CASA0022 BeeSmart
Automating the Remote Monitoring of Beehive Health

Final Project Report

Author: Patrick Whyte

Supervisor: Duncan Wilson

Student ID: 21199611

August 24, 2023

Abstract

BeeSmart is a smart connected device made to empower beekeepers with seamless remote monitoring of beehive health. BeeSmart is outfitted with four 50kg load sensors that accurately measure beehive weight. Additionally, it features two weather resistant temperature probes, providing beekeepers with the flexibility to insert them directly into their beehives at the specific locations of their choosing. BeeSmart is powered by the Arduino MKR 1010, serving as the brains of the device. It seamlessly interfaces with these sensors, capturing vital beehive data in real-time and integrating the data with the MathWorks ThingSpeak API. This allows valuable information to be effortlessly stored and presented in visually captivating displays for beekeepers on the BeeSmart dashboard. This report will describe the development, experimental setup and evaluation of the device built for this project and provide a conclusion that evaluates the performance of the device while discussing possible future work that could be done to further enhance this project in the next generations of the device.

Originality Avowal

I, Patrick Whyte, hereby declare that this dissertation is my original work and that all sources have been acknowledged. I grant the right to UCL to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival.

Patrick Whyte

August 24, 2023

Word count: 5315

Acknowledgements

I would like to acknowledge and thank Simon Saville from the London Beekeepers' Association who provided me with invaluable insight into bee keeping. I would also like to thank my supervisor Duncan Wilson for his support during this project.

Contents

1 Introduction **3**

2 Background **4**

2.1 Why it is Important to Monitor Bees 4

2.2 Expert Interview 5

2.3 Other Devices 6

3 Design & Building **8**

3.1 Sensors and Microcontroller 9

3.1.1 Weight 9

3.1.2 Temperature 10

3.1.3 Microcontroller 10

3.2 Circuit Diagram and Wiring 11

3.2.1 Circuit Diagram 11

3.2.2 Soldering 12

3.3 Software 13

3.3.1 Setup 13

3.3.2 Main Loop 14

3.3.3 Dashboard 16

3.4 Enclosure 17

3.4.1 Waterproofing 18

3.4.2 Weight Plate 19

3.4.3 Putting it all together 20

4 Experimental Setup and Methodology **21**

5 Evaluation **24**

5.1 Temperature 24

5.2 Weight 25

6 Conclusion **28**

6.1 Future Work 28

6.2 Conclusion 29

References **31**

A GitHub Links **32**
A.1 Link to the GitHub Repository 32
A.2 Link to the BeeSmart Dashboard 32

B Supervisor Meetings **33**

C Source Code **34**
C.1 Arduino Code 34

Chapter 1

Introduction

For the final project of my MSc in Connected Environments I was tasked to develop an extended personal research project with a focus to Sense, Deploy and Communicate. The central aim of the project is to build and deploy a device that creates a Connected Environment. My interest was to focus on an area that could help provide a deeper understanding of our natural biodiversity. In London there has been a growing interest in the hobby of urban beekeeping leading to an exponential rise in the number of bee colonies within our city[1]. Bees have an instrumental role in almost every ecosystem, acting as pollinators for the plants that feed all types of animals.

I decided that my Connected Environment would explore this relationship between beekeepers and their beehives. In order to create my Connected Environment I built a device which I called BeeSmart, made to sense vital environmental data from the beehive in real-time and communicate the live information directly to the beekeeper. This device was deployed in a simulated environment, where the real-time data gathered was presented on a dashboard made for the beekeepers to remotely keep track of the health of their beehives.

This report describes the development, experimental setup and evaluation of the BeeSmart device built for this project. Background information is provided on the importance of such a device, integrating my interview with an expert in the field and an evaluation of other devices that aim to connect the same environment. Finally, the report provides a conclusion about the device and its performance as a whole, while discussing possible future work that could be done to further enhance the understanding of beehives environment in the next generations of the device.

Chapter 2

Background

2.1 Why it is Important to Monitor Bees

Bees play a vital role in the ecosystem. The Woodland Trust says that without bees, it wouldn't be long before our ecosystem collapsed. Bees pollinate our trees and flowers, which then support other insects, which then support birds, bats, mammals and everything up the food chain [10]. Bees not only support our entire ecosystem but they can also act as highly efficient biomonitors providing researchers with invaluable information about environmental contaminants, pathogens, and climate change [5]. While individual bees may be susceptible to these environmental stressors, the colony as a whole can accumulate these contaminants and respond to them without the beehive collapsing [5].

NASA Scientist, Wayne Esaias, conducted a study that has shown how honey bees can be used to help understand climate change. By creating a network of citizen scientists called "HoneyBeeNet", who weighed their hives each day, Esaias was able to analyse the annual nectar flow rate from these hives and compare it to satellite data that measured the annual "green up" of vegetation in the spring. They corresponded nearly perfectly, confirming the usefulness of the citizen-science derived data from HoneyBeeNet to address changes in nectar flows [11].

In the city of London there has been a massive increase in the number of urban beekeepers. The London BeeKeepers Association produced an internal report in 2019 called the London Bee Situation that estimated that in Greater London, there were 4,844 registered colonies and likely even more when factoring in feral and unregistered colonies [1]. The increase in attention and enthusiasm towards urban beekeeping has only further increased the number of colonies since this report was published. With all these new urban beekeepers there is an excellent

opportunity for researchers to gain invaluable information.

I believe that London is the perfect city to leverage the power of smart connected devices to gather crucial data on the countless colonies of bees found in our urban environment. By building an open sourced device, we could provide a way for these urban beekeepers to gather real-time data on the health of their colonies and share this live data with researchers. They could leverage this new information to make faster and more informed decisions to the benefit of society as a whole.

2.2 Expert Interview

When researching this project I decided to contact the London BeeKeepers Association (LBKA) to see if they would be able to provide me with any first hand experience on the beekeeping process. Since the scope of the project was so wide I was looking for information that could help me narrow down what types of data is truly most helpful to beekeepers. I was able to interview Simon Saville from the LBKA, an expert in the field of beekeeping, who provided me with valuable information about the environmental data that would be most useful to have generated by a smart beehive device.

While talking with Mr. Saville, it became clear that temperature is a very important aspect to beehive health, as it can provide valuable insight about the activity level of bees. He specifically mentioned that having multiple temperature probes would be helpful. With multiple temperature sensors, the beekeeper can be informed if a specific section of the beehive is too cold during winter which in turn could potentially prevent bees from reaching other sections of the hive (where food or brood are). It can also help ensure that the temperature around the queen and brood stays within an optimal temperature range as they are sensitive to changes in temperature.

Weight was also identified as another very important data point for beekeepers as it can indicate the rate at which the bees are producing honey. Mr. Saville went on to say how weight can inform us as to when the beehive is ready to be harvested for honey. Another use for weight sensors had to do with the fact that during the winter, the population of bees within the beehives dramatically decreases. The weight of the beehive can indicate if the entire beehive is dangerously underweight which could lead to beehive collapse.

Additionally, Mr. Saville mentioned that sound can be an important indicator for beekeepers as it gives them insight into the bees' temperament. For example, if the bees are making lots of noise it could be because a predator is attacking their beehive. Alerting a beekeeper of

specific noises produced by the beehive could give the beekeeper early warning needed to ward off potential predators. I asked Mr. Saville if air quality sensors or humidity sensors would be useful data points to monitor in real-time and he did not seem to think these specific data points would be particularly important to beekeepers to monitor in real-time. However, he mentioned that the surrounding weather data would be useful to provide further context to the other data gathered by the device.

