcontroller and the WiFi network at UBC. With the use of two libraries: WiFiNINA and SPI, the WiFi-module passes the network SSID (domain of the desired WiFi network: char ssid[] = "ubcvisitor"), along with the corresponding network password (char pass[] = ""), establishing the microcontroller as a client of the UBC visitor WiFi network. After authentication, the microcontroller obtains an IP address from the network's Dynamic Host Configuration Protocol (DHCP) server, which provides it with network configuration parameters (i.e., subnet mask, default gateway, DNS server).
With a valid IP address, the microcontroller can establish an internet connection and securely communicate with the ThingSpeak database.
To communicate with ThingSpeak, the microcontroller passes channel ID and write API keys. It uses the MQTT communication protocol (Appendix X) to transmit data. ThingSpeak relies on token-based authentication for secure communication between the device and the platform. The microcontroller sends the channel ID to associate the device with the user's account and the write API key to authorize data uploads to the corresponding ThingSpeak channel.

9.2 LoRaWAN for Dragino LHT65 Sensors

LoRaWAN was selected as the communication protocol for the window-surface sensors due to its low power consumption, long-range capabilities, and ability to handle many devices. The LoRaWAN protocol is specifically designed for low-power, wide-area networks (LPWANs) and is well-suited for IoT applications such as this project. Alternative options like NB-IoT and Sigfox are possible, but LoRaWAN was chosen for its wider coverage, flexibility, and ease of deployment. NB-IoT and Sigfox are also LPWAN options, but they have some limitations. NB-IoT requires more infrastructure and is more suitable for dense urban environments, while Sigfox has lower data rates and may not be available in all regions. LoRaWAN offers a more versatile and adaptable solution for this project.
LoRaWAN also uses a star-of-stars network topology, which means that individual devices can communicate with one or more gateways connected to the internet. This allows for many devices to be connected to the network without creating a bottleneck: affecting communication speed. In addition, LoRaWAN uses a spread spectrum modulation technique that allows for a high degree

of immunity to interference and noise. This means that the network can maintain high levels of reliability and availability even in noisy or crowded environments.

The Dragino temperature and humidity sensors are connected to LoRaWAN through their built-in LoRa module, which enables it to communicate with LoRaWAN gateways. To establish a connection between the sensors and the LoRaWAN network, the sensors use an AppEUI and AppKey. The AppEUI is a unique identifier for the application, while the AppKey is a secret key that is shared between the sensor and the network to ensure secure communication. As the sensors periodically take temperature measurements, packets of data are sent using the LoRa modulation to the nearest gateway (there are two located on the UBC campus). The gateway then forwards these packets to a LoRaWAN network server, which authenticates the devices and decrypts the packets. From there, the LoRaWAN network server sends the data to The Things Network (TTN) along with a DevEUI.

9.3 MQTT for ThingSpeak Integration

MQTT was chosen for its lightweight design, low power consumption, and efficient use of bandwidth. It is a widely adopted protocol for IoT applications and is natively supported by ThingSpeak. This protocol enables seamless communication between the Arduino, Dragino sensors, and the ThingSpeak platform. Alternative protocols like CoAP and AMQP are possible, but MQTT's simplicity, minimal overhead, and broad support made it the preferred choice. CoAP, while also lightweight, is more suited for constrained environments and uses a request-response model, which is not ideal for this application. AMQP, although reliable and suitable for complex messaging, has a larger overhead compared to MQTT and is less optimal for low-power IoT devices.

The DevEUI is a 64-bit globally unique Extended Unique Identifier (EUI-64) assigned by the manufacturer of the end-device (Dragino) to distinguish it from other devices in a network and is a key component of the device authentication and secure communication process. Once the DevEUI is passed to TTN and the connection between the sensors, through LoRaWAN, and TTN is established, data can then be passed from TTN. To transmit the window-surface temperature data to ThingSpeak, TTN uses the MQTT (Appendix X) protocol, but first must be configured with the appropriate Channel ID and Write API Keys. The Channel ID is a unique

identifier for the ThingSpeak channel, while the Write API Keys provides access to write data to the channel.

9.4 HTTP for External Temperature Data

The external temperature data from the weather report website is retrieved using the HTTP protocol. HTTP was chosen for its simplicity, widespread use, and compatibility with web-based data sources. It is a stateless protocol that allows easy fetching of data from web servers, making it suitable for integrating the ambient air temperature data into the project. Alternatives such as FTP or WebSocket are possible, but HTTP was deemed more suitable due to its widespread adoption and ease of use. FTP, while designed for file transfer, is more complex and less suited for fetching simple data from web servers. WebSocket, on the other hand, is more appropriate for real-time, bi-directional communication, which is not required in this case.

10.0 Data Storage and User Interface

10.1 Significance to requirements

The data storage system acts as the central repository for all the information gathered from various sources, necessitating adequate capacity to accommodate the collected data. Additionally, it is crucial to design data storage systems with accessibility as a priority. Researchers need convenient access to the required data to perform their studies efficiently. Consequently, a user-friendly interface that is simple to navigate and comprehend enables researchers to access data promptly, thereby promoting effective data analysis.

10.2 Context with respect to system, design, architecture

The user interface establishes communication with multiple devices, such as the microcontroller, The Things Network, and the temperature webpage of the UBC ESB Rooftop Weather Station. The microcontroller is programmed to check for new impacts every 5 minutes. Upon detecting a new impact, it sends a data packet containing the cumulative collision count and timestamp. The interface processes the received cumulative collision count using its built-in analytics tools before updating the database. This ensures that only the latest collision data is stored and analyzed. Additionally, if the accelerometer falls, the microcontroller sends a flag, as outlined in

the fall detection algorithm. ThingSpeak's built-in analytics tools include pre-set codes that run every 5 minutes to identify falls. If fall detection data is received, the interface automatically sends an email notification to the user.

The Things Network transmits two distinct data packets at 30-minute intervals, each containing surface temperature readings from separate sensors. These readings are used to compute the heat flow rate across the window. The updated results are then stored in the database and displayed on the user interface at the same frequency.

Furthermore, the built-in analytical tool's pre-set code gathers external air temperature information from the UBC ESB rooftop weather station website every 30 minutes. This ensures a comprehensive and up-to-date data analysis and visualization for users.

10.3 Design

10.3.1 Design Decision

ThingSpeak, an IoT platform and analytics tool, is chosen as the data storage and user interface solution for our system. It allows users to collect, analyze, and visualize data from multiple connected devices. ThingSpeak's capacity is measured in units: one unit of the free plan provides the ability to process and store up to 3 million messages per year (~8,200 messages/day) and create up to 4 channels (customizable container for storing and visualizing data collected from IoT devices or other data sources) [32].

Each channel can have up to 8 data fields that represent different types of data collected from devices or websites. A data field is a container that holds a single data value, such as a temperature reading, humidity level, or any other sensor measurement. For our project, the 8200 messages per day available in ThingSpeak are more than sufficient to handle the maximum of 196 data entries (calculated in section 10.3.3) collected daily. Additionally, ThingSpeak has a deprecated version of MATLAB built in for analysis, allowing us to read surface temperature data from two separate data fields, calculate heat flow rates in real-time, and store the results in other data fields. We can also send an email notification to users using pre-set codes in the analytics tools when an accelerometer fall message is detected. Moreover, the TimeControl app in ThingSpeak provides a flexible way to automate tasks such as heat flow rate analysis and web temperature data scraping by setting recurring intervals of 30 minutes.

For the user interface, ThingSpeak enables us to plot data fields on charts, visualize them in gauges, display them in tables, and export them to other tools for further analysis. Researchers can export all stored data from the channel, or specific data from a single data field, in one of three file formats. The dashboard created in ThingSpeak is presented in Appendix IX.

10.3.2 Data Storage Design Consideration

Research conducted by the UBC Science faculty indicates that the highest number of collision deaths occurred during the fall sampling period, with a rate of 1.38 collisions per day for 8 buildings, compared to other seasons [10]. Based on this data, the number of impacts per building per day is approximately 0.17. For estimating the data storage requirements, one impact per building per day is used. Additionally, we tested the equipment and determined that it would only fall once per month. As a result, we assume that the device will be dropped at most once a month. To ensure the selected database has sufficient storage size for 1 bird strike, and constituent data, per day, the choice for data storage must meet the minimum specifications in Table 12.

Table 12 illustrates the components considered in the selection of data storage to meet ***F2***, ***F3***, ***F4***, ***F5***, ***F6***:

Table 12: Data Storage Criteria.

Store one data entry for each bird impact, which includes the total number of collisions for a specific window in a building along with the date and time of the bird collision.
Store two separate data segments for each impact received from the microcontroller, detailing: • The cumulative number of bird collisions for a specific window in a building. • The corresponding timestamp of the bird impact.
Store 1 individual data segment for the fall detection sent from the microcontroller.
Store 4 individual temperature data segments updated every 30 minutes: • Outside air temperatures • Inside surface temperatures • Outside surface temperatures • Heat flow rate for window’s insulation analysis
Store the latitude and longitude of the building that window placed when the location needs to be changed each time.
Maximum data segment size of 32 characters.

For the storage of collision data, each set of detection data from the microcontroller is first stored in a specific location and then read out during data analysis, after which it is decided whether the received collision data needs to be stored in a collision-specific database and displayed. As a result, 3 distinct data segments need to save for each impact. Additionally, since the microcontroller checks every 5 minutes to see if the device has been dropped, it will send the fall detection data only if a fall is detected.

The maximum data segment size of 32 characters is made by observing the sample data (Appendix VI) downloaded from our existing channel created on the ThingSpeak platform. The sample data in Appendix VI shows that the timestamp has the longest data segment size of 20 characters; therefore, 32 characters is sufficient for every individual data segment that needs to be stored for each bird strike.

Given that the location of all strikes detected by a specific device is the same, the building's latitude and longitude only need to be stored once when the device is set up.

10.3.3 Minimum Specification Description for Data Storage

Minimum specifications (Table 13) are based on the size of the sample data file (Appendix VI), which contains 102 data entries received and stored within a 12-hour period, with a maximum size of 3790 bytes. This means that a maximum of 7.1 kilobytes (7283 bytes) of data storage is required each day to store the 196 data entries that need to be collected per day, including:

- 48 outside surface temperature data segments
- 48 inside surface temperature data segments
- 48 outside air temperature data segments
- 48 heat flow rate data segments
- 2 collision data segments sent from the microcontroller as the impact detected
- 1 analyzed bird impact counts data segments being stored in the database
- 1 fall detection data sent from the microcontroller if the accelerometer falls down

Thus, our database must have at least 213 kilobytes data storage size to store 5910 data entries per 30 days period. To ensure sufficient storage remaining at the end of 30-day maintenance period, the minimum storage size within one maintenance period needs to be at least 300

kilobytes to meet ***F2***, ***F3***, ***F4***, ***NF4*** and ***NF7***. Equations used for the values contained in this paragraph are attached in the Appendix VII.

Table 13: Data storage minimum specifications.

Feature	Minimum Specifications
Storage Size	300KB
Data Segment Length	32
Maximum Entries of Data/day	196

10.3.4 User Interface Design Considerations

The following components are considered to satisfy requirements of ***F6*** and ***F7*** and characterized as minimum specifications for the UI design:

- Display all required data through various parts
- Represent historical data graphically
- Export all stored data from the database to Excel

10.3.5 Other Options Evaluated

Other candidates that were evaluated for data storage and user interface are:

- Cayenne [33]
- MongoDB [34]
- MySQL [35]

Appendix VIII compares ThingSpeak and Cayenne to the two options that were not chosen. Cayenne meets all minimum requirements for data storage and user interface and was implemented in early iterations of the system. Due to issues encountered with the complexity of the system design, it was removed from the final version of the system. More details on why Cayenne was not implemented are presented in the Appendix VIII.

