Perch-Scale: An Exploration into Data Capturing Systems for Small Wild Bird Research

Prepared by:
Tristyn Ferreiro (FRRTRI001)
Mishay Naidoo (NDXMIS011
Alex Van Rijswijck (VRJALE001)
Meg Wilson (WLSMEG005)

Prepared for:
EEE4113F
Department of Electrical Engineering
University of Cape Town

August 13, 2023

Declaration

1. We know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. We have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is our own work.
4. We have not allowed, and will not allow, anyone to copy our work with the intention of passing it off as their own work or part thereof.

August 13, 2023

Tristyn Ferreiro — Date

August 13, 2023

Mishay Naidoo — Date

August 13, 2023

Alex Van Rijswijck — Date

August 13, 2023

Meg Wilson — Date

Contents

1 Introduction **1**

1.1 Background . . . 1
1.2 Problem Statement . . . 2
1.3 Scope & Limitations . . . 2
1.4 Report Outline . . . 2

2 Literature Review **3**

2.1 Current Weighing Techniques . . . 3
2.1.1 Manual Data Capture . . . 3
2.1.2 Smart Scales and Nests . . . 4
2.2 Sensing Techniques for Bird Identification . . . 5
2.3 Camera Capturing Techniques . . . 5
2.3.1 Camera traps . . . 5
2.4 How to bait birds . . . 6
2.5 Power Supply . . . 6
2.6 Data Processing . . . 7
2.6.1 Machine Learning for Filtering . . . 7
2.6.2 Compression . . . 8
2.7 Wireless Forms of Data Transfer . . . 8
2.7.1 Bluetooth Low Energy . . . 8
2.7.2 ZigBee . . . 9
2.8 Literature Critique . . . 9

3 Scale and Housing Design **10**

3.1 Introduction . . . 10
3.2 Design Choices . . . 10
3.2.1 The Weighing Mechanism . . . 11
3.2.2 The Perch . . . 12
3.2.3 The Housing . . . 12
3.2.4 Specifications . . . 13
3.3 Prototype Design . . . 13
3.3.1 Designing the Test Stand and Perch . . . 14
3.3.2 Load Cell Code . . . 15
3.3.3 Designing the Housing . . . 16
3.4 Testing and Results . . . 17
3.4.1 Acceptance Tests . . . 17
3.4.2 The Test Stand and Perch . . . 17

3.4.3 Load Cell Code Results . . . 17
3.4.4 The Housing . . . 19
3.4.5 Bill of Materials . . . 20
3.5 Conclusion . . . 20

4 Camera Design **21**
4.1 Introduction . . . 21
4.2 Requirements and specifications . . . 22
4.3 ESP32-cam hardware specifications . . . 22
4.4 Design choices . . . 23
4.5 Design Process . . . 23
4.5.1 Programming the ESP32 Cam . . . 23
4.5.2 System overview . . . 23
4.5.3 Initialization . . . 24
4.5.4 Capture image . . . 25
4.5.5 Image processing . . . 25
4.6 Testing and Results . . . 26
4.6.1 Cropping . . . 26
4.6.2 Colour thresholding . . . 28
4.6.3 Acceptance Tests . . . 29
4.7 Conclusion . . . 30

5 Power Module **31**
5.1 Introduction . . . 31
5.2 Design Choices and Process . . . 31
5.2.1 Requirements and Specifications . . . 31
5.2.2 Power connections to the ESP32 Development board . . . 31
5.2.3 Power Sources to ESP32 Development Board . . . 32
5.2.4 Solar Power Management Board . . . 34
5.3 Prototype Design . . . 35
5.3.1 First Design . . . 35
5.3.2 Final Design . . . 36
5.4 Testing and Results . . . 36
5.4.1 Acceptance Tests . . . 37
5.4.2 Results . . . 38
5.4.3 Conclusion . . . 40

6 Data Retrieval Design **41**
6.1 Introduction . . . 41
6.2 Design Choices . . . 42
6.2.1 Data Storage . . . 42
6.2.2 Facilitating Data Acquisition . . . 42
6.2.3 Retrieving Data from the device . . . 44
6.3 Prototype Design . . . 45