After the interview with Mr. Saville, I decided I would focus on the two main environmental factors that Mr. Saville thought were most important for beekeepers to have real-time data: temperature and weight.

2.3 Other Devices

Continuing my research for this project I decided to investigate similar devices on the market to gain an understanding of what each of them prioritise in their devices. It seems that the top of the line product for smart connected beehives comes from a company called Beewise. Their product, described as the first robotic beehive, has a multitude of features such as; climate and humidity control, automated harvesting of honey, real-time problem alerts and more. The Beewise beehive is an outstanding device for beekeepers with large colonies of bees. However, it is a serious investment costing around $400 per month per colony with an additional $2000 delivery fee. The Beewise device is also made to completely replace your existing beehives as the device itself acts as its own beehive. As impressive as the device is, I figured the aim for my project would be to build something affordable and open sourced that could be utilised by urban beekeepers that tend to have much smaller colonies [4].

The next device I wanted to look into was a weight focused device called BeeScales. This device allows the users to use their own beehives by employing two load cells that sit directly below the beehive. The BeeScales device comes with some helpful solar panels to power the device without the need to connect the device to a wall outlet. The device also comes with an app that allows the user to view the live real-time data from their phones. While this device is great for this single data point it lacks the ability to sense anything other than weight and with a price of £490 it is still a considerable investment for a smaller beekeeper [3].

The last device I wanted to investigate was the Hive Heart 3.0. This device's main purpose is to monitor the beehive's temperature and humidity. At only £56 it is definitely on the affordable side of these devices which makes it a great choice for smaller urban beekeepers. The Hive Heart also comes with a helpful app to monitor the live data which is a great addition

to the product. However, the Hive Heart is only able to sense temperature and humidity data from one location as well as only taking these readings from the surface of the beehive which loses context on how the beehive is doing inside [8].

Chapter 3

Design & Building

The first step in designing and building the smart beehive was to decide on the scope of the project and identify what types of information would be most useful for the beekeepers while also focusing on what would be achievable given the limited time frame. Considering my preliminary research, and information gathered from my interview with Simon Saville, I decided to focus on the two most important environmental factors to be monitored for this project: temperature and weight.

When designing the device there were several factors to keep in mind. I wanted it to be open sourced, affordable and possible for any beekeeper to easily build themselves. It needed to be versatile allowing it to be used with a wide variety of different beehives, requiring minimal intervention from the beekeeper, and allowing them to easily monitor the beehive health remotely. Keeping this in mind, I created a design similar to a digital scale, with two wired temperature probes that freely stuck out from the device. In this way any beehive could be placed on top of the device and the two temperature probes could be inserted into the beehive at the desired locations.

3.1 Sensors and Microcontroller

3.1.1 Weight

The average weight of a beehive can vary a lot depending on the type of beehive, and whether the beehive is full of honey and bees or not. Even the time of year can have an effect on the weight of beehives. The PerfectBee provides a guide to the average weight of beehives which can be around 18-23 kg for smaller beehives, and between 32-41 kg for larger beehives assuming the beehives are full of honey [2].

I decided to use four 50kg load cells with a HX711 load cell amplifier to measure the weight of the beehive. This combination is not only affordable and easily available (I purchased mine on Amazon), but there is also a lot of helpful information online explaining how to set-up and use these sensors, in combination with the Arduinos, making them perfect for this project.

Figure 3.1 is a picture of one of the load cells.

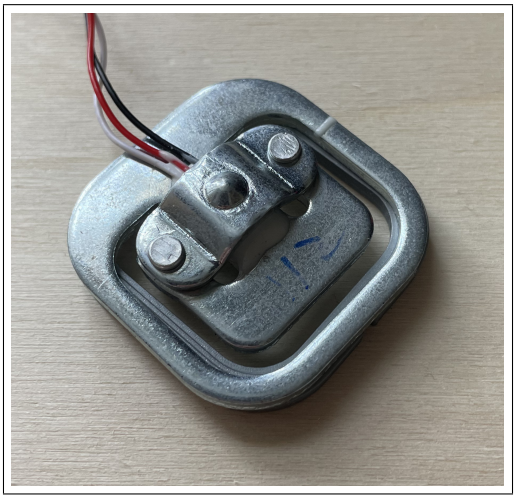


Figure 3.1: Load Cell

3.1.2 Temperature

When deciding which temperature sensors to use, I was again considering affordability and ease of use. The main objective was to ensure that there would be no problems inserting the device into a beehive, surrounded by honey and beeswax, and that it should be resistant to rain and dirt because the temperature probes are outside of the main enclosure and are therefore open to the elements. With these requirements in mind, I decided to go with the DS18B20 1-Wire Digital Temperature Probe Sensor.

Figure 3.2 is a picture of the tip of the temperature sensor.

Figure 3.2: Temperature Sensor

3.1.3 Microcontroller

I decided to use the Arduino MKR 1010 microcontroller due to the fact that this specific Arduino has WIFI compatibility, making it significantly easier to send live data to the API for this first prototype. In addition, I have familiarity working with Arduinos and there is readily available documentation on interfacing between this board and the chosen sensors.

3.2 Circuit Diagram and Wiring

3.2.1 Circuit Diagram

Figure 3.3 shows the circuit diagram used for this project.

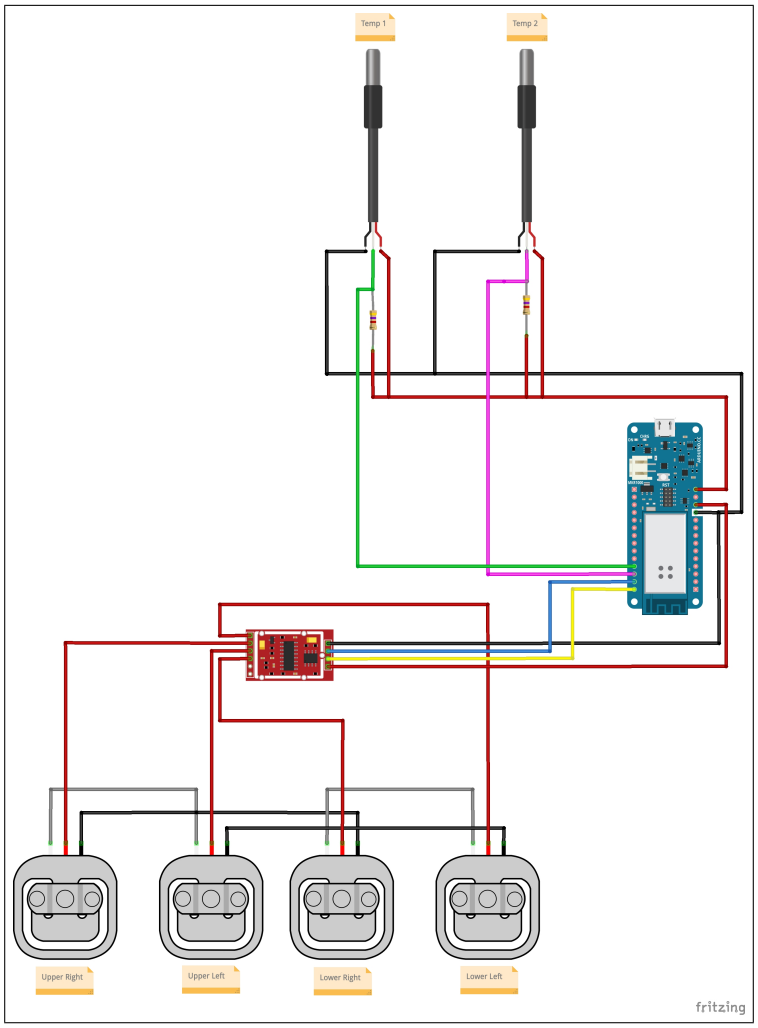


Figure 3.3: Circuit Diagram

The circuit was designed following two online tutorials that outlined how to interface with these specific sensors. Instructables.com has an online tutorial that uses these weight sensors with the Arduino Uno to create a digital scale which was a helpful guideline [6]. For the temperature sensor section, I also managed to find a helpful online tutorial to use as a guide on lastminuteengineers.com [7].