MongoDB was excluded due to its limited free storage size, while other three options offer sufficient storage size, with no fee. Although MySQL meets or exceeds the minimum requirements for data storage, MySQL was not selected because it does not provide dashboards that can be customized to satisfy the requirements for user interface, while both custom

dashboards of ThingSpeak and Cayenne offer a flexible way for users to manage and display data through multiple widgets.

10.4 Evidence it works

ThingSpeak is the best option for data storage and user interface in the system design. The calculation accuracy and stability of the heat flow rate by codes in MATLAB Analysis are tested in section 3.0 in Verification & Validation. The test results indicate that the heat flow rate calculated by the MATLAB Analysis codes is accurate, and the update interval of every thirty minutes meets our requirements. In addition, section 2.0 in Verification & Validation indicates that the timestamp received from microcontroller is correctly converted and stored into the ThingSpeak with its corresponding impact counts. The email that is triggered to notify the user that the accelerometer falling is also tested in section 3.0 in Verification & Validation, more detailed testing results please see the Verification & Validation section.

11.0 Hardware

11.1 Hardware Enclosure

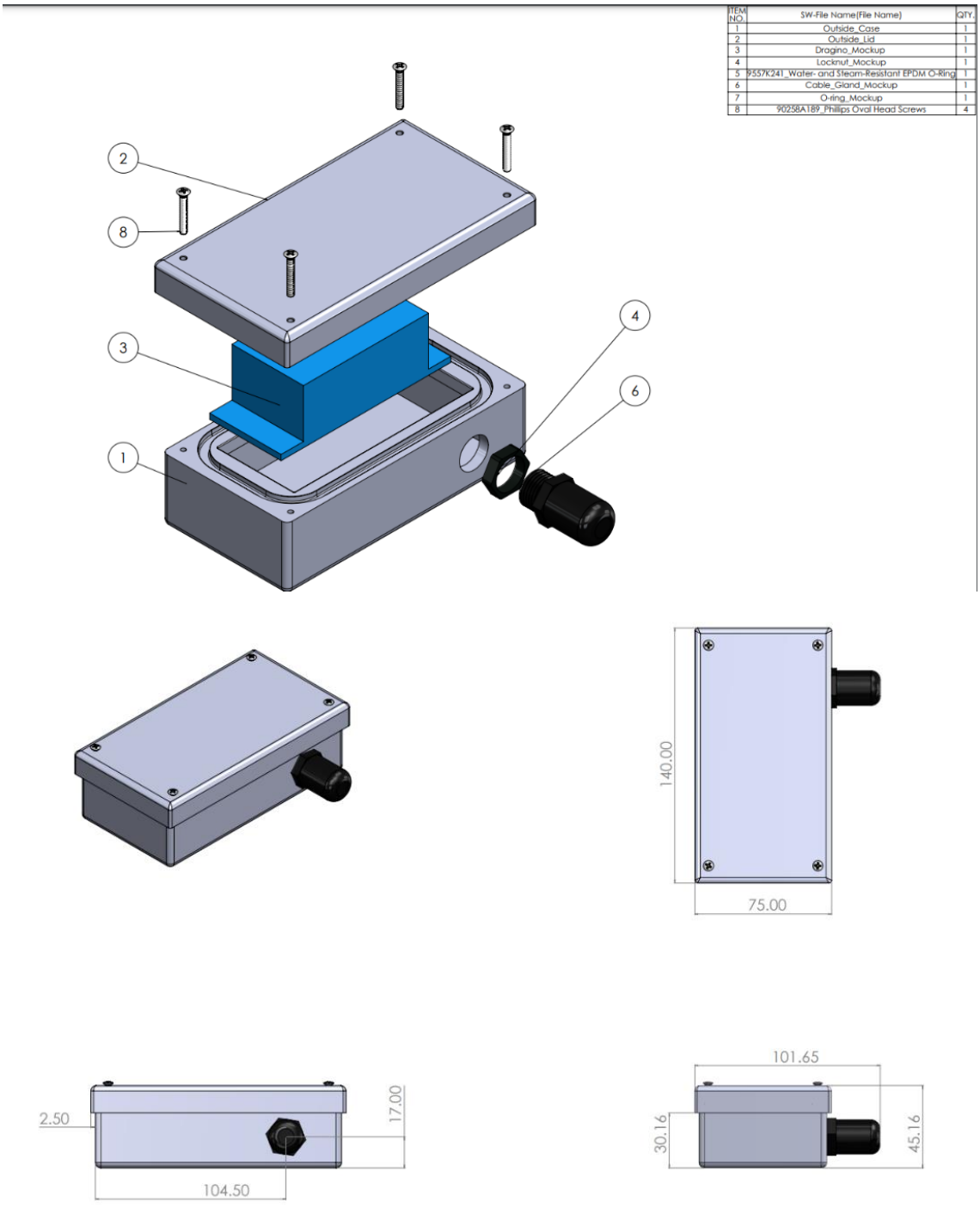


Figure 19: Current hardware design for the outside temperature sensor.

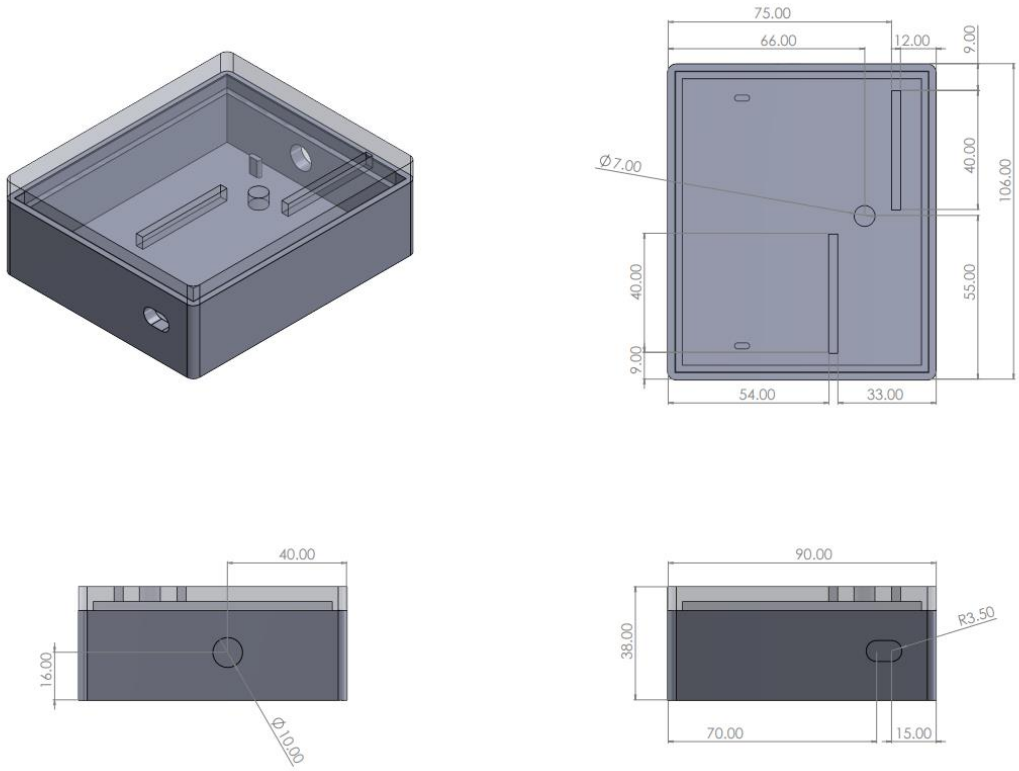


Figure 20: Current hardware design for the inside temperature sensor.

Since our device is susceptible to encountering conditions such as rain, someone dropping water on it, wind and being hit by objects such as tree leaves, the device's casing must be made secure and able to withstand these conditions. To satisfy ***F5*** requirement, the case is attached on the windowsill using Velcro and is aligned in such a way that doesn't compromise the holes in the case to severe weather conditions according to ***NF5***. The cases are printed using FDM (Fused Deposition Modeling) technology using PLA plastic material.

11.1.1 Hardware Enclosure for inside

For our inside device, the Arduino and temperature sensors are placed in a 3D printed case with adequate openings for wires coming in and out of the case. The case must be placed on a windowsill and will be secured using Velcro that is going to be attached to the bottom of the case. To satisfy our requirements (***F5*** and ***NF5***), the case on the inside will have to satisfy an IP rating of 2x or above. This will mean that the case must be protected against solid objects over 12mm (such as a finger).

11.1.2 Hardware Enclosure for outside

On the outside, the case is more prone to rain and random objects, such as leaves, hail or birds, hitting it. This means that the outside case must have an outside rating of IP23 or higher which entails that it should be secure enough to be protected against solid objects over 12mm (such as a finger) and water spraying on it for 3 min for up to a 60-degree angle [36]. Again, the case is printed using the same method and will contain the Dragino sensor with openings for wires (the inputs and outputs of the system). The case for the outside will have 4 screw holes on the lid to secure the lid. In addition, the case will have a seam where a rubber cable gland will be placed to prevent water from going into the case. This rubber cable gland will be secured tightly when the lid is tightened with the screws. The case also has a rubber gasket to allow wires to come out of the case without compromising the case to water going in through the wholes. This is because the gasket is tightened on the wires.
Velcro strip is attached to the bottom of the case. The case is attached to the window still to make sure it does not fall off the sill. This also helps make sure that the case will withstand wind and rain.
The chosen colors for the outside and inside modules will be white as certain studies show that white and other bright colors are signs of danger for birds and other flying mammals. This means that birds will try and avoid it meaning that the modules set on the windowsill inside and outside are less prone to be directly hit by birds.

11.2 Battery Powered Option

11.2.1 Design Decisions

The bird collision detection system was designed to be self-sufficient, environmentally friendly, and versatile by utilizing a lithium-ion battery with a capacity of 3.7V 3200mAh. Several factors influenced this decision, with the primary focus being on compatibility, longevity, efficiency, and flexibility in system placement.

11.2.2 Advantages

- Lithium-ion batteries possess a high energy density, ensuring that our compact system has a long-lasting and reliable energy storage solution.

- They exhibit a relatively low self-discharge rate compared to other rechargeable batteries, thus prolonging the system's operational time between recharges.
- Lithium-ion batteries experience minimal memory effects, preventing the loss of capacity if they are not fully discharged before recharging.
- Being battery-powered allows the system to be placed on numerous windows, even those not near a power outlet, providing increased flexibility and broader monitoring coverage.

11.2.3 Disadvantages

- Lithium-ion batteries can be temperature-sensitive, which could impact their performance and lifespan in extreme environments.
- A dedicated charging circuit is often necessary to prevent overcharging and potential damage.

11.2.4 Alternative Approaches

Although other battery types such as nickel-metal hydride (NiMH) or lead-acid could have been considered, lithium-ion batteries were ultimately chosen due to their superior energy density, favorable discharge characteristics, and the added advantage of flexible system placement.

11.2.5 Battery-Saving Design Decisions

- **Scheduled Data Upload:** The Arduino uploads data to ThingSpeak only once every 24 hours, reducing the frequency of wireless communication and conserving battery life.
- **Optimized Fall Detection:** Fall detection tasks are called every 5 minutes, as opposed to continuous monitoring of the accelerometer, which reduces power consumption.
- **Efficient Task Management:** The system leverages FreeRTOS tasks to manage the concurrent execution of tasks, ensuring minimal power consumption by running tasks only when necessary.

11.2.6 Battery Life Calculation

Based on the battery life calculations (see Appendix XI), the system can run using a rechargeable Lithium-ion battery (3.7V / 3200mah) for 4 days. This extended battery life has greatly increased

the system's versatility, allowing it to monitor a wide variety of windows without being dependent on a wall outlet for power.

Verification & Validation

1.0 Temperature Communication System Verification

1.1 Purpose

The purpose of testing a temperature communication system is to evaluate its performance and reliability in accurately measuring, transmitting, and receiving temperature data. This test ensures that users are provided with accurate data on the thermal insulation performance of windows, thereby facilitating the achievement of low-energy objectives in new building construction. This test also ensures that ThingSpeak does in fact output the temperature measurements and is able to calculate the heat flow rate accurately.