6.3.1 Sub-module Integration . . . 45
6.3.2 Data Storage . . . 48
6.3.3 Website . . . 48
6.3.4 Final System Flow . . . 49
6.4 Testing and Results . . . 49
6.4.1 Acceptance Tests (ATs) . . . 49
6.4.2 Acceptance Test Procedures (ATPs) . . . 49
6.4.3 Results . . . 51
6.5 Conclusion . . . 52

7 Conclusions and Recommendations **53**
7.1 Recommendations . . . 54
7.1.1 Scale . . . 54
7.1.2 Camera . . . 54
7.1.3 Power . . . 54
7.1.4 Data . . . 54

Bibliography **55**

Chapter 1

Introduction

How many cans do you need to make a bird?

— *Two cans*

1.1 Background

This project focuses on research being done on the Southern Yellow-Billed Hornbills in the Kalahari. The research is being done by Benjamin Murphy, an Ornithologist at the Fitzpatrick Institute of Ornithology. The Hornbill population is being threatened by climate change and any data that researchers like Ben can obtain on the birds is useful.

Hornbills mate in pairs and nest in tree hollows. When the female Hornbills nest, they enclose themselves in the tree. The birds are hard to research in these hollows, prompting researchers to use artificial nestboxes to re-home the Hornbills so that they can be studied easily. The nest boxes have lids on the roof that can be opened to measure the chicks and are fitted with temperature sensors to monitor the heat inside the boxes. The boxes are designed to dissipate heat with the goal of reducing heat stress on the female Hornbills and chicks. The nest boxes look like this:

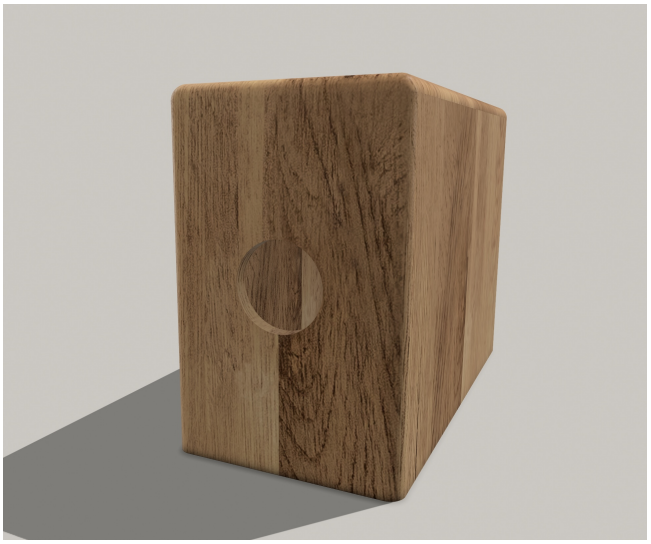


Figure 1.1: 3D Render of Nest Boxes in the Kalahari

Each Hornbill pair nests in one box. Researchers go to the box every second week to measure egg sizes and chick wingspans. Currently, they are unable to obtain weight data on the Hornbills. The male Hornbill will visit the nest box to either feed the female while she is nesting, or to enter the nest.

1.2 Problem Statement

The overarching problem is that researchers are unable to obtain any weight data on the Southern Yellow-Billed Hornbills. Furthermore, when obtaining this data it is important to identify which bird is being weighed. The research is conducted in a remote location with limited access to infrastructure. The harsh conditions of the Kalahari cause things to break easily and there is limited funding.

1.3 Scope & Limitations

This project has the following scope:

- The research deals with the Southern Yellow-billed Hornbills.
- The goal of this project is to obtain weight information on the birds.
- The birds are being researched in the Kalahari, which has harsh outdoor conditions. The Kalahari is remote meaning the weight capturing device must function remotely.

This project has the following limitations:

- The harsh environmental conditions pose a risk of frequent component failures, necessitating affordable and easily replaceable parts.
- The final prototype has to be completed within 6 weeks.
- Design and development of the prototype will take place in Cape Town, where there is no access to the nest boxes or the Hornbills for testing purposes.
- It is crucial that the design ensures non-intrusive data collection, prioritizing the well-being and safety of the birds.