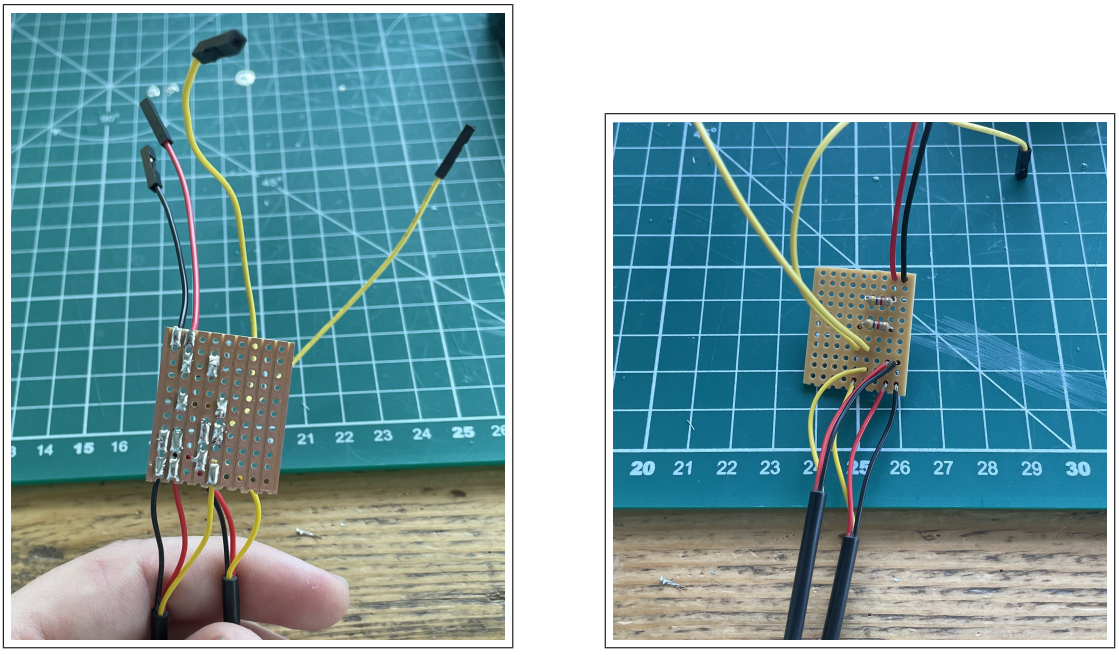
Since for this initial prototype I was not concerned with the power usage because the Arduino was going to be plugged into the wall outlet, so there was no need to add external power such as a battery or solar panels. The load cells all connect to the HX711 load cell amplifier which

is a specialised integrated circuit that interfaces with the load cells, converting their analog output into a digital signal. It also amplifies the small electrical signals generated by the load cells in response to the applied force or weight, allowing for precise weight measurement. The HX711 load cell amplifier then connects to the 4 and 5 digital pins as well as the 3.3V pin for power and the ground pin to complete the circuit.

Both temperature sensors were powered using the 5V pin. The first and second temperature sensors connect to the 2nd and 3rd digital pins respectively, each with a 4.7kΩ resistor used to limit current to the sensors.

3.2.2 Soldering

In order to simplify the circuit for the temperature sensors and ensure the reliability of this first prototype, I used a perfboard to solder the two temperature sensors and their resistors to a single board which could then be attached to the Arduino. Figure 3.4 shows how the perfboard was soldered together.



(a) Back (b) Front

Figure 3.4: The perfboard created for the temperature sensors

3.3 Software

During this section I will explain the main parts of the Arduino code and provide contexts to the methods used for this project.

3.3.1 Setup

```
62  void setup(void)
63  {
64    Serial.begin(9600);
65
66    // Set up on board LEDs
67    WiFiDrv::pinMode(25, OUTPUT); //define red pin
68    WiFiDrv::pinMode(26, OUTPUT); //define green pin
69    WiFiDrv::pinMode(27, OUTPUT); //define blue pin
70
71    // Connect to WiFi
72    setupWiFi();
73
74    // start temp sensors
75    tempSensor1.begin();
76    tempSensor2.begin();
77
78    //start weight sensor
79    scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
80    scale.set_scale(calibration_factor); //This value is obtained by using the SparkFun_HX711_Calibration sketch
81    scale.tare(); //Assuming there is no weight on the scale at start up, reset the scale to 0
82  }
```

Figure 3.5: Setup Method

During the Setup method, a connection to the WiFi was established first which can be seen in Figure 3.6. Then the temperature and weight sensors are started-up. It is important that the beehive is not placed on the device when it is first turned on so that the load sensors are able to establish a base level zero for no weight.

```
38  void setupWiFi()
39  {
40    // Set LED to red to indicate it has not connected to WIFI yet
41    WiFiDrv::analogWrite(25, 255);
42    WiFiDrv::analogWrite(26, 0);
43
44    WiFi.begin(ssid, password);
45
46    while (WiFi.status() != WL_CONNECTED)
47    {
48      delay(500);
49      Serial.print(".");
50    }
51
52    Serial.println("\n");
53    Serial.println("WiFi connected");
54    Serial.println("IP address: ");
55    Serial.println(WiFi.localIP());
56
57    // Set LED to green to indicate it has successfully connected to WIFI
58    WiFiDrv::analogWrite(25, 0);
59    WiFiDrv::analogWrite(26, 255);
60  }
```

Figure 3.6: WiFi setup Method

The WiFi setup method is a standard connection method that takes advantage of the

WiFiNINA, and *WiFiClient* libraries to establish a connection to the WiFi given the ssid and password provided in the secrets.h file.

3.3.2 Main Loop

```
84  void loop(void)
85  {
86    // get data from sensors
87    getTemperatureData();
88    getWeightData();
89
90    //print data to serial monitor for debugging
91    Serial.print("Temperature 1: ");
92    Serial.print(temperature1);
93    Serial.print("C");
94    Serial.print("\n");
95
96    Serial.print("Temperature 2: ");
97    Serial.print(temperature2);
98    Serial.print("C");
99    Serial.print("\n");
100
101   Serial.print("Weight: ");
102   Serial.print(currentWeight);
103   Serial.print(" kg");
104   Serial.print("\n");
105
106   sendSensorData();
107   //check that the Wifi is still connected
108   if (WiFi.status() != WL_CONNECTED) {
109     // set LED to RED to indicate that some intervention is needed
110     WiFiDrv::analogWrite(25, 255);
111     WiFiDrv::analogWrite(26, 0);
112   }
113
114   // we only need to run the loop every 5 min
115   delay(1000*60*5);
116 }
```

Figure 3.7: Main Loop

During the main loop data is first collected and saved from the temperature and weight sensors. These methods can be seen in Figure 3.8. Using the serial connection we print the values gathered from the sensors to the serial logger for general debugging purposes. After this is done the *sendSensorData* method is called. This function can be seen in Figure 3.9.