1.2 Equipment

Table 14: Temperature Communication System testing equipment.

Equipment	
1	Dragino LHT65
2	Test window
3	Computer [Development tool]

1.3 Set up

1. Register the Dragino LHT65 in The Things Network V3 by using AppEUI, DevEUI and the AppKey
2. Connect the external temperature sensor DS18B20 to the Dragino LHT65
3. Use ACT button to activate LHT65 and it will automatically join the TTN network
4. On TTN, replace downlink queue with the command “0xA201” to set external sensor type to E1 Temperature Sensor to read the surface temperature data from DS18B20
5. On TTN, replace downlink queue with the command “0x01000708” = 1800 seconds to change LoRaWAN End Node Transmit Interval to 30 minutes
6. Observe the uplink data message sent from LHT65 in the Live data section
7. Select the ThingSpeak in the Webhooks of the integrations section in the TTN

8. Input the ChannelId and Write API Key given in the API keys section of ThingSpeak
9. Go to the Payload formatters section of the End devices webpage in the TTN
10. Change the Formatter type to Custom JavaScript formatter
11. Copy the codes given in the type of Use Device Repository formatter
12. Add the commands to assign measured surface temperature data to the specified data field in ThingSpeak
13. Open the data field used to collected data from TTN networks
14. Observe the measured temperature data in the Dashboard of ThingSpeak

1.4 Testing Tasks

The testing process involves evaluating the efficiency of the communication protocols and interfaces used for data transmission, and analyzing the system's performance under varying conditions, such as temperature ranges, environmental factors, and communication loads.

Table 15: Dragino-TTN and TTN-ThingSpeak Connection Results.

Task		Result
1	Dragino LHT65 connection to TTN	Pass
2	TTN sending data to ThingSpeak	Pass

Criteria:

- Pass: connection success
- Fail: connection fail

1.5 Results

1.5.1 Task 1

Figure 21 illustrates the smooth communication between the Dragino LHT65 and TTN through the LoRaWAN Network. The “TempC_DS” indicates the temperature in degree of the external temperature sensor (Window surface temperature) and the “field 1” indicates the specified data field connected in ThingSpeak. In the figure, it can be seen that both Dragino LHT65 sensors send temperature data (Uplink Message) once every 30 minutes verifying that the connection is in fact successful.

Figure 21: Dragino LHT65 sensor connection to TTN.

1.5.2 Task 2

Figure 22 depicts the connection status of the Dragino sensors with ThingSpeak. Both connections use the LoRaWAN network to then pass through to TTN (The Things Network). The connection is verified as the "Healthy" status indicates that the connection is established and secured.

ID	Base URL	Template ID	Status	Created at
data-to-thingspeak	https://api.thingspeak.com/things_network/v3/update	thingspeak	Healthy	Mar 7, 2023

Figure 22: TTN webhook connection.

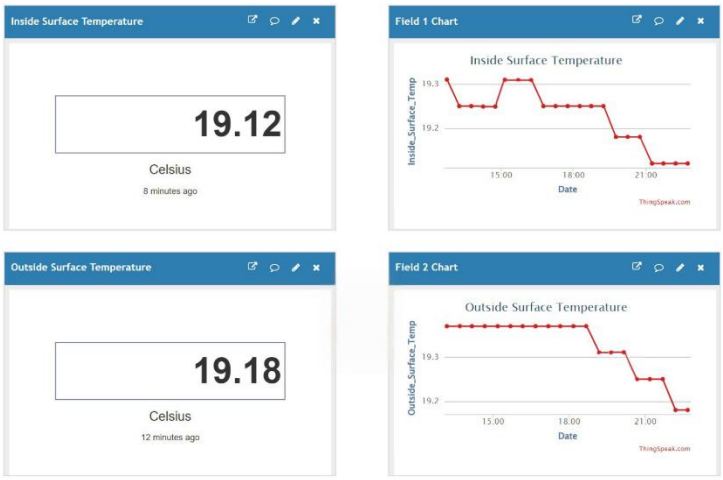


Figure 23: Two surface temperature data's display on the dashboard of ThingSpeak.

The line charts the right side of Figure 23 illustrate the field 1 and field 2 in this ThingSpeak channel is connected to two separate Dragino sensors. ThingSpeak receives two surface temperature data packets every 30 minutes from TTN through the LoRaWAN Network. After that, it displays their updated values and represent these received data in line chart on the Dashboard of ThingSpeak.

The line charts on the right side of Figure 23 show historical stored data in the data fields 1 and 2 of the ThingSpeak channel. These two data fields receive surface temperature data every 30 minutes from The Things Network (TTN) via the LoRaWAN network. The updated values are

displayed and represented in a line chart on the ThingSpeak dashboard, which also satisfy the requirement ***F8*** for the user interface. This verifies that the data is read and stored on ThingSpeak which can later be accessed by researchers for analysis.

2.0 Microcontroller FreeRTOS and data Communication System Verification

2.1 Purpose

The goal of the testing is to assess FreeRTOS's ability to provide deterministic and reliable task scheduling, meet real-time deadlines, manage system resources efficiently, and ensure overall system stability and robustness.

2.2 Equipment

Table 16: Microcontroller testing equipment.

Equipment	
1	Arduino MKR WiFi 1010
2	Test window
3	Development tool [IDE]
4	ThingSpeak Page
5	Accelerometer
6	Jumper wires
7	Duct seal putty

2.3 Set up

1. Connect the accelerometer to the Arduino MKR WiFi 1010
2. Fixing the accelerometer to the test window using duct seal putty
3. Connect the Arduino MKR WiFi 1010 to the computer
4. Verify the connections and ensure that the accelerometer is properly powered
5. Create a new test file for FreeRTOS in the Arduino IDE
6. Download any needed library

7. Complete the corresponding code for testing
8. Open the ThingSpeak webpage to ensure real-time data visualization

2.4 Testing Tasks

The testing process involves running FreeRTOS on a microcontroller, executing different tasks with varying priorities and timing requirements, and measuring the system's performance in terms of task scheduling, and context switching.

Table 17: Microcontroller testing tasks and result.

Test		Result
1	Plan the execution sequence of the corresponding threads and achieve the expected results in FreeRTOS by setting different priority orders and delay time	Pass
2	Testing microcontroller's time data transmission accuracy to ThingSpeak page during simulated bird impact event on glass surface	Pass
3	Removing the accelerometer from the window to simulate a fall and observing if the microcontroller can detect the occurrence and send correspond information to ThingSpeak	Pass

2.5 Results

2.5.1 Test 1

This is the conclusive outcome of Test 1, illustrating that FreeRTOS effectively and reliably scheduled the various tasks as per our predetermined objectives. This attests to the stability and dependability of FreeRTOS in the context of our project.

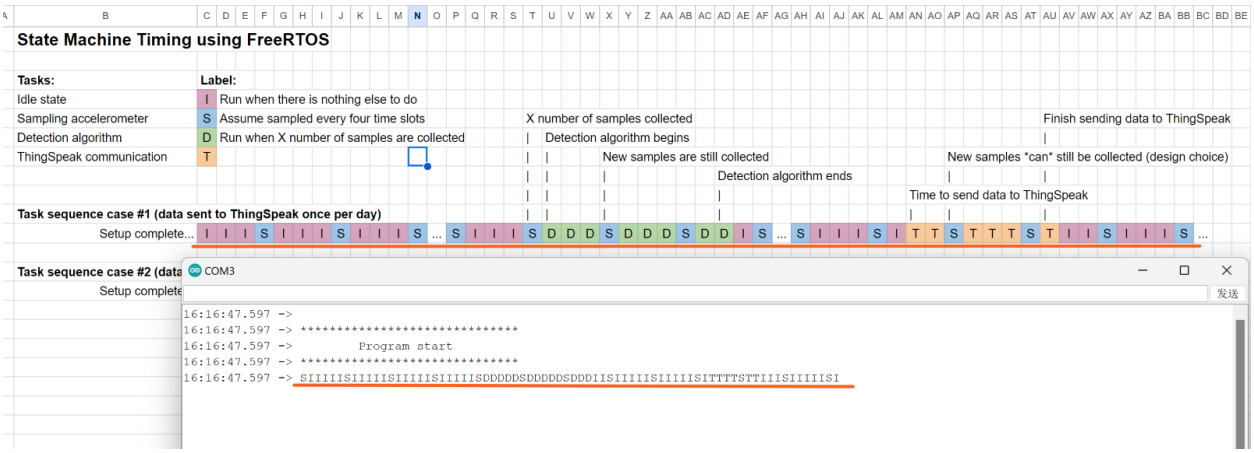


Figure 24: FreeRTOS Timing testing sequence case 1 result.

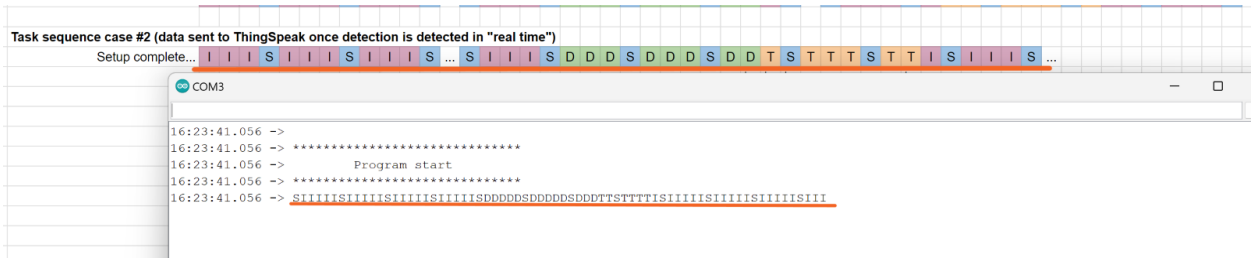


Figure 25: FreeRTOS Timing testing sequence case 2 result.

2.5.2 Test 2

The outcome of Test 2 demonstrates that under the coordination and scheduling of the microcontroller and FreeRTOS, the bird impact events were accurately detected, recorded, and transmitted to the backend of ThingSpeak. Figure 24 shows the initialization stage of the system, which includes the initialization of Task 3 and 4, as well as the WiFi connection. Figure 25 displays the information of impact detection and timestamp logging.

```
18:28:43.503 -> Please upgrade the firmware
18:28:43.503 -> Attempting to connect to WPA SSID: ubcvisitor
18:28:50.683 -> SSID: ubcvisitor
18:28:50.683 -> BSSID: CC:DB:93:44:A7:C5
18:28:50.683 -> signal strength (RSSI):-78
18:28:50.683 -> Encryption Type:7
18:28:50.683 ->
18:28:50.683 -> IP Address: 137.82.226.173
18:28:50.683 -> 137.82.226.173
18:28:50.683 -> MAC address: 24:62:AB:B9:33:94
18:28:50.732 -> 28
18:28:50.732 -> Task 3: Fall Detection Test
18:28:50.732 -> Accelerometer Y value:1.06
18:28:50.732 -> Task 3: Fall Detection Test END
18:28:50.732 -> Task 4: Sending data to ThingSpeak start
18:28:50.732 -> 0
18:28:53.238 -> Task 4: Sending data to ThingSpeak End
18:28:53.238 -> Task 2: Detection algorithm begin
18:28:53.238 -> ......Task 1: 256 samples collected
18:28:53.865 -> Task 2: Detection algorithm begin
18:28:53.993 -> .....Task 1: 256 samples collected
18:28:54.515 -> Task 2: Detection algorithm begin
```

Figure 26: Initialization of the system.

```
18:35:51.693 -> .....Task 1: 256 samples collected
18:35:52.294 -> Task 2: Detection algorithm begin
18:35:52.334 -> detected
18:35:52.383 -> 183551
18:35:52.383 -> .....Task 1: 256 samples collected
18:35:52.955 -> Task 2: Detection algorithm begin
18:35:53.003 -> detected
18:35:53.003 -> 183552
18:35:53.003 -> .....Task 1: 256 samples collected
18:35:53.625 -> Task 2: Detection algorithm begin
18:35:53.673 -> .....Task 1: 256 samples collected
18:35:54.243 -> Task 2: Detection algorithm begin
```

Figure 27: Detecting simulated bird collisions.