1.4 Report Outline

The report starts with a summary of the relevant literature surrounding the problem statement and possible solutions in chapter 2. The scale and housing design, along with a detailed examination of weight data processing methods, are presented in chapter 3. Thereafter the camera design and image processing are discussed in chapter 4. In chapter 5 the power system is discussed in the context of this remote application. In chapter 6 considers and implements a data retrieval solution. Finally, in chapter 7, conclusions are drawn and recommendations for improvements are made.

All design files can be found at the project GitHub.

Chapter 2

Literature Review

What do you get when you kiss a diseased bird?

— *Cherpies*

This literature review explores previous research in order to understand approaches to bird wildlife monitoring. This includes research into current animal weighing and camera capturing techniques. Methods for baiting birds are also investigated. Efficient data processing, memory usage and wireless communication techniques are explored with a special focus on applications in systems with limited resource availability.

2.1 Current Weighing Techniques

This section details a review of existing literature on the weight capturing techniques used on birds in the wild. There is a large amount of research in this field due to the importance of body weight data for both field and laboratory studies [1]. Both manual and smart weight data capturing techniques are covered.

2.1.1 Manual Data Capture

Historically, people have often obtained weight data on birds by capturing them in traps or nets. However, this method makes it difficult to obtain multiple readings on an individual bird [2] and can stress the bird. In his 1989 paper, Svein Haftorn [2] describes needing to capture weight measurements for wild titmice at specific times in the day and found that capturing the birds multiple times a day to be problematic due to the reasons mentioned above.

Haftorn thus, opted to place a battery powered electronic scale (specifically a Sartorius model) outside an observer hut that they used for watching the birds. This scale was enclosed in a simple box with one side open. The birds were baited to the scale (explored in 2.4) and Haftorn was able to read the scale from inside the hut.

They were able to collect the necessary data but experienced two problems: having to replenish the bait regularly and needing to sit and watch the scale to record data readings.

Benjamin Murphy, an ornithologist studying Fork-Tailed Drongos at the Fitzpatrick Institute of Ornithology, was interviewed and described using a similar method to capture weight measurements as the one used by Haftorn. Benjamin uses a perch attached to a kitchen scale to obtain his data. The

bird is baited to the perch using a worm. The weight measurement is then read off of the scale using binoculars from far away, so as not to scare the bird.

2.1.2 Smart Scales and Nests

With the progress made in machine learning and sensing over the last 20 years, smart weight data capture has seen an increase in popularity. In their paper on automated weight recording, Boisvert and Sherry [1] detail how weight and feeding data can be obtained from passive integrated transponder (PIT) tags and smart nests. Their smart nest contained a food dispenser to attract the birds, a PIT scanner and reader, and an electronic scale to capture the bird's weight.

Once the bird was attracted to the nest by the feeder, the PIT scanner would trigger the weight recording mechanism and identify the bird. The weighing mechanism would send measurements, accurate to 0.01g, [1] every 0.25s for a duration of 3s. When no bird was sensed on the perch the scale transmitted a weight reading every 5 minutes. These readings were used to dynamically generate the tare reading (the weight recorded by the scale when no bird was on it). The tare reading was used to zero the scale and prevented debris build up on the scale from affecting the measurements. Boisvert and Sherry found that taking the maximum weight recorded each time the bird landed on the perch was the most accurate measurement. However, they admit that for other scale layouts and heavier birds, a more complex algorithm for estimating the weight of the bird may be necessary.

The above system was successful in recording several hundred feeder visits each day and allowed them to detect daily fluctuations in the birds' weight. This further allowed them to predict eating patterns in the birds, allowing the researchers to understand whether the birds were eating the seeds or storing them elsewhere. This type of data analysis was previously inconceivable when using manual techniques.

In a more recent paper released in 2013, Larios et al. [3] detail their project on a smart scale system designed to capture weight data on lesser kestrels without stressing the animal. They designed the system to integrate with artificial nest boxes. This allowed them to place the scale in an area of the nest that the animal had to pass through often, in most cases the entry point to the nest.

An important distinction between stable and unstable weight measurements was made in their paper [3]. A stable measurement was defined as one in which the bird was standing still when the readings were taken. An unstable reading was one in which the bird was moving on the scale, resulting in a slightly inaccurate measurement.