The next step is a quick check to ensure that the device is still connected to the WiFi. The onboard LED on the Arduino MKR 1010 will turn red if there is ever a disconnection from the WiFi to alert the user that something went wrong. Finally, a 5 minute delay is added to the end of the method so that the API is not overloaded and the data is only gathered from the sensors in 5 min intervals.

The method *getTemperatureData* shown in Figure 3.8 simply takes advantage of the *OneWire* and *DallasTemperature* libraries to call the *getTempCByIndex* function to get the temperature values in celsius.

```
118  void getTemperatureData() {
119    tempSensor1.requestTemperatures();
120    temperature1 = tempSensor1.getTempCByIndex(0);
121
122    tempSensor2.requestTemperatures();
123    temperature2 = tempSensor2.getTempCByIndex(0);
124  }
125
126  void getWeightData() {
127    currentWeight = 0.454 * scale.get_units(); // convert lbs to kgs
128  }
```

Figure 3.8: Functions used to get sensor data

The *getWeightData* method takes advantage of the *HX711* library to call the *get_units* method on the scale to retrieve the weight in lbs which is then converted into kgs by multiplying the value by 0.454.

Finally, in order for this data to be saved and visualised on the dashboard, the saved sensor data needs to be sent to the ThingSpeak API. ThingSpeak is a powerful IoT analytics platform built by MathWorks. ThingSpeak provides a way for developers to aggregate, visualise and analyse the live data collected from their devices on their cloud servers. I chose to use ThingSpeak for this project as it made it significantly easier for me to communicate the data to the client and it is free to use for smaller non-commercial products. However, requiring users to create a ThingSpeak account and replace the API key with their own is not ideal. I felt that the advantage of rapid development of this initial prototype made up for this inconvenience.

```
130  void sendSensorData() {
131    // send data to Matlab API
132    String url = String("/update?api_key=") + apiKey +
133                 "&field1=" + String(temperature1, 3) +
134                 "&field2=" + String(temperature2, 3) +
135                 "&field3=" + String(currentWeight, 3);
136    client.get(url);
137
138    // read the status code and body of the response
139    int statusCode = client.responseStatusCode();
140    String response = client.responseBody();
141
142    Serial.print("Status code: ");
143    Serial.println(statusCode);
144    Serial.print("Response: ");
145    Serial.println(response);
146
147    // if api status code is not good then flash red to indicate data was not sent
148    if (statusCode != 200) {
149      WiFiDrv::analogWrite(25, 255);
150      WiFiDrv::analogWrite(26, 0);
151      delay(1000);
152      WiFiDrv::analogWrite(25, 0);
153      WiFiDrv::analogWrite(26, 255);
154    }
155  }
```

Figure 3.9: Function used to send sensor data to ThingSpeak API

Figure 3.9 shows the sendSensorData function which is used to send the data to the ThingSpeak API. First the URL is constructed using the APIKey (which is stored in the *secrets.h* file)

and the stored values from the sensors. This URL is then used with the *WiFiClient* library to send an HTTP request to the ThingSpeak API which updates the values for each of the fields. After printing the status code and response to the serial logger for debugging purposes, the response code is checked and confirmed successful. If there is a response code other than 200, the Arduino on-board LED flashes red indicating the missed API request.

3.3.3 Dashboard

In order to communicate the data gathered from the device to the beekeeper I created the BeeSmart Dashboard. The BeeSmart Dashboard is a website that has two main utilities. The first and main purpose of the dashboard is to get the data from the ThingSpeak API and visualise it in clear graphs for the user. I used the ThingSpeak API to get the data from the beehive and used Chart.js to plot the data points on a line graph.

The second use for the website was to provide a map for the user to allow them to place a pin on the location of their beehive and get real-time weather information from the surrounding area. To do this I used a combination of the google maps API to show the map on the page and get the coordinates of the pin and the Open Weather API which I used to make a request for the temperature and a description of the weather in the given location.

Lastly, I included an about page to give context to the project as a whole. The website was hosted using GitHub pages. The following figures show screenshots taken from the BeeSmart Dashboard.