The data was sent to ThingSpeak after a 3-minute interval for faster result monitoring. Figure 28 illustrates the sending processes with three loops for collision time and total collision number. Figure 29 displays how the data is shown on the user interface, accurately capturing impact count and timing. The timing of the second collision shown on the user interface matches the detection time printed on Figure 27. This validates the successful implementation of our system for collecting and presenting bird impact data using FreeRTOS.

```
18:37:49.222 -> .....Task 1: 256 samples collected
18:37:49.827 -> Task 2: Detection algorithm begin
18:37:49.875 -> .....Task 4: Sending data to ThingSpeak start
18:37:50.390 -> 2
18:37:52.860 -> start1
18:37:52.860 -> Loop0
18:37:53.238 -> Updating channel. HTTP code 200
18:38:13.248 -> Loop1
18:38:13.627 -> Updating channel. HTTP code 200
18:38:33.634 -> Loop2
18:38:34.120 -> Updating channel. HTTP code 200
18:38:34.120 -> Task 4: Sending data to ThingSpeak End
18:38:34.120 -> Task 2: Detection algorithm begin
18:38:34.120 -> .Task 1: 256 samples collected
18:38:34.168 -> Task 2: Detection algorithm begin
```

Figure 28: Sending data to ThingSpeak process.

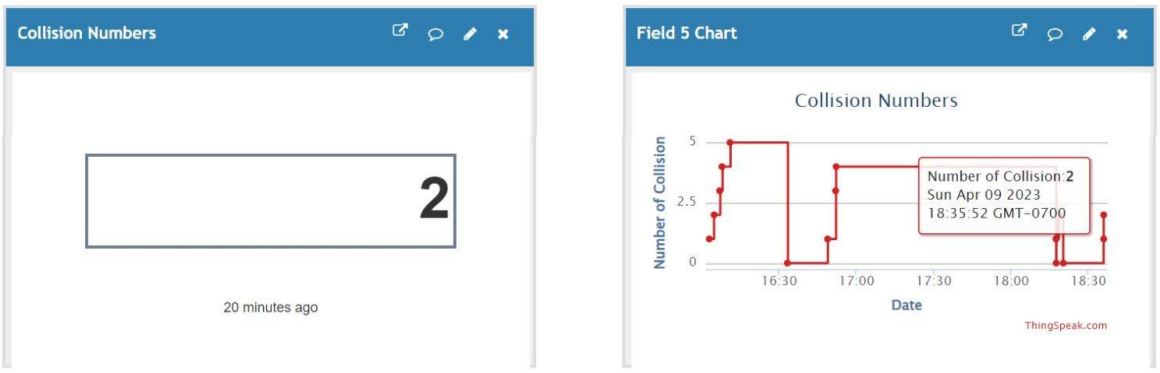


Figure 29: ThingSpeak demonstration.

2.5.3 Test 3

The results of Test 3 indicate that our microcontroller successfully detected the fall of the accelerometer and sent the information to ThingSpeak, as evident from the graph below. In Figure 30, during the execution of Task 3, the microcontroller detected a Y-value of -0.15 for the accelerometer, which is significantly lower than our threshold, indicating a fall event. In Figure 31 and Figure 32, we observed that Task 3 ran again after 5 minutes, detecting another fall event and immediately suspending other tasks while sending information to ThingSpeak. When the reset button was pressed, Task 2 and Task 3 were reactivated, followed by Task 1 after a delay of 5 seconds, as seen in Figure 32.

```
19:14:01.870 -> Task 3: Fall Detection Test
19:14:01.870 -> Accelerometer Y value:-0.15
19:14:01.870 -> Fall1
19:14:01.870 -> Task 3: Fall Detection Test END
19:14:01.870 -> Task 4: Sending data to ThingSpeak start
19:14:01.870 -> 0
19:14:04.413 -> Task 4: Sending data to ThingSpeak End
19:14:04.413 -> Task 2: Detection algorithm begin
19:14:04.413 -> ......Task 1: 256 samples collected
19:14:05.050 -> Task 2: Detection algorithm begin
19:14:05.134 -> .....Task 1: 256 samples collected
19:14:05.648 -> Task 2: Detection algorithm begin
```

Figure 30: First Fall detection.

```
19:19:00.484 -> .....Task 1: 256 samples collected
19:19:01.085 -> Task 2: Detection algorithm begin
19:19:01.134 -> .....Task 3: Fall Detection Test
19:19:01.700 ->
19:19:01.700 -> Still not Press Reset Button Yet
19:19:01.700 -> FALL EMAIL
19:19:02.129 -> Updating channel. HTTP code 200
19:19:02.129 -> 0Still not Press Reset Button Yet
19:19:02.129 -> 0Still not Press Reset Button Yet
19:19:02.129 -> 0Still not Press Reset Button Yet
19:19:02.129 -> 0Still not Press Reset Button Yet
```

Figure 31: Second Fall detection.

```
19:19:07.199 -> 0Still not Press Reset Button Yet
19:19:07.199 -> 0Still not Press Reset Button Yet
19:19:07.199 -> 1Reset Press
19:19:07.199 -> Resume Task B and C First
19:19:07.199 -> Task 2: Detection algorithm begin
19:19:07.199 -> Task 4: Sending data to ThingSpeak start
19:19:07.199 -> 0
19:19:09.686 -> Task 4: Sending data to ThingSpeak End
19:19:09.734 -> Task 2: Detection algorithm begin
19:19:09.734 -> .Task 1: 256 samples collected
19:19:09.734 -> Task 2: Detection algorithm begin
19:19:09.815 -> .....Task 1: 256 samples collected
19:19:10.355 -> Task 2: Detection algorithm begin
```

Figure 32: After Reset button is pressed.

3.0 Verification of Codes in the ThingSpeak's Built-in Analytics Tools

3.1 Purpose

The purpose of testing several codes programmed in the MATLAB Analysis which is the built-in analytics tools of ThingSpeak. This section evaluates each code's performance when ThingSpeak interacts with multiple devices, including the microcontroller, The Things Network, and the

temperature webpage of the UBC ESB Rooftop Weather Station. The goal of the testing is to ensure the accuracy and stability of codes developed in the MATLAB Analysis.

3.2 Equipment

Table 18: MATLAB Analysis Testing Equipment.

Equipment	
1	MATLAB Analysis in ThingSpeak
2	Dashboard Page of ThingSpeak
3	Webpage of UBC ESB Rooftop Weather Station
4	Mailbox

3.3 Tasks

The testing processing including running the codes of each task below and observing their corresponding testing results to determine whether it is passed or not.

Table 19: Contents and Results of Tasks Testing.

Task		**Result**
1	Test the calculation accuracy and stability of the heat flow rate by codes in MATLAB Analysis	Pass
2	Test the conversion accuracy of the bird impact timestamp by codes in MATLAB Analysis	Pass
3	Test the stability of the email notification of equipment fall detection by codes in MATLAB Analysis	Pass
4	Test the stability and accuracy of codes that used to collect the outside air temperature collection from the UBC ESB Rooftop Weather Station website	Pass

3.4 Results

3.4.1 Task 1

From the Figure 23 in section 1.5 displays two surface temperature collected are 19.12°C and 19.18°C respectively. Using the equation provided in section 5.1 in Design, the heat flow rate

can be calculated, which is 0.028Watts. The calculation result is same with the heat flow rate shown in the Figure 33 below. In addition, we can verify that the heat flow rate calculation is updated every 30 minutes and displayed on the dashboard via the line graph in Figure 33.

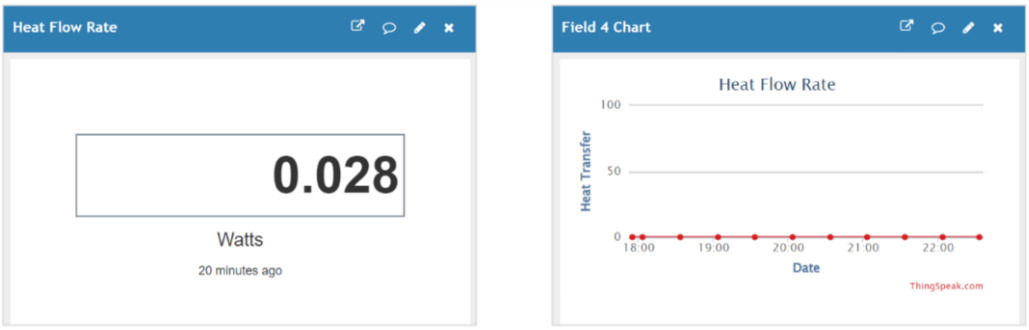


Figure 33: Dashboard Display of Heat Flow Rate.

3.4.2 Task 2

By observing Figure 27 in the section 2.5, we can see the second impact is detected at the timestamp of 18:35:52 2023/4/9. From Figure 29 it is clearly shown that the second impact is stored with its correct timestamp Apr 09, 2023, 18:35:52, which means that the timestamp received from microcontroller is correctly converted and stored into the ThingSpeak with its corresponding impact counts.

3.4.3 Task 3

Figure 34 indicates that the accelerometer falling is detected at 18:56:51. After the fall information is detected by the 5-minute recurring codes in MATLAB Analysis, an email notification is sent to the user. Figure 35 shows the email contents and the time when the email was sent out. The email just sends after the fall message is detected.

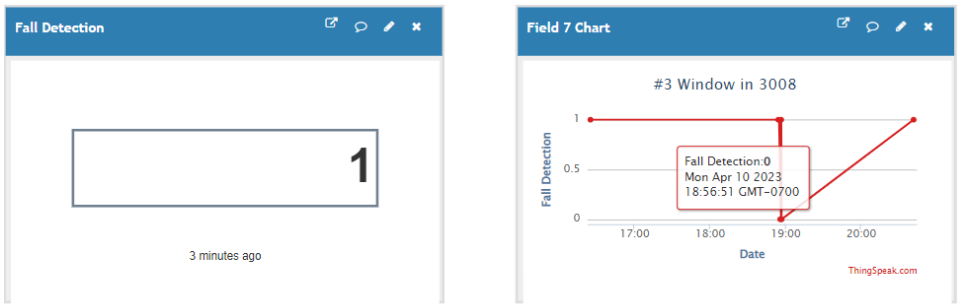


Figure 34: Dashboard Display of Fall Detection.