Unlike Boisvert and Sherry [1], Larios et al. [3] found that their system was unable to capture stable weight measurements. This is because their measurements were taken when the bird was moving whilst Boisvert recorded the data when the bird was standing still. To account for the unstable measurements, Larios et al. used a machine learning algorithm (discussed in 2.6.1) to predict the birds' weight using both stable and unstable measurements.

Their [3] design also allowed researchers to observe the size of prey the birds were capturing. This was done by measuring fluctuations in the birds' weight throughout the day.

Like Boisvert's [1] design, Larios et al. [3] also calculate a tare reading which is continually updated when there is no bird on the scale.

These boxes were applied in the field to investigate the lesser kestrel in urban areas of Western Europe. It was found that the birds often returned to the nests indicating that they were not stressed.

2.2 Sensing Techniques for Bird Identification

Electronic tracking devices have been attached to birds for the last 50 years and have been used to track their areas of movement and for identification [4][1]. Such tags include the before mentioned PIT tags which use Radio Frequency Identification (RFID) to transmit the information of a bird to a reader [5]. These tags are the size of a grain of rice and are either embedded under the skin of the bird or attached to a leg ring [5]. The microchip in the tag transmits a 10-character RFID code whenever the bird is near a PIT scanner [1]. These tags have a very low failure rate with Boisvert [1] stating that they only experienced one tag failure out of 10 used over a period of 60 days.

2.3 Camera Capturing Techniques

The development of modern camera technologies has lead to their increased usage in the study of avian nesting. This is because cameras allow for the capture of data that would otherwise be impossible. The amount of data obtained through cameras far exceeds the capabilities of human observers [6]. However, using camera's as an observation tool does not come without its issues. Cox et al. [6] found that nest's monitored by cameras have lower predation rates (the proportion of the prey population killed by predators) and as such, can be a cause of unnatural bias to data being collected. Additionally, camera faults and malfunctions cause data loss and corruption, as well as increased expenses for replacing or repairing damaged equipment [6].

2.3.1 Camera traps

Camera traps are remotely activated cameras that are typically used to capture images or videos of wildlife in their natural habitats. They present a growing opportunity to the research of bird breeding and foraging behaviour [7]. Camera traps are commonly used to to study mammals and large terrestrial birds [7]. The use of stationary cameras to monitor bird nests is a well-established practice [6], although habitat constraints often hinder the ability to study small or tree nesting birds [7].

Palencia et al. [8] tested five models of the most frequently used camera traps. They found that false negatives can lead to substantial data loss while false triggers drain battery life and fill storage space. This increase in raw data also increases image processing time. Additionally, as the camera sensitivity increases, the number of false triggers increases. This showed that there is a trade off between power consumption, storage, processing time and the camera sensitivity. The findings of their study offer recommendations for optimal methods in conducting camera trapping research. These recommendations can improve the usefulness and dependability of camera trapping studies and are tabulated in figure 2.1.

They found that it is not necessarily the case that the most expensive camera traps perform the best, and as such careful care should be taken when choosing a camera trap [8].

Key factors	Practical limitations	Practical recommendations
Sensitivity	Trade-off between detection sensitivity and memory and battery capacities because of blank-images	Set high sensitivity only when memory and battery can be checked frequently, and/or when open areas are sampled
Day time	Different probability of detection between day and night time	Include this variability in the models when analysing daily parameters (e.g. activity)
Models	Different probability of detection according to models (makes)	Avoid using different camera trap models in the same study
Height	Bias in probability of detection when camera traps are not set at shoulder height of target species	Monitor exclusively morphologically similar species, consider specific sampling design for each species, or place more than one camera at different heights
Trigger speed	Slower activations as more distant animals enters FOV	Use lure to increase the amount of time in front of the FOV (if it does not violate method assumptions)

Figure 2.1: Factors and limitations when using camera traps [8]

2.4 How to bait birds

In Biosvert's investigation [1], the birds were baited to an automatic feeder to obtain their weight readings and other important data. Sunflower seeds were placed at the back of the feeder to attract the birds towards it. It was found that the birds had to become comfortable with the feeder before taking any seeds from it. After overcoming this initial hurdle, the birds responded well to this method of baiting. Similarly, Haftorn [2] describes using a piece of solidified fat with sunflower seeds inside to bait birds to his electronic scale. However, the research conducted in [9] [10] found that certain birds have a low response to feeders/bait indicating that the efficiency of baiting is sometimes dependent on the type of bird.