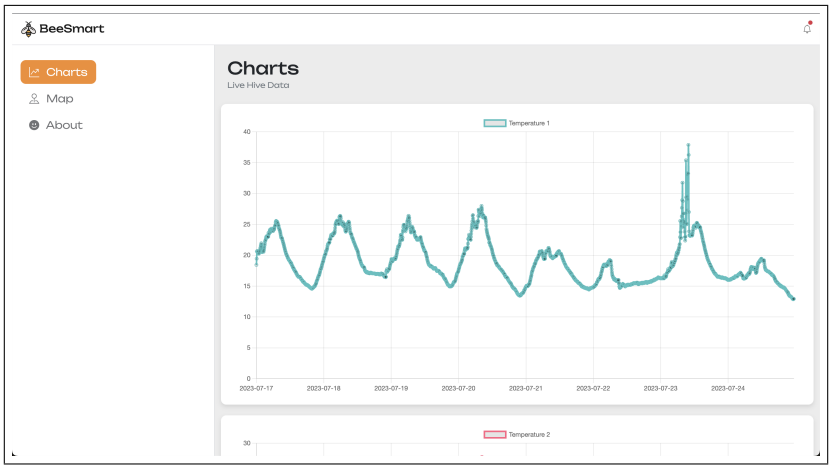


Figure 3.10: The main page on the BeeSmart dashboard

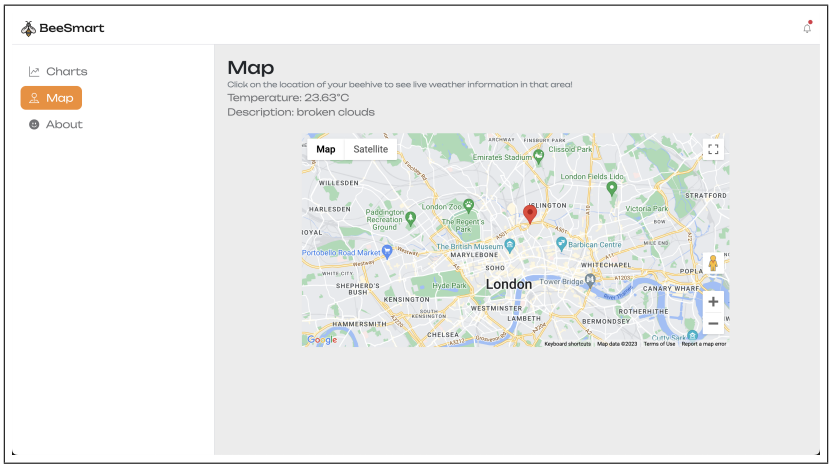


Figure 3.11: The map page on the BeeSmart dashboard

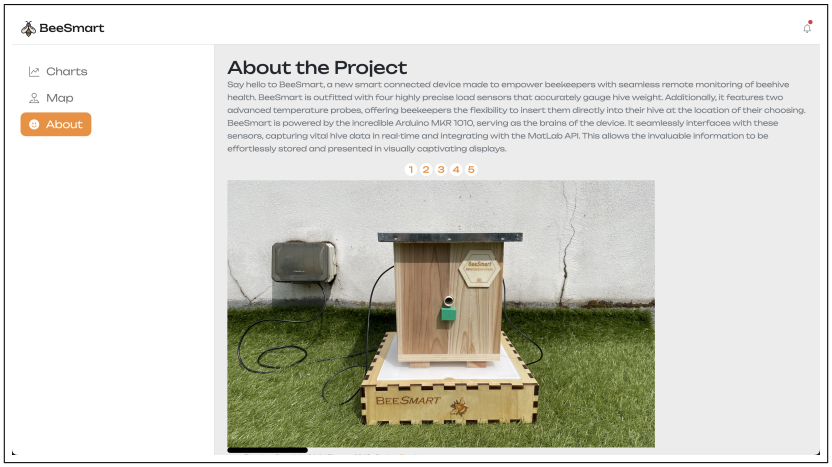


Figure 3.12: The about page on the BeeSmart dashboard

3.4 Enclosure

When creating the enclosure for the device, I determined the most effective way for me to ensure that I could follow my designs most accurately, was to use a laser cutter. The laser cutter allowed me to precisely cut the wood used for the enclosure to fit perfectly together to make the box design. Figure 3.13 shows the laser cutter and the design cut into the wood that was used to create the enclosure.

Using the laser cutter enabled me to create precise holes in the centre of three of the sides of the box enclosure to allow the power cord to enter the enclosure and the temperature sensors to hang out of the enclosure while keeping the sensitive hardware safe inside.

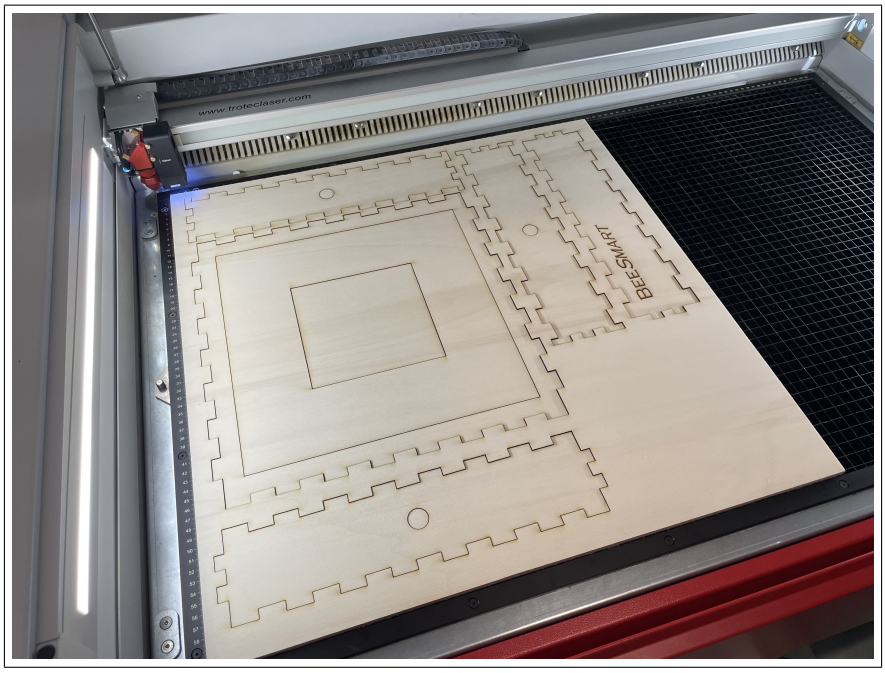


Figure 3.13: The laser cutter cutting out the enclosure

3.4.1 Waterproofing

Since the device is intended to be deployed in an outside environment that will be subject to regular weather conditions, I made some further modifications to the enclosure to ensure that the inside of the enclosure would remain dry, keeping the sensitive hardware safe. A rubber lining was used on the inside of the box over the holes made for the wires to limit potential rain from entering the enclosure shown in Figure 3.14.

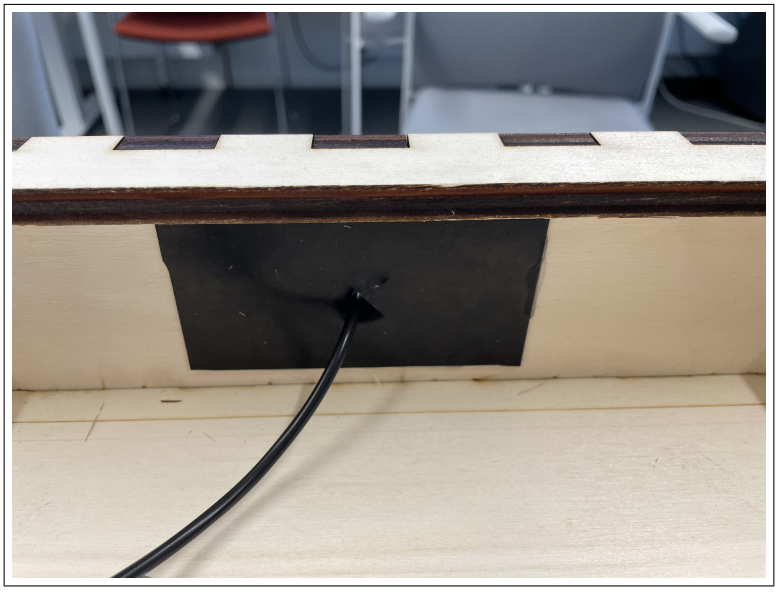


Figure 3.14: The rubber lining added to the inside of the enclosure

Additionally, the entire box was coated with a yacht varnish used to ensure moisture does not penetrate through the wood of the box. Figure 3.15 shows the enclosure ready to be varnished.

Figure 3.15: The enclosure ready to be applied with yacht varnish

3.4.2 Weight Plate

In order for the load sensors to work properly several modifications were made to ensure their regular operation. Firstly, the laser cutter was used to cut a weight plate out of acrylic. This weight plate was placed directly on top of the load sensors and where the beehive was placed. Acrylic was used as it is perfectly flat ensuring that each of the load sensors has equal contact to the weight plate. Secondly, an internal platform was made to raise the height of the load sensors so that they would be directly below the top of the enclosure making contact with the weight plate.

Finally, a 3D printed piece was made to sit directly below each load sensor. This piece was used so that the internal structure of the load sensor had the freedom to move which is how it is able to measure the load placed on it. The STL for this 3D printed part was created by the user patrick3345 and is available on Thingiverse [9]. Figure 3.16 shows the piece on the inner platform. To reduce the vertical height of the 3D printed piece I only used the inner section of the piece and super glued it to the sensors so they would not move around.

Figure 3.16: 3D printed piece

3.4.3 Putting it all together

With all the separate components needed for the device completed, the device could be assembled. Figure 3.17 shows the devices internals with all the hardware inside.