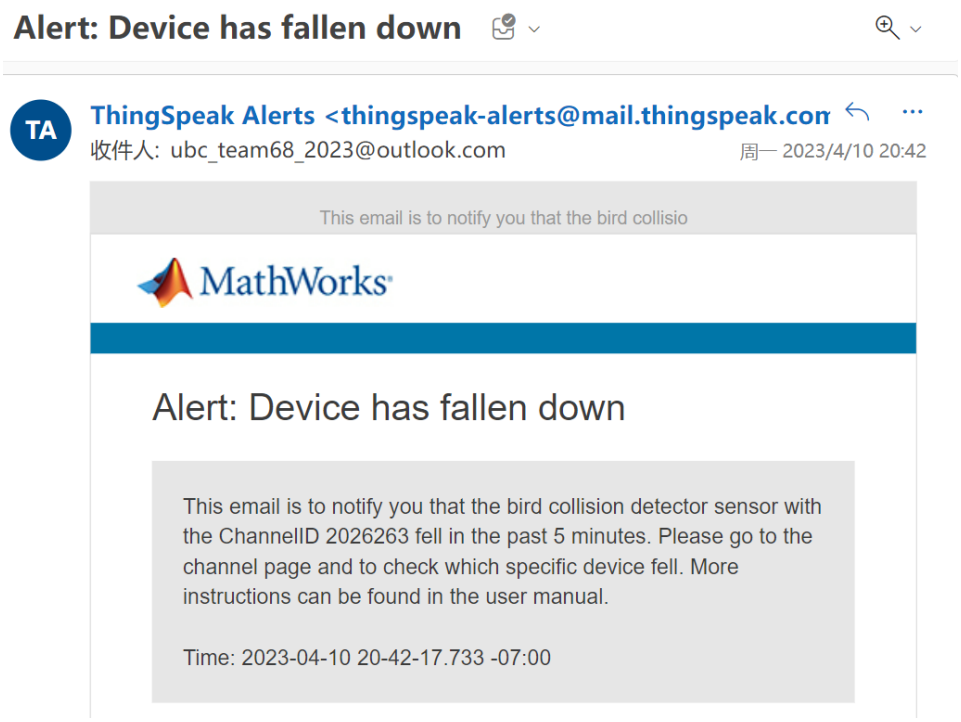


Figure 35: Notification Email Received.

3.4.4 Task 4

From the line chart on the Figure 36 below, it can be seen that ThingSpeak successfully retrieves outside air temperature data from the website of UBC ESB Rooftop Weather Station every 30 minutes.

Figure 36: Dashboard Display of Outside Air Temperature.

Moreover, the temperature data accuracy is verified by comparing the temperature data in Figure 37 with the outside air temperature stored and displayed on the dashboard of ThingSpeak in Figure 38. Both temperatures around 18:40 are 8.8°C, which indicates that the collection of outside air temperature has very high accuracy.

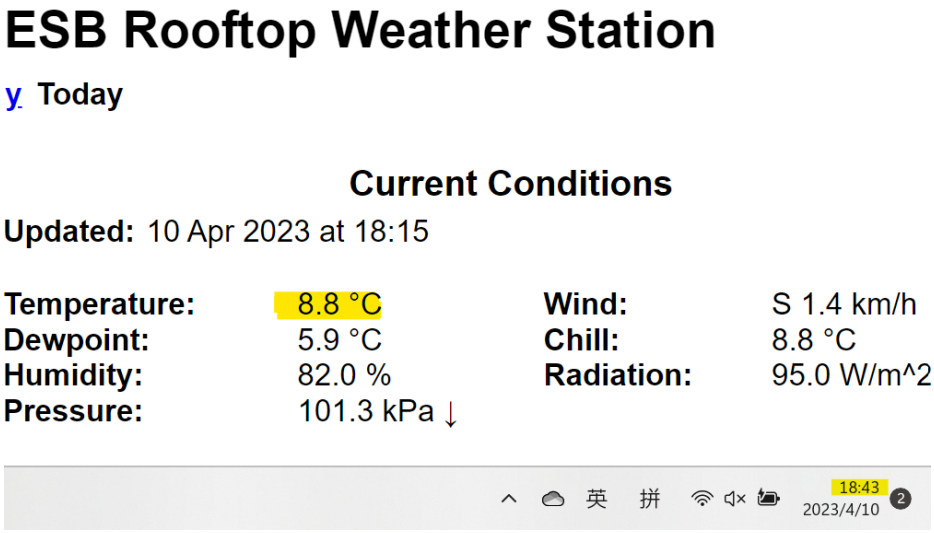


Figure 37: Temperature Data from the UBC ESB Rooftop Weather Station.

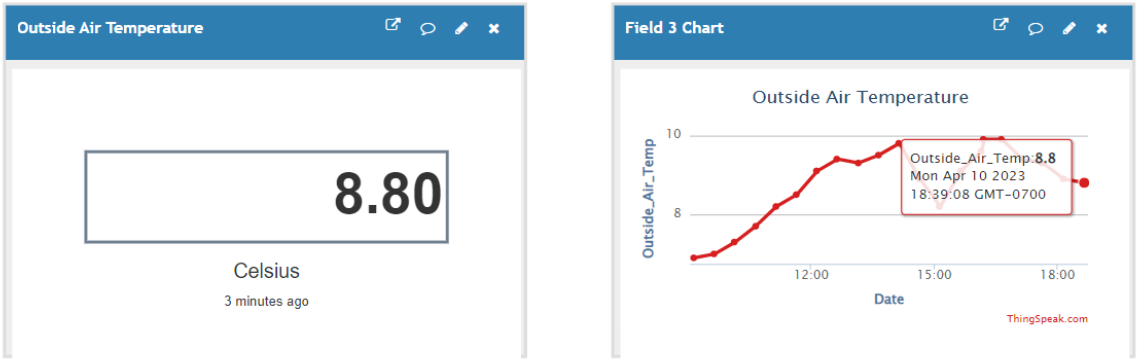


Figure 38: Dashboard Display of Outside Air Temperature.

4.0 Hardware System

4.1 Purpose

The purpose of testing the 3D printed enclosures is to make sure that they are in fact waterproof enough to prevent water from seeping into the cases in the weather conditions that the cases will be put in. By conducting this testing, it can be ensured that the enclosures can be safely placed outside of windows and will not fall when exposed to those conditions.

4.2 Set up for inside case

1. Put one side of Velcro on the bottom of the case

2. Put the other side of Velcro on the windowsill
3. Make sure the case is set up so that the holes for inputs and outputs are facing the window surface instead of the inside of the room
4. Make sure the case is secured by pressing on it to make it stick tightly to the windowsill
5. Poke the case with a finger or any object around 12mm in diameter
6. Validate that the 12mm object is not making its way into the case

4.3 Set up for outside case

1. Put one side of Velcro on the bottom of the case
2. Put the other side of Velcro on the windowsill
3. Make sure the case is set up so that the holes for inputs and outputs are facing the window surface instead of the inside of the outside
4. Make sure the case is secured by pressing on it to make it stick tightly to the windowsill
5. Poke the case with a finger or any object around 12mm in diameter
6. Validate that the 12mm object is not making its way into the case
7. Pour drops of water on the case using a syringe
8. Validate that water is not going into the case and that the Dragino sensor is dry

4.4 Equipment

Table 20: Testing Equipment.

Equipment	
1	Velcro
2	Outside case
3	Inside case

4.5 Task Results

Table 21: Task Results.

Task		**Result**
1	Test that 12mm object is not making its way into the outside case	N/A

2	Test that 12mm object is not making its way into the inside case	Pass
3	Test that drops of water are not going into the outside case	N/A
4	Test that drops of water are not going into the inside case	Pass

Due to a delay in the printing of the outside case, the testing has not been done to ensure waterproofness. The waterproofness is expected to match an IP rating of x3.

References

[1] The University of British Columbia Campus + Community Planning, “Pathway to a Net Positive Campus: UBC GREEN BUILDING ACTION PLAN.” 2018. Accessed: Oct. 16, 2022. [Online]. Available: https://planning.ubc.ca/sustainability/sustainability-action-plans/green-building-action-plan

[2] C. Chaisson, “What Does ‘Bird-Safe Glass’ Even Mean?,” *Audubon*, Jul. 25, 2014. https://www.audubon.org/news/what-does-bird-safe-glass-even-mean (accessed Oct. 13, 2022).

[3] S. Moreno-Garcia, “Bird’s-eye view - How UBC community members are flocking together to bird-proof campus buildings,” *Medium*, Jan. 26, 2022. https://focus.science.ubc.ca/birds-c2be73bc2f4f (accessed Jan. 23, 2023).

[4] Rainford Solutions, “IP Ratings & Standards Explained.” https://rainfordsolutions.com/products/ingress-protection-ip-rated-enclosures/ip-enclosure-ratings-standards-explained/ (accessed Jan. 23, 2023).

[5] M. Mayntz, “Landscaping Colors That Attract Birds to Your Yard,” *The Spruce*, Apr. 25, 2022. https://www.thespruce.com/colors-that-attract-birds-386400 (accessed Nov. 27, 2022).

[6] Time and Date AS, “Weather in University of British Columbia, British Columbia, Canada,” *timeanddate.com*. https://www.timeanddate.com/weather/@7910029 (accessed Nov. 15, 2022).

[7] S. Chen, D. He, K. Morton, E. Xiong, and S. Vicentijevic, “CAPSTONE Group 062 Bird Impact Detection System Design Document,” 2019. https://open.library.ubc.ca/collections/18861/items/1.0387026

[8] K. Clocker, C. Hu, J. Roadman, R. Albertani, and M. L. Johnston, “Autonomous Sensor System for Wind Turbine Blade Collision Detection,” *IEEE Sensors Journal*, vol. 22, no. 12, pp. 11382–11392, Jun. 2022, doi: 10.1109/JSEN.2021.3081533.

[9] Adafruit, “ADXL343 - Triple-Axis Accelerometer (+-2g/4g/8g/16g) w/ I2C/SPI - STEMMA QT / Qwiic.” Adafruit. Accessed: Apr. 11, 2023. [Online]. Available: https://www.adafruit.com/product/4097

[10] K. L. De Groot, A. N. Porter, A. R. Norris, A. C. Huang, and R. Joy, “Year-round monitoring at a Pacific coastal campus reveals similar winter and spring collision mortality and high vulnerability of the Varied Thrush,” *Ornithological Applications*, vol. 123, no. 3, p. duab027, Aug. 2021, doi: 10.1093/ornithapp/duab027.

[11] S. Hanly, “Accelerometer Specifications: Deciphering an Accelerometer’s Datasheet,” *enDAQ Blog*. https://blog.endaq.com/accelerometer-specifications-decoding-a-datasheet (accessed Nov. 28, 2022).

[12] Analog Devices, “ADXL343 | 3-Axis, ±2 g/±4 g/±8 g/±16 g Digital Accelerometer.” Analog Devices. Accessed: Apr. 11, 2023. [Online]. Available: https://www.analog.com/en/products/adxl343.html

[13] SparkFun Electronics, “H3LIS331DL - Accelerometer, 3 Axis Sensor Evaluation Board.” Digi-Key Electronics. Accessed: Nov. 28, 2022. [Online]. Available: https://www.digikey.ca/en/products/detail/sparkfun-electronics/SEN-14480/8032538

[14] SparkFun Electronics, “LIS331HH MEMS, nano Accelerometer, 3 Axis Sensor Evaluation Board.” Digi-Key Electronics. Accessed: Nov. 28, 2022. [Online]. Available: https://www.digikey.ca/en/products/detail/sparkfun-electronics/SEN-10345/5140799

[15] K. Tuck, “How Many Bits are Enough? The Trade-off Between High Resolution and Low Power Using Oversampling Modes.” NXP Semiconductors, Sep. 2010. [Online]. Available: https://www.nxp.com/docs/en/application-note/AN4075.pdf

[16] Analog Devices Inc., “ADXL355 - Accelerometer, 3 Axis Sensor Evaluation Board.” Digi-Key Electronics. Accessed: Apr. 11, 2023. [Online]. Available: https://www.digikey.ca/en/products/detail/analog-devices-inc/EVAL-ADXL355Z/6557295

[17] F. J, “10 Best Arduino Cameras,” *Wonderful Engineering*, Jul. 26, 2016. https://wonderfulengineering.com/10-best-arduino-cameras/ (accessed Apr. 11, 2023).

[18] R. Keim, “The Nyquist–Shannon Theorem: Understanding Sampled Systems,” *All About Circuits*, May 06, 2020. https://www.allaboutcircuits.com/technical-articles/nyquist-shannon-theorem-understanding-sampled-systems/ (accessed Apr. 11, 2023).

[19] D. Jung and D. Park, “Real Time Sensor Signal Processing Techniques Using Symmetric Dual-Bank Buffer on FreeRTOS,” in *2021 IEEE 3rd Global Conference on Life Sciences and Technologies (LifeTech)*, 2021, pp. 331–332. doi: 10.1109/LifeTech52111.2021.9391911.