The above mentioned studies found a few additional problems that come with baiting birds/wildlife, such as other species consuming the bait[1] [10] and the birds/wildlife becoming reliant on the bait as food effecting their ecological cycle [9].

Research conducted in urban areas [10] found that certain birds dominated feeder use, preventing research into a diverse group of bird species. Additionally, the birds that dominated the feeder began to display defensive and territorial behaviours over the food source. This had a noticeable effect on the ecological cycle of the birds including changes in population size, species distribution, and migratory patterns.

2.5 Power Supply

The two main ways electronic systems are powered in remote areas are: offsite-rechargeable and solar-rechargeable batteries.

The use of solar powered batteries ensures that the system being powered has a constant supply of power without the need for recharging the batteries offsite[11]. The solar panel is able to create energy through solar radiation and use this energy to charge a battery.

Alippi et al.[12] used a battery with a storage capacity that is larger than the required daily capacity of the system. This meant that when fully charged, the batteries would be above system power needs. Solar power could be used to fully charge the batteries. Since the battery capacity is above system needs, on cloudy/rainy days when solar power cannot charge the battery, the battery would still have enough reserves to power the system. One drawback of this system is that the batteries are only being partially charged/discharged which is bad for battery health and results in accelerated deterioration in the battery's capacity [12]. To circumvent this, two batteries can be used with one being charged by the panel until it reaches full capacity whilst the other powers the system. When the battery being charged reaches full capacity and the battery powering the system is completely drained, a microcontroller is used to swap these batteries so that the depleted one gets charged and the full one powers the system. This system provided a solution to prolonging the sensor network life without the need for human intervention.

2.6 Data Processing

As sensor network technology continues to advance, it is becoming increasingly popular in wildlife monitoring systems. Sensor networks often allow for automation of data gathering processes which subsequently allows for more data to be collected. This increase in data quantity requires higher bandwidth transmission solutions and larger storage space [3]. These requirements cannot always be met in remote sensing and resource constrained environments and can put stress on systems. Hence it is becoming increasingly necessary to implement pre-processing (like filtering and compression) techniques on raw data before data transmission or storage [13].

2.6.1 Machine Learning for Filtering

Larios, et al. [3] uses an Artificial Neural Network (ANN) algorithm to estimate the weight of a lesser kestrel falcon. The weight value for a single weighing event needed to be estimated from 16 unstable data points. Since the data was unstable, an arithmetic averaging approach was found to be inaccurate. Instead, ANN and Support Vector Regression (SVR) were considered since they are able to approximate patterns in unknown models based on historical data. ANN and SVR were able to model unstable weighing systems with high accuracy (98.5% and 97.3% respectively). Both algorithms were designed with resource constrained systems in mind since the weighing occurs remotely. Due to it's higher modeling accuracy, ANN was chosen for implementation.

Implementation of the ANN processing algorithm was split into two parts: training and live implementation. The ANN training was performed on a PC using historical data and then deployed on each nest for live estimation. The live system was designed to have two modes: one that processes stable data and another to process unstable data. In the case of unstable data, the trained ANN used the data points to estimate the weight. By using this algorithm they were able to reduce the number of data points from 16 to 1 for each weighing event. Additionally, their design allows them to remotely configure each scale to send the raw data when verifying the ANN. Otherwise, it only transmits the estimated value which reduces the resource demands of the system.

2.6.2 Compression

Data compression is a commonly used tool for reducing data size in systems. It involves targeting and removing redundant data to reduce the number of bits needed to be stored or transmitted [14]. It allows for more efficient use of communication and memory resources. The effectiveness of a compression algorithm entirely depends on the type of data in the system. Compression has two classifications: lossy and lossless. Lossless compression allows for full recovery of the original data after decompression and lossy does not. In general, text-based data (like weight measurement) systems are more likely to use lossless algorithms [14].