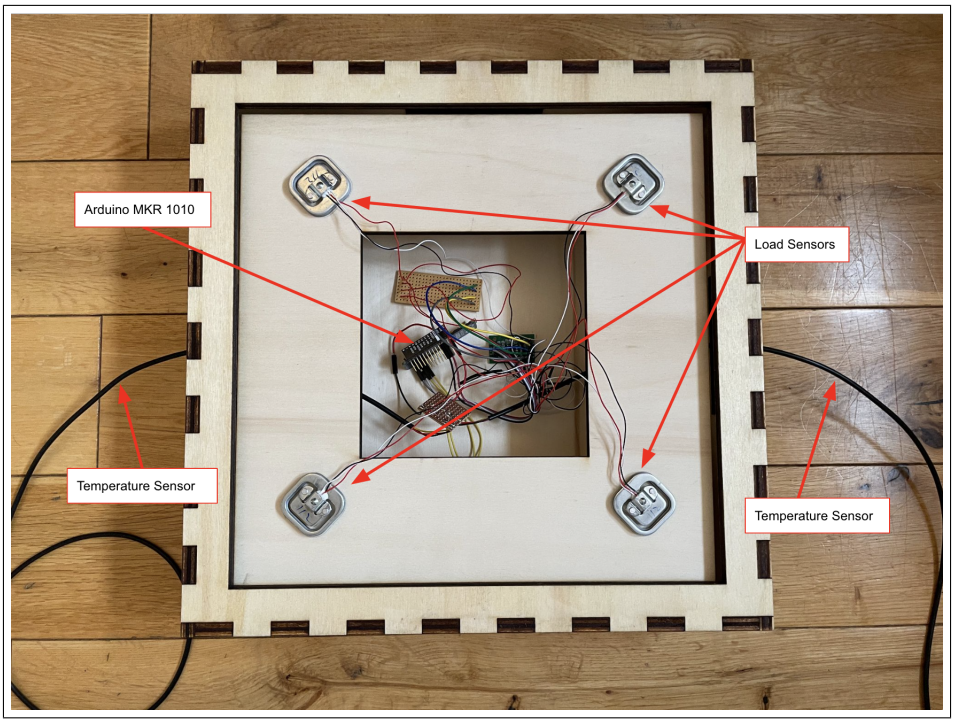


Figure 3.17: A labeled picture of internals of the device

Chapter 4

Experimental Setup and Methodology

This section outlines the experimental design and methodology employed to test the functionality and performance of the BeeSmart device. The primary objective of this experiment was to evaluate the device's effectiveness in a simulated environment resembling real-world beekeeping scenarios. The experiment was conducted over a week, during which data was collected at regular intervals to assess the device's capabilities in monitoring and managing beehive conditions.

Initially the plan was to find an urban beekeeper within London to test the device. However, it was found to be too difficult to coordinate with beekeepers willing to subject their beehives to experimentation, so it was decided to replicate real-world conditions, by using a smaller beehive with no honey bees for this experimentation. Although not made for honey bees, the beehive is made for solitary bees. These types of bees only produce small amounts of honey and use the beehive to form nests and lay eggs.

The device and beehive was positioned outdoors on the roof of my house, exposing it to environmental factors such as temperature fluctuations, sunlight exposure, and ambient conditions. This placement enabled the simulation of the device's performance in an external setting, akin to its practical application as well as allowing me to connect to my own WiFi network and give me the ability to check up on it if anything went wrong during the experiment. Figure 4.1 shows the device fully deployed on my roof with the beehive placed on top of it.

Figure 4.1: The deployed device

The two temperature probes were placed within the beehive, each serving a distinct purpose. The first temperature probe was positioned at the top interior of the beehive, emulating a scenario where the device measures the temperature at the beehive's surface. The second temperature probe was situated at the bottom of the beehive beneath insulation, mimicking a scenario where the probe is embedded within the beehive structure. This configuration allowed for the evaluation of the device's ability to monitor temperature variances both at the beehive's exterior and within its core, a critical aspect in assessing the overall health of the colony. Figure 4.2 shows the placement of the two temperature probes within the beehive.

Figure 4.2: The interiror of the beehive

To replicate the gradual accumulation of honey over time, 32 glass beads were placed into the beehive each day, with each glass bead weighing approximately 4.5 grams. This test aimed to simulate the incremental buildup of honey within a beehive and provided a way to measure the device's responsiveness to changes in weight over time. This weight simulation was instrumental in assessing the device's accuracy in detecting changes in beehive conditions, as honey production is a fundamental aspect of beekeeping management. Figure [hive-interior] shows these glass beads at the bottom of the beehive.

The experiment was conducted over a span of one week. This duration was deemed sufficient to capture a range of environmental conditions, enabling a comprehensive assessment of the device's performance and reliability.

In summary, the experiment sought to closely emulate the real-world use of this device and provide a valuable understanding of this first prototype's strengths and weaknesses.

Chapter 5

Evaluation

5.1 Temperature

The evaluation of the data gathered from the BeeSmart device over the course of the experiment provides valuable information on the effectiveness of the device. The following figures show the data gathered from each of the sensors from the device over the course of the experiment.