[20] F. Miao, R. Zhao, and X. Wang, “A New Method of Denoising of Vibration Signal and Its Application,” *Shock and Vibration*, vol. 2020, p. 7587840, May 2020, doi: 10.1155/2020/7587840.

[21] Analog Devices, “Accelerometer Specifications - Quick Definitions.” https://www.analog.com/en/products/landing-pages/001/accelerometer-specifications-definitions.html (accessed Apr. 11, 2023).

[22] MathWorks, “Time-Frequency Analysis and Continuous Wavelet Transform,” *MathWorks Help Center*. https://www.mathworks.com/help/wavelet/ug/time-frequency-analysis-and-continuous-wavelet-transform.html (accessed Feb. 13, 2023).

[23] Arm Ltd., “CMSIS DSP Software Library,” *CMSIS DSP Software Library Version 1.10.0*, May 02, 2022. https://arm-software.github.io/CMSIS_5/DSP/html/index.html (accessed Apr. 11, 2023).

[24] Arm Ltd., “CMSIS,” *Common Microcontroller Software Interface Standard Version 5.9.0*, May 02, 2022. https://arm-software.github.io/CMSIS_5/General/html/index.html (accessed Apr. 11, 2023).

[25] MathWorks, “Calculate Channel Latencies Required for Wavelet Reconstruction,” *MathWorks Help Center*. https://www.mathworks.com/help/dsp/ug/calculate-the-channel-latencies-required-for-wavelet-reconstruction.html (accessed Apr. 11, 2023).

[26] Dragino Technology Co., Limited, “Dragino LHT65 Temperature & Humidity Sensor - US/CAN - 915 MHz.” CalChip Connect. Accessed: Nov. 28, 2022. [Online]. Available: https://www.calchipconnect.com/collections/dragino/products/dragino-lht65-temperature-humidity-sensor

[27] “9 Arduino Compatible Temperature Sensors for Your Electronics Projects,” *RandomNerdTutorials.com*. https://randomnerdtutorials.com/9-arduino-compatible-temperature-sensors-for-your-electronics-projects/ (accessed Nov. 28, 2022).

[28] “LoRaWAN Temperature & Humidity Sensor, LHT65.” Dragino Technology Co., Limited. Accessed: Nov. 28, 2022. [Online]. Available: https://cdn.shopify.com/s/files/1/0447/5633/6807/files/Datasheet_LHT65_LoRaWAN_Temperature_Humidity_Sensor.pdf?v=1629215041

[29] Arduino, “MKR 1000 WiFi Battery Life,” *Arduino Documentation*. https://docs.arduino.cc/tutorials/mkr-1000-wifi/mkr-1000-battery-life (accessed Feb. 12, 2023).
[30] M. Davies, “How long can an Arduino run on a battery?,” *Quora*. https://www.quora.com/How-long-can-an-Arduino-run-on-a-battery (accessed Feb. 11, 2023).
[31] Amazon Web Services, Inc. or its affiliates., “FreeRTOS FAQ - What is This All About?,” *freertos.org*. https://www.freertos.org/FAQWhat.html#WhyUseRTOS (accessed Apr. 10, 2023).
[32] MathWorks, “ThingSpeak™.” [Online]. Available: https://thingspeak.com/
[33] myDevices, “Cayenne - The world’s first drag-and-drop IoT project builder.” myDevices. Accessed: Nov. 28, 2022. [Online]. Available: https://developers.mydevices.com/cayenne/features/
[34] MongoDB, Inc., “MongoDB.” MongoDB, Inc. Accessed: Feb. 13, 2023. [Online]. Available: https://www.mongodb.com/
[35] Oracle, “MySQL.” Oracle. Accessed: Feb. 13, 2023. [Online]. Available: https://www.mysql.com/
[36] S. Green, “Guide to IP (Ingress Protection) Codes for Vehicle Electrical Components,” *The Waytek Blog*, Oct. 22, 2021. https://info.waytekwire.com/blog/ip-code-2/ (accessed Feb. 12, 2023).
[37] Transport Canada, “Appendix 12.1 — Bird-impact Forces — The Physics,” May 2010. https://tc.canada.ca/en/aviation/publications/sharing-skies-guide-management-wildlife-hazards-tp-13549/appendix-121-bird-impact-forces-physics (accessed Nov. 28, 2022).
[38] “Golden-crowned Kinglet.” All About Birds. Accessed: Nov. 28, 2022. [Online]. Available: https://www.allaboutbirds.org/guide/Golden-crowned_Kinglet/id
[39] “Varied Thrush - Identification.” All About Birds. Accessed: Nov. 28, 2022. [Online]. Available: https://www.allaboutbirds.org/guide/Varied_Thrush/id
[40] “American Robin - Identification.” All About Birds. Accessed: Nov. 28, 2022. [Online]. Available: https://www.allaboutbirds.org/guide/American_Robin/id
[41] “Fox Sparrow - Identification.” All About Birds. Accessed: Nov. 28, 2022. [Online]. Available: https://www.allaboutbirds.org/guide/Fox_Sparrow/id
[42] “Dark-eyed Junco - Identification.” All About Birds. Accessed: Nov. 28, 2022. [Online]. Available: https://www.allaboutbirds.org/guide/Dark-eyed_Junco/id
[43] T. Alerstam, M. Rosén, Johan Bäckman, Per G. P Ericson, and O. Hellgren, “Flight Speeds among Bird Species: Allometric and Phylogenetic Effects,” *PLoS Biology*, vol. 5, no. 8, p. e197, Jul. 2007.
[44] DIYI0T, “Arduino vs ESP8266 vs ESP32 Comparison,” *DIYI0T*. https://diyi0t.com/technical-datasheet-microcontroller-comparison/ (accessed Nov. 28, 2022).
[45] “MongoDB vs MySQL,” *Software Advice*. https://www.softwareadvice.com/ca/database-management-systems/mongodb-profile/vs/mysql/ (accessed Nov. 28, 2022).
[46] “MongoDB Pricing,” *MongoDB*. https://www.mongodb.com/pricing (accessed Nov. 28, 2022).
[47] MathWorks, “Student License,” *ThingSpeak*. https://thingspeak.com/prices/thingspeak_student?license_name=Student&number_of_devices=10&intervals=2&interval_type=Minutes (accessed Feb. 13, 2023).

[48] "ThingSpeak," *Software Advice*. https://www.softwareadvice.com/ca/bi/thingspeak-profile/ (accessed Feb. 13, 2023).
[49] S. Majocchi, "Powering MKR WiFi 1010 with Batteries," *Arduino Documentation*. https://docs.arduino.cc/tutorials/mkr-wifi-1010/powering-with-batteries (accessed Feb. 12, 2023).

Revisions

Rev.	Date (DD/MM/YYYY)	Author(s)	Change description
00	16/10/2022	Mohamed Salah	Initial release.
01	22/11/2022	Huawen Li Gengran Li Ryotaro Hokao Benjamin Powell Mohamed Salah	Renamed 1.0 High-level Design Diagram to 1.0 System Architecture. Removed 2.0 Design Decisions. Added the following sections: • 2.0 Bird Collision Detection • 3.0 Temperature ○ Microcontroller • 5.0 Data Storage & UI • 6.0 Next Steps • References • Appendices
02	13/02/2023	Huawen Li Gengran Li Ryotaro Hokao Benjamin Powell Mohamed Salah	Added the following sections: • System Architecture • 2.5 Detection Algorithm Development • 5.0 Communication System • 7.0 Hardware
03	11/04/2023	Huawen Li Gengran Li Ryotaro Hokao Benjamin Powell Mohamed Salah	Combined Requirements, Deliverables, Design, and Verification and Validation Documents. Renamed 2.0 Bird Collision detection to Detection System Added the following sections: • 3.0 Detection Algorithm • Accelerometer Fall Detection • 6.0 Outside Ambient Air

			Temperature Detection • 8.0 Task scheduling algorithm
04	11/05/2023	Ryotaro Hokao Benjamin Powell	Fixed Table and Figure references. Fixed Multi-level list, and section references. Fixed misspelled words. Added Executive Summary.

Appendices

Appendix I: Report Contribution

Section	Major content	Minor content	Author	Reviewer
Requirements	GL,HL		GL,HL	BP,RH,MS
Deliverables	BP		BP	RH
System Architecture	BP,RH		BP	HL,GL,RH,MS
Detection System	RH		BP,RH	BP, MS
Temperature	MS	HL	MS	HL, BP, RH, MS, GL
Microcontroller	BP, GL		BP,GL	HL, BP, RH, MS
Task Scheduling Algorithm	GL	BP	GL	RH
Communication	BP		BP	GL, RH, MS,HL
Data Storage & UI	HL		HL	BP,GL,RH, MS
Hardware	MS	MS	BP,MS	RH, MS
V&V	BP,RH,MS,GL, HL		MS,GL,HL	BP,RH,MS,GL, HL

Appendix II: Impact force estimation

This section outlines the process of estimating the impact force for bird-window collisions. Impact force can be computed using the following formula: $F = mv^2/2d$ [37], where m is the mass of different bird species, v is the velocity of bird, and d is the distance over which impact is delivered (length of the bird). Calculated force in Newtons can then be converted to g-force by: $F/9.8$.

As discussed in section 2.2.3, impact force is estimated for the following five bird species:

- Varied Thrush
- American Robin
- Fox Sparrow
- Darkeyed Junco
- Golden-Crowned Kinglet

Mass, length, and velocity data for each species are collected, and impact force is estimated.

Table 22: Mass, speed, and lengths data for each bird species [38]–[43].

Species	Min mass [g]	Max mass [g]	Min speed [m/s]	Max speed [m/s]	Min length [cm]	Max length [cm]
Varied Thrush	65	100	1	15	19	26
American Robin	77	85	1	15	20	28
Fox Sparrow	26	44	1	15	15	19
Darkeyed Junco	18	30	1	15	14	16
Golden-Crowned Kinglet	4	8	1	15	8	11

Table 23: Estimated minimum and maximum g-force for each bird species. The minimum and maximum g-force for all five species are highlighted.

Species	Min g-force [g]	Max g-force [g]
Varied Thrush	0.01275510204	6.04189044
American Robin	0.01403061224	4.878826531
Fox Sparrow	0.006981740064	3.367346939
Darkeyed Junco	0.005739795918	2.459912536
Golden-Crowned Kinglet	0.00185528757	1.147959184

Appendix III: Complete list of accelerometer candidates

Table 24: Full list of accelerometer candidates.