Traditional versions of compression algorithms are focused on reducing only memory consumption regardless of power consumption. Sadler and Martonosi [15] explore a form of power and memory efficient compression that can be implemented in sensor networks. Their proposed algorithm is Sensor-Lempel–Ziv–Welch (S-LZW), a dictionary-based compression algorithm specifically designed for sensor networks. Dictionary-based compression scans through data and compiles a dictionary of repeating patterns. Recurring patterns are paired with a unique index in the dictionary. The encoder runs through the data and replaces all exact pattern matches with its dictionary index pair [16].

Szalapski et al. [17] investigates the effectiveness of different compression algorithms in sensor networks. One such algorithm is delta compression which expresses each reading as the difference between it and a reference reading. This algorithm works since the data can now be represented by smaller values. It also explores different types of entropy encoding which represents common patterns with few bits and rare patterns with many bits [18]. The investigation found two-layer (delta then entropy) compression algorithms to be effective in resource-constrained systems.

2.7 Wireless Forms of Data Transfer

2.7.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a newer, lightweight version of Bluetooth that can be implemented on resource constrained systems. BLE, like traditional Bluetooth, uses radio waves to transmit data [19]. There are two types of channels, one for advertising and one for data transfer.

Before exchanging data, BLE devices must connect to each other. To do this, the slave periodically sends out an advertisement on the advertising channels. If the advertisement is not answered, the slave enters sleep mode for a predetermined duration. When a master device wants to connect to a slave, it listens for the advertisement and on reception transmits a connection request to the slave. If the slave receives a request, it will transmit the data to the master using the data channels. The master, usually, determines the sleep/wake-up periods of the slaves [20].

A medical survey [21] found BLE to be a solution for continuous and real-time data transmissions, in a resource-constrained medical health monitoring context. The system required a low-cost, low-power consumption and low latency solution. The system was set up as a BLE112 module (acting as a slave) transmitting to a BLE dongle (acting as a master). The throughput, end-to-end delay, and packet error rate of BLE were found to be acceptable while still remaining energy efficient.

2.7.2 ZigBee

ZigBee is a simple, low cost, low power and short range communication protocol. It is split into three different parts: master, router and end device (sensor node). The master controls the network and stores the data, the routers link groups of devices and the sensor node gathers data. The sensor node can communicate directly with the master when necessary, effectively removing the need for the router in certain applications.A sensor node can be set up to be in a default sleep state and to periodically wake to transmit data. This setup reduces power consumption [22].

When the master is cut-off from the system, it can be setup to remember the sensor node and router configurations. This allows for seamless connection/disconnection from the network. Sensor nodes have similar functionality, allowing them to enter sleep mode without losing context in the network [22].

Tabish et al. [23] state that ZigBee has been widely used as a low power solution to mobile medical applications. They found that although ZigBee is still usable, it is being overtaken by newer technology like BLE.

2.8 Literature Critique

This review has revealed that there is a lot of existing research on both smart and manual bird weight capturing and camera trapping systems. The smart techniques have demonstrated more efficient and useful results. Furthermore, there is an abundance of information on general compression and wireless data transmission techniques. However, applications of these compression algorithms in resource-constrained environments is limited. One paper thoroughly explores the use of machine learning to estimate the weight of birds however, this topic is under-researched. There are various rechargeable battery solutions but there is not much research into small-scale solar powered battery charging systems. This report hopes to explore a solution which combines some of the above researched areas, to create an effective bird weighing and monitoring system.

Chapter 3

Scale and Housing Design

What kind of bird can carry the most weight?

— *The Crane*

This section was completed by Mishay Naidoo, NDXMIS011.

3.1 Introduction

This sub-module focuses on obtaining the weight data of the Southern Yellow-Billed Hornbills and designing the housing for the entire system. The scale system has to obtain accurate weight readings every time a Hornbill visits its nest box. The housing has to attach easily to the nest box and integrate the weighing mechanism correctly. A Perch-Scale design was chosen, where the housing attaches to the side of the nest box and a perch extends across front of the nest. The perch is positioned such that the birds land on it whenever entering/exiting the nest or feeding the chicks/female. The process for designing and implementing this sub-module are detailed in the following sections.