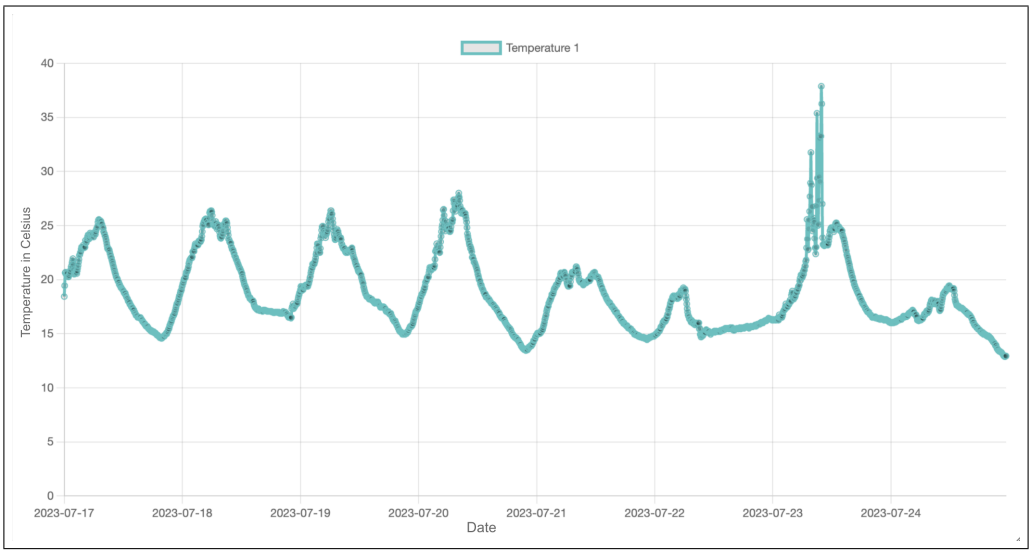


Figure 5.1: The graph for temperature sensor 1

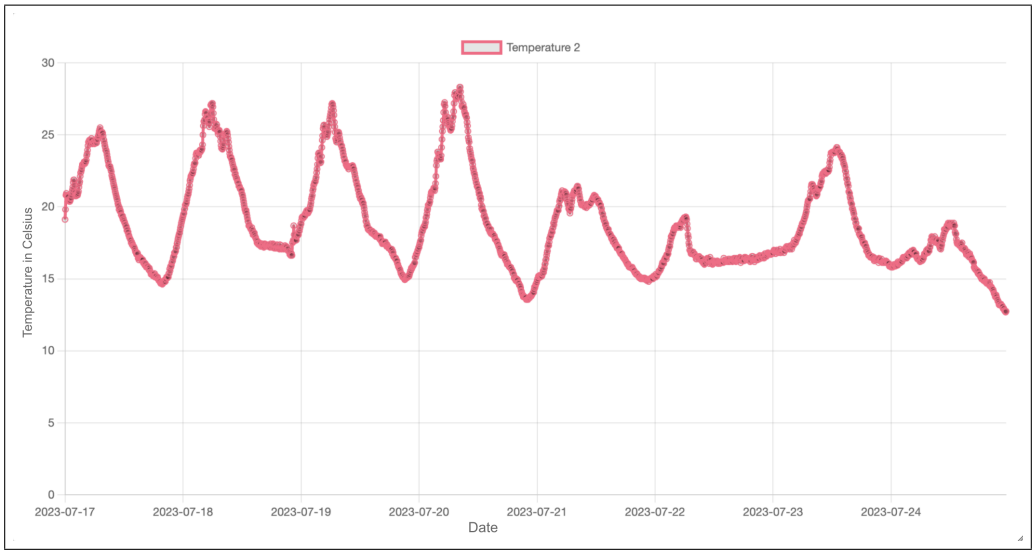


Figure 5.2: The graph for temperature sensor 2

The figures 5.1 and 5.2 show how the temperature sensors were able to accurately measure the regular changes in temperature from day to night each day. Temperature sensor 1 had a few anomalous data points towards the last day where strong winds knocked the sensor out of the beehive. Once I noticed these strange values appearing I quickly checked up on the device and placed the sensor back within the beehive at which point the values returned back to normal. It is interesting to note that despite one of the sensors (temperature sensor 2) being insulated the temperature values between the two sensors are very similar to each other. However, I do believe given an active beehive we would have seen a bigger difference between the centre of the beehive and the extremities as the bees themselves produce a noticeable heat signature.

5.2 Weight

The weight graph illustrates a very interesting story. The very first value captured was the initial setup weight value of zero used to calibrate the device. The graph from the 17th to the end of the 22nd shows the device working phenomenally well where subtle variations in the weight can be seen but the overall trend upwards is clearly shown. I believe these changes in the weight during the day when we would expect the graph to remain flat can be explained by the load sensors being affected by the changes in temperature.

Given the design of the load sensors higher temperatures would allow the inner section of the load sensors to move more, resulting in higher weight values and inversely colder temperatures would show a lighter weight. This theory is corroborated when comparing the weight graph to

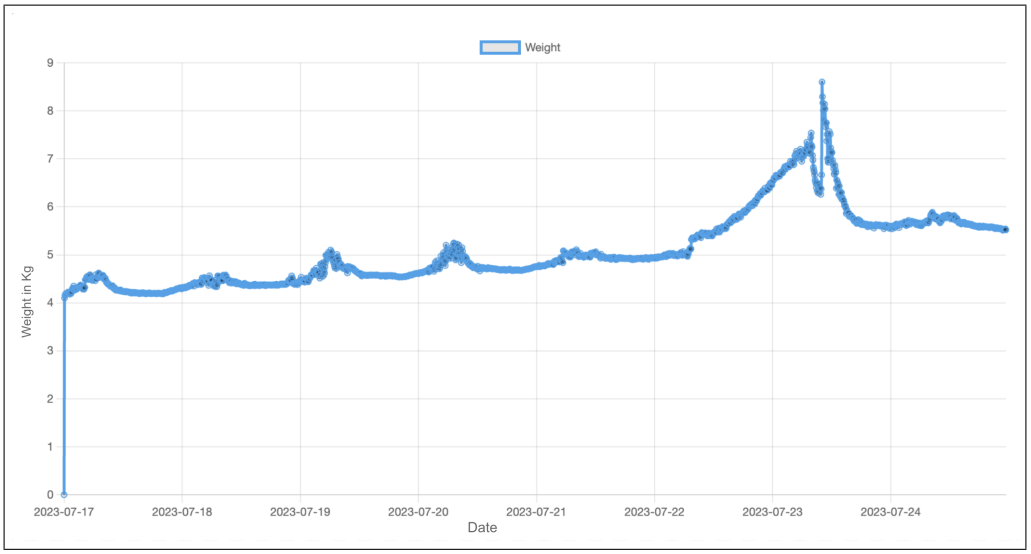


Figure 5.3: The graph for beehive weight

the temperature graphs. The weight 'sag' seems to occur over night when the temperatures are lowest and then there are periods where the weights are sporadic and these seem to correspond with the hottest times of the day. It is worth noting that these temperature fluctuations are only around a few 10s of grams and for the purpose of monitoring overall beehive health in a real world scenario, it is negligible.

However, there is a large anomaly in the weight value over the 23rd. The night before (22nd), I had noticed that the weather forecast for the 23rd was expected to be heavy rain. In preparation for these rains, I decided to place a waterproof protective covering over the top of the device to further protect it from the heavy rains in an attempt to ensure that no water would seep into the enclosure destroying the sensitive hardware inside. Figure 5.4 shows a picture of the device taken the day after the rain with this protective covering on it.

Figure 5.4: The device with the additional waterproof covering

As you can see from the picture above, a substantial amount of rain collected on top of the beehive and the protective covering. I believe this additional weight from the rain water is the reason why the weight sensor shows this massive increase in weight shown on the 23rd of July. As the graph shows, once the rain evaporated over the next day the weight values returned to their expected values. Even though the weight values during this period were unreliable it is notable that the device still had no problem sending the data to the API despite the heavy rains.

Chapter 6

Conclusion

6.1 Future Work

After completing this first prototype, I have identified several areas I feel could be improved in future iterations of the BeeSmart device. In order to make the device more durable in outdoor environments I would suggest adding additional rubber lining to the top of the enclosure around the weight plate to prevent rain from entering from the top of the device. I would also recommend adding insulation to the inside of the enclosure to help regulate the temperatures around the load sensors and keep the components in place.

In future generations of the device I would also like to include new sensors such as a microphone and camera. As Mr. Saville mentioned in the interview, being able to be alerted of potential threats to the beehive would be very helpful to the urban beekeepers. Although adding these sensors did not fit into the scope of this project, they would serve as a great future addition to the next generations of the BeeSmart device.

Finally, given more time I would like to move away from relying on the ThingSpeak API and instead develop the backend infrastructure to store data on the site itself. My long term idea for the BeeSmart dashboard would be to serve as a social hub where urban beekeepers could share their results with each other and researchers alike so they could gain a deeper understanding of the collective health of beehives within a community and work collaboratively. The map could serve as a way of measuring the number and density of urban colonies within a city, a task that is currently very hard for researchers to estimate.

6.2 Conclusion

To form a conclusion, I want to go over what this project set out to achieve. The goal was to create a device intended for urban beekeepers to automate the remote monitoring of beehive health. I wanted this device to be affordable, open sourced and something that a hobbyist could follow along and build themselves if they wanted to. This way more urban beekeepers would have access to real-time sensor data from their beehives. To meet this end, the BeeSmart device was built to sense the weight and temperature of a given beehive and the dashboard was made to communicate this data to the beekeepers. During the week the device was tested it showed it was capable of providing reliable real-time data. Although there were a few outliers in the data, these outliers were only off from their expected values by a small amount and they seemed to not impact the device's ability to properly provide accurate data over time.

Ideally, I would have liked to test this device over a much longer set of time. Allowing the test to be conducted over several months would have provided valuable insight on how well the load sensors would hold up to prolonged use as well as testing how reliable they are during the hottest days of summer and coldest days of winter. Given more time, I would have liked to have found a real urban beehive to test the device on. Even though the test had the temperature sensors in a very similar setting to their intended use case, I would have liked to know if the accumulation of beeswax around the sensor or the constant commotion of the colony would have affected the sensors in a way I could not foresee.

In conclusion, the first prototype for the BeeSmart device has its limits but it was a successful first prototype for this project and it shows good promise for future generations of the device especially with open sourced collaboration.