Sensor candidates	Price	Sensor type	Measurement range [+/-g]	Sensitivity	Operating temperature [°C]	Operating voltage [V]	Interface	URL	Notes
Adafruit Industries LLC, ADXL345 Breakout Board	CA$25.76	Capacitive, Digital	2,4,8,16 selectable	31.2mg/LSB for +/- 16g	-40 to 85	3.3V compatible	SPI, I2C	https://www.digikey.ca/en/products/detail/adafruit-industries-llc/1231/4990764	Arduino library available
DFRobot, ADXL345 Breakout Board	CA$11.63	Capacitive, Digital	2,4,8,16 selectable	31.2mg/LSB for +/- 16g	-40 to 85	3.3V compatible	SPI, I2C	https://www.digikey.ca/en/products/detail/dfrobot/SEN0032/6588489	Arduino library available
STMicroelectronics, AIS3624DQ Breakout Board	CA$23.49	Capacitive, Digital	6,12,24 selectable	5.9mg/LSB for +/- 12g	-40 to 105	3.3V compatible	SPI, I2C	https://www.digikey.ca/en/products/detail/stmicroelectronics/STEVAL-MKI158V1/5180532	May arrive with I/O pins soldered. No library available
Analog Devices Inc., ADXL357 Evaluation Board	CA$67.08	Digital	10,20,40 selectable	0.039mg/LSB for +/- 20g	-40 to 125	3.3V compatible	SPI, I2C	https://www.digikey.ca/en/products/detail/analog-devices-inc/EVAL-ADXL357Z/8554443	May arrive with I/O pins soldered. No library available
SparkFun Electronics, STMicroelectronics H3LIS331DL Evaluation Board	CA$19.87	Capacitive, Digital	100, 200, 400 selectable	49mg/LSB for +/- 100g	-40 to 85	3.3V compatible	SPI, I2C	https://www.digikey.ca/en/products/detail/sparkfun-electronics/SEN-14480/8032538	Arduino library available
SparkFun Electronics, STMicroelectronics LIS331HH Evaluation Board	CA$21.86	Capacitive, Digital	6,12,24 selectable	6mg/LSB for +/- 12g	-40 to 85	3.3V compatible	SPI, I2C	https://www.digikey.ca/en/products/detail/sparkfun-electronics/SEN-10345/5140799	Example firmware for AVR microcontroller available
Adafruit Industries LLC, ADXL343 Evaluation Board	CA$8.76	Capacitive, Digital	2,4,8,16	31.2mg/LSB for +/- 16g	-40 to 85	3.3V compatible	SPI, I2C	https://www.digikey.ca/en/products/detail/adafruit-industries-llc/4097/9951931	Arduino library available
Senther Technology, 540C Embedded Piezoelectric Accelerometer	CA$55.85	Piezoelectric, Analog	50 - 2000	40mV/g for +/- 50g	-40 to 125	3.3V and 5V compatible	Analog	https://www.digikey.ca/en/products/detail/senther-technology/540C/16634003	Decoupling capacitor and other components required. Arduino needs to be able to resolve the measurement
Senther Technology, 540A Embedded Piezoelectric Accelerometer	CA$59.48	Piezoelectric, Analog	500	5mV/g typical	-55 to 150	3.3V and 5V compatible	Analog	https://www.digikey.ca/en/products/detail/senther-technology/540A/13574361	Decoupling capacitor and other components required. Arduino needs to be able to resolve the measurement

Appendix IV: Complete Comparison Table of Microcontroller candidates

Table 25: Microcontroller Candidates vs. Imperative Criteria for Project Success [44].

	Arduino Uno WiFi Rev2	Raspberry Pi 3 Model B+	Arduino NANO 3	Arduino Uno R3	Arduino Mega	ESP32 NodeMCU	ESP8266 Thing	Arduino MKR WiFi 1010
Number / type of I/O (Digital)	14	26	14	14	54	36	17	14
Number I/O pin (Analog)	6/0	9	8/0	6/0	16/0	15	16/0	7/1
Price ($)	44.9	135	22	22	38.5	17.8	18.5	38.6
Size	69*35mm	85 *56mm	45*18mm	69*53mm	102*53mm	52*31mm	80*30mm	61.5*25mm
Programming Language	C/C++/ Python	C/C++/ Python	C/C++	C/C++	C/C++	python	python	C/C++
Bluetooth	no	yes	no	no	no	yes	yes	yes
USB	yes	yes	yes	yes	yes	yes	yes	yes
Temperature Sensor	no	no	no	no	no	yes	yes	no
WIFI Availability	yes	yes	no	no	no	yes	yes	yes
Flash Memory	48KB	[use flash card]	32KB	32KB	256KB	4M	512KB	256KB
Power jack	yes	yes	no	yes	yes	no	no	[Has external power jack shell]
Weight	25g	50g	5g	25g	37g	10g	11g	32g
Processor speed	20MHz	1.2Ghz	16MHz	16MHz	84MHz	80 MHz	160MHz	48MHz
Cayenne Supported	Yes	Yes	No	No	No	Yes	Yes	Yes

ThingSpeak Supported	Yes	Yes	No	No	No	No	Yes	Yes

Appendix V: Flash Memory Calculations

In this section, the process of determining the minimum flash memory capacity for the microcontroller is discussed.

To begin with, the team utilized the code for the base detection system and tested it on the Arduino Uno WiFi Rev2 to assess its flash memory requirements. Considering that the flash memory capability of Arduino Uno WiFi Rev2 falls within the mid-range among the candidate microcontrollers, the amount of flash memory needed to run the bird impact data detection system on Arduino Uno WiFi Rev2 was found to be 200% of its flash memory capacity. This resulted in a minimum requirement of 96KB of flash memory for the entire detection system, calculated as 200% of 48KB [200% x 48KB =96KB]. The minimum flash memory requirement for the data transfer code was determined to be 43.2KB. Hence, the total minimum flash memory capacity needed for the microcontroller was determined to be 140KB.

Appendix VI: Sample Collected Data

created_at	entry_id	InsideSurfaceTemp	OutsideSurfaceTemp	OutsideAirTemp	HeatFlowRate	CollisionNumber	CollisionDatafromArduinc	FallDetecitonData
2023/4/9 6:04:02	1452				0.060666667			
2023/4/9 6:08:55	1453			7.9				
2023/4/9 6:11:13	1454		19					
2023/4/9 6:15:07	1455	18.93						
2023/4/9 6:33:54	1456				0.032666667			
2023/4/9 6:38:55	1457			8.1				
2023/4/9 6:41:13	1458		19.06					
2023/4/9 6:45:07	1459	18.93						
2023/4/9 7:03:43	1460				0.060666667			
2023/4/9 7:08:55	1461			8.2				
2023/4/9 7:11:13	1462		19					
2023/4/9 7:15:07	1463	18.87						
2023/4/9 7:33:54	1464				0.060666667			
2023/4/9 7:38:54	1465			8.3				
2023/4/9 7:41:13	1466		19					
2023/4/9 7:45:07	1467	18.87						
2023/4/9 8:03:43	1468				0.060666667			
2023/4/9 8:08:54	1469			8.4				
2023/4/9 8:11:13	1470		19					
2023/4/9 8:15:07	1471	18.87						
2023/4/9 8:33:45	1472				0.060666667			
2023/4/9 8:39:08	1473			8.6				
2023/4/9 8:41:13	1474		19					
2023/4/9 8:45:07	1475	18.87						
2023/4/9 9:03:27	1476				0.060666667			
2023/4/9 9:09:05	1477			8.8				
2023/4/9 9:11:13	1478		19					
2023/4/9 9:15:07	1479	18.87						
2023/4/9 9:33:45	1480				0.060666667			
2023/4/9 9:38:54	1481			8.9				
2023/4/9 9:41:13	1482		19					
2023/4/9 9:45:07	1483	18.87						
2023/4/9 10:03:43	1484				0.060666667			
2023/4/9 10:08:53	1485			8.7				
2023/4/9 10:11:13	1486		19					
2023/4/9 10:15:07	1487	18.87						
2023/4/9 10:33:52	1488				0.060666667			
2023/4/9 10:38:54	1489			9.1				
2023/4/9 10:41:13	1490		19					
2023/4/9 10:45:07	1491	18.87						
2023/4/9 11:03:41	1492				0.060666667			
2023/4/9 11:09:08	1493			9.1				
2023/4/9 11:11:13	1494		19					
2023/4/9 11:15:07	1495	18.87						
2023/4/9 11:33:33	1496				0.060666667			
2023/4/9 11:39:07	1497			9.2				
2023/4/9 11:41:13	1498		19					
2023/4/9 11:45:07	1499	18.89						
2023/4/9 12:04:13	1500				0.051333333			
2023/4/9 12:08:53	1501			9.6				
2023/4/9 12:11:13	1502		19					
2023/4/9 12:15:08	1503	18.93						
2023/4/9 12:33:33	1504				0.032666667			
2023/4/9 12:38:55	1505			9.8				
2023/4/9 12:41:13	1506		19					
2023/4/9 12:45:07	1507	18.93						
2023/4/9 13:03:53	1508				0.032666667			
2023/4/9 13:04:08	1509					0		
2023/4/9 13:05:08	1510							0
2023/4/9 13:09:08	1511			10				
2023/4/9 13:11:13	1512		19					
2023/4/9 13:15:07	1513	18.93						
2023/4/9 13:33:28	1514				0.032666667			
2023/4/9 13:39:16	1515			10				
2023/4/9 13:41:13	1516		19					
2023/4/9 13:45:07	1517	18.93						
2023/4/9 14:04:04	1518				0.032666667			
2023/4/9 14:08:54	1519			10.1				
2023/4/9 14:11:13	1520		19					
2023/4/9 14:15:07	1521	18.93						
2023/4/9 14:33:44	1522				0.032666667			
2023/4/9 14:39:05	1523			10.2				
2023/4/9 14:41:13	1524		19					
2023/4/9 14:45:07	1525	18.93						
2023/4/9 15:03:41	1526				0.032666667			
2023/4/9 15:08:53	1527			10.5				
2023/4/9 15:11:13	1528		19					
2023/4/9 15:15:07	1529	19						
2023/4/9 15:33:38	1530				0			
2023/4/9 15:38:54	1531			10.7				
2023/4/9 15:41:13	1532		19					
2023/4/9 15:45:07	1533	19.02						
2023/4/9 16:03:03	1534				0.009333333			
2023/4/9 16:08:01	1535			10.9				
2023/4/9 16:11:06	1536		19					

2023/4/9 16:15:07	1537	19.02									
2023/4/9 16:33:21	1538				0.009333333						
2023/4/9 16:38:54	1539			10.8							
2023/4/9 16:41:13	1540		19								
2023/4/9 16:45:35	1541	19									
2023/4/9 16:48:50	1542						1				
2023/4/9 16:50:38	1543							164751			
2023/4/9 16:50:59	1544							1			
2023/4/9 16:53:17	1545								1		
2023/4/9 17:04:13	1546				0						
2023/4/9 17:08:53	1547			10.7							
2023/4/9 17:11:13	1548		19								
2023/4/9 17:15:08	1549	19									
2023/4/9 17:33:21	1550				0						
2023/4/9 17:38:55	1551			10.7							
2023/4/9 17:41:13	1552		19								
2023/4/9 17:45:07	1553	19									
2023/4/9 18:03:53	1554				0						

Figure 39: Example of ThingSpeak data exported as a spreadsheet.

Appendix VII: Minimum Data Storage Calculation Process

$$Number\ of\ outside\ surface\ tempearture\ data\ entries\ per\ day = \left(\frac{60\ mins}{30\ mins} * 24hrs\right) = 48$$

$$Number\ of\ inside\ surface\ tempearture\ data\ entries\ per\ day = \left(\frac{60\ mins}{30\ mins} * 24hrs\right) = 48$$

$$Number\ of\ outside\ air\ tempearture\ data\ entries\ per\ day = \left(\frac{60\ mins}{30\ mins} * 24hrs\right) = 48$$

$$Number\ of\ heat\ flow\ rate\ data\ entries\ per\ day = \left(\frac{60\ mins}{30\ mins} * 24hrs\right) = 48$$

$$Number\ of\ collision\ detection\ data\ entries\ per\ day$$
$$= 2\ \ collision\ data\ segments\ sent\ from\ the\ microcontroller$$
$$+ 1\ analyzed\ bird\ impact\ counts\ data\ segments\ being\ stored\ in\ the\ database\ = 3$$

$$Number\ of\ fall\ detection\ data\ entries\ per\ day = 1$$

$$Number\ of\ total\ data\ entries\ per\ day = 48 * 4 + 3 + 1 = 196$$

$$Total\ storage\ size\ required\ each\ day = \left(\frac{3790\ bytes\ within\ 12\ hrs}{102\ data\ entries\ within\ 12\ hrs} * 196\ data\ entries\ per\ day\right) \approx 7283\ bytes\ per\ day$$

$$7283\ bytes\ per\ day = \frac{7283}{1024}\ kilobytes\ per\ day = 7.1KB\ per\ day$$