References

[1] The London Beekeepers Association. *The London Bee Situation*, May 2020. URL: https://drive.google.com/file/d/1-Y3XCP50RCDy_9jq_oCB8VI6FOQtUns9/view.

[2] Perfect Bee. *Why is weighing your beehive important?*, 2023. URL: https://www.perfectbee.com/blog/weighing-your-beehive-why-and-how#:~:text=A%20medium%20super%20will%20weigh,should%20weigh%20about%2080%20lbs.

[3] BeeScales. Beehive scale for monitoring two beehives, 2023. URL: https://www.beescales.io/en.

[4] Beewise. Beewise, 2019. URL: https://www.beewise.ag/.

[5] Morgan M. Cunningham, Lan Tran, Chloe G. McKee, Rodrigo Ortega Polo, Tara Newman, Lance Lansing, Jonathan S. Griffiths, Guillaume J. Bilodeau, Michael Rott, and M. Marta Guarna. Honey bees as biomonitors of environmental contaminants, pathogens, and climate change. *Ecological Indicators*, 134:108457, 2022.

[6] DegrawSt. Arduino bathroom scale with 50 kg load cells and hx711 amplifier, Oct 2020. URL: https://www.instructables.com/Arduino-Bathroom-Scale-With-50-Kg-Load-Cells-and-H/.

[7] Last Minute Engineers. Interfacing ds18b20 1-wire digital temperature sensor with arduino, Oct 2022. URL: https://lastminuteengineers.com/ds18b20-arduino-tutorial/.

[8] Beehive Monitoring. Hive heart 3.0, internal hive monitoring, 2023. URL: https://www.beehivemonitoring.com/en/home/1-hive-heart-30-internal-hive-monitoring-8588007887029.html.

[9] patrick3345. 50kg loadcell bracket versionf, Nov 2017. URL: https://www.thingiverse.com/thing:2624188.

[10] Charlotte Varela. Why are bees important? - the woodland trust - woodland trust, Apr 2023. URL: https://www.woodlandtrust.org.uk/blog/2023/04/why-are-bees-important/.

[11] Adam Voiland and Wayne Esaias. *Honey Bees Turned Data Collectors Help Scientists Understand Climate Change*, Aug 2009. URL: https://www.nasa.gov/topics/earth/features/beekeepers.html.

Appendix A

GitHub Links

A.1 Link to the GitHub Repository

https://github.com/wezpez/bee-smart

A.2 Link to the BeeSmart Dashboard

https://wezpez.github.io/bee-smart/

Appendix B

Supervisor Meetings

I had a meeting with my supervisor, Duncan Wilson, on 06/18/2023. During this meeting we discussed the following topics.

1. Discussed a timeline for how to complete the project given the remaining time.
2. Which sensors I should focus my attention on.
3. Whether access to real bees was a necessary requirement for this project.
4. How I could conduct an experiment using a beehive not containing any bees.

I had a meeting with my supervisor, Duncan Wilson, on 07/13/2023. During this meeting we discussed the following topics.

1. How the device I built was going to work.
2. Problems I encountered with the load sensors.
3. What needed to be included in my report.

Appendix C

Source Code

C.1 Arduino Code

This file contains the code uploaded to the Arduino MKR 1010.

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include "HX711.h"
#include <WiFiNINA.h>
#include <WiFiClient.h>
#include <ArduinoHttpClient.h>
#include <utility/wifi_drv.h>
#include "secrets.h"

// API host
const char* host = "api.thingspeak.com";

// define temperature sensor pins
#define TEMP1_PIN 2
#define TEMP2_PIN 3

// define weight sensor pins and calibration factor
#define LOADCELL_DOUT_PIN 4
#define LOADCELL_SCK_PIN 5
```

```
#define calibration_factor -7050.0 //This value is obtained using the SparkFun_HX711_

// Setup oneWire and DallasTemperature instances for each temperature sensor
OneWire oneWire1(TEMP1_PIN);
DallasTemperature tempSensor1(&oneWire1);

OneWire oneWire2(TEMP2_PIN);
DallasTemperature tempSensor2(&oneWire2);

// global variables
float temperature1 = 0;
float temperature2 = 0;
float currentWeight = 0;
HX711 scale;
// wifi variables
WiFiClient wifi;
HttpClient client = HttpClient(wifi, host, 80);

void setupWiFi()
{
  // Set LED to red to indicate it has not connected to WIFI yet
  WiFiDrv::analogWrite(25, 255);
  WiFiDrv::analogWrite(26, 0);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\n");
```

```
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  // Set LED to green to indicate it has successfully connected to WIFI
  WiFiDrv::analogWrite(25, 0);
  WiFiDrv::analogWrite(26, 255);
}

void setup(void)
{
  Serial.begin(9600);

  // Set up on board LEDs
  WiFiDrv::pinMode(25, OUTPUT); //define red pin
  WiFiDrv::pinMode(26, OUTPUT); //define green pin
  WiFiDrv::pinMode(27, OUTPUT); //define blue pin

  // Connect to WiFi
  setupWiFi();

  // start temp sensors
  tempSensor1.begin();
  tempSensor2.begin();

  //start weight sensor
  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
  scale.set_scale(calibration_factor); //This value is obtained by using the SparkFun_
  scale.tare(); //Assuming there is no weight on the scale at start up, reset the scal
}

void loop(void)
{
```

```
  // get data from sensors
  getTemperatureData();
  getWeightData();

  //print data to serial monitor for debugging
  Serial.print("Temperature 1: ");
  Serial.print(temperature1);
  Serial.print("C");
  Serial.print("\n");

  Serial.print("Temperature 2: ");
  Serial.print(temperature2);
  Serial.print("C");
  Serial.print("\n");

  Serial.print("Weight: ");
  Serial.print(currentWeight);
  Serial.print(" kg");
  Serial.print("\n");

  sendSensorData();
  //check that the Wifi is still connected
  if (WiFi.status() != WL_CONNECTED) {
    // set LED to RED to indicate that some intervention is needed
    WiFiDrv::analogWrite(25, 255);
    WiFiDrv::analogWrite(26, 0);
  }

  // we only need to run the loop every 5 min
  delay(1000*60*5);
}

void getTemperatureData() {
```

```
  tempSensor1.requestTemperatures();
  temperature1 = tempSensor1.getTempCByIndex(0);

  tempSensor2.requestTemperatures();
  temperature2 = tempSensor2.getTempCByIndex(0);
}

void getWeightData() {
  currentWeight = 0.454 * scale.get_units(); // convert lbs to kgs
}

void sendSensorData() {
  // send data to Matlab API
  String url = String("/update?api_key=") + apiKey +
                    "&field1=" + String(temperature1, 3) +
                    "&field2=" + String(temperature2, 3) +
                    "&field3=" + String(currentWeight, 3);
  client.get(url);

  // read the status code and body of the response
  int statusCode = client.responseStatusCode();
  String response = client.responseBody();

  Serial.print("Status code: ");
  Serial.println(statusCode);
  Serial.print("Response: ");
  Serial.println(response);

  // if api status code is not good then flash red to indicate data was not sent
  if (statusCode != 200) {
    WiFiDrv::analogWrite(25, 255);
    WiFiDrv::analogWrite(26, 0);
    delay(1000);
```

```
      WiFiDrv::analogWrite(25, 0);
      WiFiDrv::analogWrite(26, 255);
    }
}
```