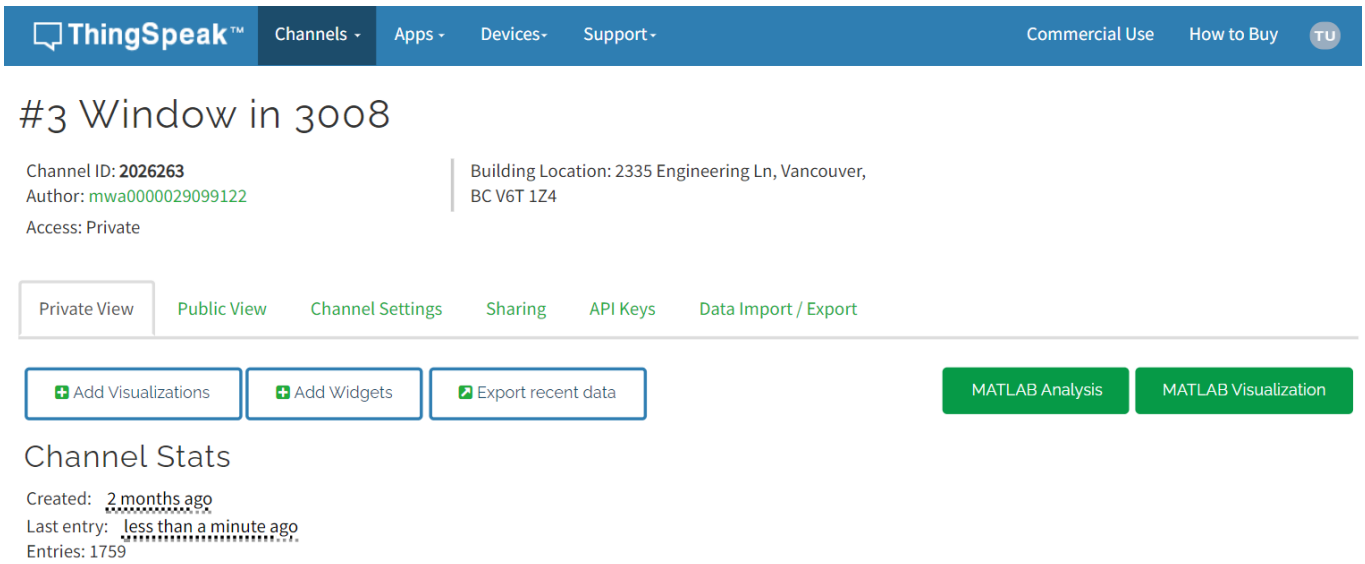
$$Total\ storage\ size\ required\ per\ \ 30\ day\ period = \ 7.1KB\ per\ day * 30 = 213KB$$

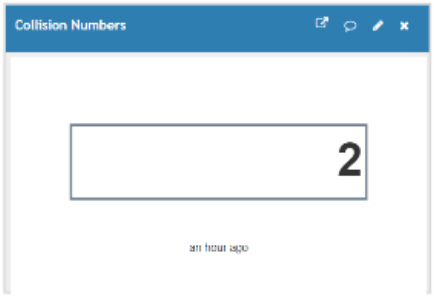
Appendix VIII: Complete Comparison Table of Data Storage Candidates

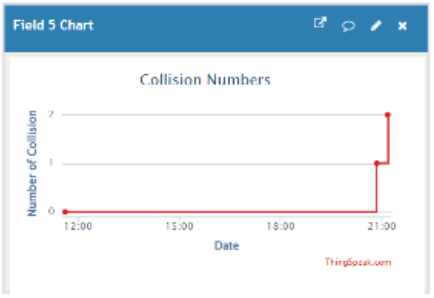
Table 26: Data Storage Evaluation Comparison [7], [33], [45]–[48].

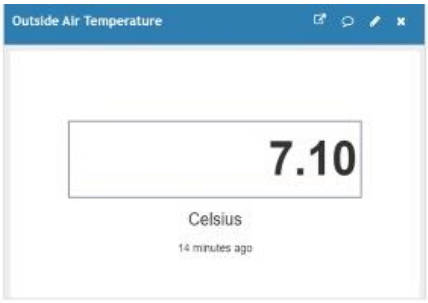
Feature	Cayenne	MongoDB	MySQL	ThingSpeak
Storage Size	Unlimited	512GB	Unlimited	~8200 messages/day
Data Segment Length	Unlimited	Unlimited	Unlimited	Unlimited
Unique Data Segments	Unlimited	Unlimited	Unlimited	~8200 messages/day
Build-in Analytics Tool	No	Yes	Yes	Yes
Self-Contained User Interface Tool	Yes	Yes	No	Yes
Deployments	- Cloud, SaaS, Web-Based -Desktop: Windows, Mac, Linux -Mobile: Android, iPhone	- Cloud, SaaS, Web-Based -Desktop: Windows, Mac, Linux -Mobile: not available	- Cloud, SaaS, Web-Based -Desktop: Windows, Mac -Mobile: not available	- Cloud, SaaS, Web-Based

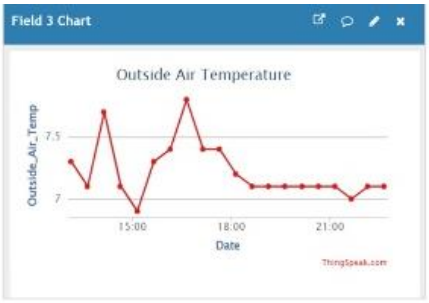
Appendix IX: User Interface Visualisation




Collision Numbers
2
an hour ago


Field 5 Chart
Collision Numbers
Number of Collision
Date
ThingSpeak.com

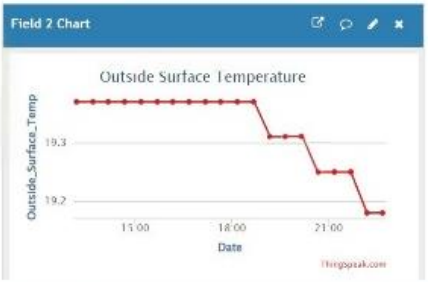

Outside Air Temperature
7.10
Celsius
14 minutes ago


Field 3 Chart
Outside Air Temperature
Outside_Air_Temp
Date
ThingSpeak.com


Inside Surface Temperature
19.12
Celsius
8 minutes ago


Field 1 Chart
Inside Surface Temperature
Inside_Surface_Temp
Date
ThingSpeak.com


Outside Surface Temperature
19.18
Celsius
12 minutes ago


Field 2 Chart
Outside Surface Temperature
Outside_Surface_Temp
Date
ThingSpeak.com


Heat Flow Rate
0.028
Watts
20 minutes ago

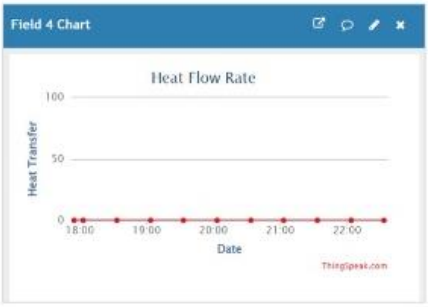

Field 4 Chart
Heat Flow Rate
Heat Transfer
Date
ThingSpeak.com

Figure 40: ThingSpeak dashboard with counters and charts to visualize the collision count, heat flow rate, temperatures, and accelerometer status.

Appendix X: Communication Protocols and Authorizations

Table 27: Definitions of varying types of communication protocols and authorizations.

HTTP (Hypertext Transfer Protocol)
An application protocol that enables the transfer of data over the internet. It is the foundation of data communication for the World Wide Web and is used to fetch resources such as HTML files, images, and videos from web servers to web browsers. HTTP runs on top of the TCP/IP network protocol and generally uses port 80.
HTTPS (Hypertext Transfer Protocol Secure)
An extension of HTTP that provides security features such as data encryption and server authentication. It operates on top of the SSL/TLS encryption protocol to ensure that data transmitted between the client and server is secure and cannot be intercepted by third parties. HTTPS typically uses port 443.
MQTT (Message Queuing Telemetry Transport)
A lightweight messaging protocol designed for use in constrained environments with limited bandwidth and low computational power. It is commonly used in IoT applications where devices need to communicate with each other and with servers. MQTT is ideal for machine-to-machine (M2M) communication because it requires minimal network bandwidth, supports bi-directional communication, and allows for disconnected operation. Its efficiency and scalability have made it a popular choice for IoT applications, especially those involving sensors and remote devices.
IEEE 802.11
A set of standards for wireless local area networks (WLANs) developed by the Institute of Electrical and Electronics Engineers (IEEE). The standards specify the physical and data link layer protocols for wireless communication, ensuring that devices from different manufacturers can interoperate with each other.
OAuth 2.0
An authorization protocol that enables third-party applications to obtain limited access to a user's data on another web service without requiring the user's credentials. It provides a secure way to access an API by using a token that represents the authorization granted to the application. OAuth

2.0 can be used with various types of APIs and is commonly used for social networking, digital media, and cloud computing services.
API Key
A unique identifier that grants access to an API (Application Programming Interface) to an application or user. It serves as a security measure by preventing unauthorized access and tracking API usage. An API Key is usually included in API requests and is validated by the server to determine if the request is authorized. It is commonly used for payment gateways, social networking, and other web-based services.
LPWAN
LPWAN, or Low-Power Wide-Area Network, is a wireless communication protocol designed for Internet of Things (IoT) devices. LPWAN allows for long-range communication with low power consumption, making it ideal for IoT applications that require devices to operate for years on a single battery charge. LPWAN protocols use unlicensed frequency bands and spread-spectrum technology to transmit data wirelessly over long distances with minimal infrastructure requirements.

Appendix XI: Microcontroller Battery Charging Calculation

This section focuses on calculating the power consumption and battery charge life of the microcontroller when running the required program. Based on research [30], [49], the WiFi connection is the key function of the Arduino MKR WiFi 1010 board that impacts battery life: when linked to an access point and data transfer is in progress, it consumes roughly 100mA [29]. If the WiFi NINA module is not in use, the microcontrollers continual running tasks use roughly 20 mA at that time. Based on the battery life calculation equation:

Battery Life = (Battery Capacity)/(Average Current Consumption) *0.9

[The factor of 0.9 is used because it considers other factors that may affect battery life, such as ambient temperature, cycling, and battery chemistry.]

On average, the WiFi module takes 1 minute to transfer information to ThingSpeak. The average number of times it is used in a day is once, and the total time used is 1*1 = 1 minutes, which occupies 0.006944% of the time in a day. Therefore, the average energy consumption for a day is 0.006944 % * 100mA + 99.305%*30mA = 20.04859 mA. Using our selected 3.7v 3200mAH lithium-ion battery, the microcontroller can be powered for up to [(3200 / 24) / (30.4859)] *0.9 = 6.648days.

Appendix XII: Ranking of Criteria for microcontroller

Table 28: Ranking of Criteria Points with respect to success of the project via requirements and constraints.

Criteria	Importance
Number / type of I/O (Digital)	High

WiFi Availability	High
Flash Memory	High
Number I/O pin (Analog)	Mild-High
Power jack	Mild-High
Price ($)	Mild-High
Size	Medium
Weight	Medium
Processor speed	Medium
USB Connection	Low
Bluetooth	Low
Programming Language	Low
Integrated Temperature Sensor	Low

Appendix XIII: Cayenne Information

Cayenne, a free data service platform provided by myDevices [33], was initially considered as the primary option for data storage and user interface in our system. It meets or exceeds all our minimum requirements for data storage and offers customizable dashboards to satisfy our user interface needs. Additionally, it supported data from multiple microcontrollers and other hardware through unique identifiers and wireless networks. These were the main reasons why we implemented Cayenne in the early stages of our system.

However, as we looked to simplify the system and reduce costs, we replaced Cayenne with ThingSpeak. ThingSpeak was able to analyze the collected data using its built-in analytics tools, whereas Cayenne failed to calculate the heat flow rate by sharing surface temperature data from two separate sensors. As a result, Cayenne was not included in the final design of the system.