3.2 Design Choices

The design process is divided into three separate sections:

1. The weighing mechanism: a weight recording device had to be chosen that could integrate easily into the Perch-Scale design and obtain weight readings of the birds with reasonable accuracy. An accuracy of 0.3g was deemed adequate after consulting Ben.

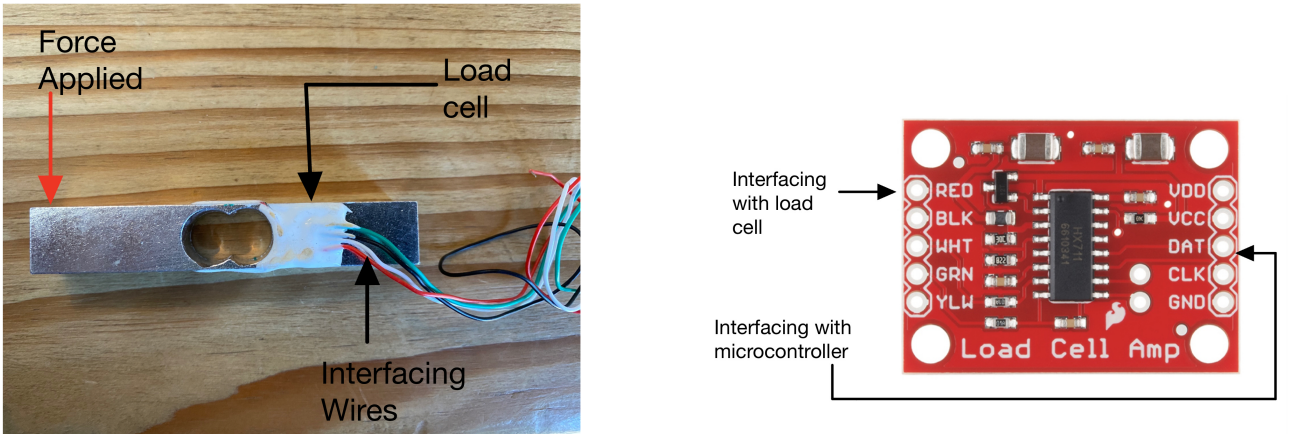
2. The perch: a perch had to be designed that could attach easily to the scale device chosen. The perch would have to apply a force to the scale mechanism in the correct fashion to allow the birds weight on the perch to translate to the scale without significant loss in accuracy.

3. The housing: a housing had to be designed that could hold all of the electronic components including the scale and resources to power and interface with the scale. This housing would need to attach easily to the existing nest box and allow the perch to connect the scale. Furthermore, because of the harsh outdoor conditions of the Kalahari, the housing would need to protect all the components inside from dust/sand, heat and water.

3.2.1 The Weighing Mechanism

The industry standard for weighing objects was found to be load cells. Load cells use strain gauges (electronic components which change resistance when strain is applied to them) arranged in a wheatstone bridge configuration. These load cells output a voltage proportional to the strain on the cell. The load cell selected had to be accurate to less than a gram to meet Ben's requirement and because the Southern Yellow-Billed Hornbills weigh on average around 132g-242g [24]. Furthermore, the load cell would need to integrate with Arduino IDE (C++) as the micro-controller used to interface with the scale was programmed using Arduino. The output voltages of the wheatstone bridge also needed to be amplified and decoded to show weight readings in grams instead of a voltage.

To meet these requirements a 1kg load cell from HKD [25] was selected. The load cell is accurate to 0.3g and can easily support the weight of multiple Hornbills and debris without losing accuracy. The load cell selected looks as follows:



(a) Load Cell

(b) HX711

Figure 3.1: Load Cell and Interfacing Board

This load cell outputs a weight reading based on the moment applied to it by the force in the area indicated in 3.1a. The right end must be screwed onto a support structure and the left end has screw holes that allow weighing platforms to be attached. The load cell can be interfaced through the HX711 board [26]. This board is designed to amplify the voltage output of the load cell and communicate these values to a micro-controller using an ADC (analog to digital converter).

The load cell also needs to account for debris build-up on the perch (dust, leaves and other objects likely to land on the perch in the Kalahari) and re-calibrate itself to account for this. As well as debris, strain gauges are affected by temperature. Therefore, the load cell would also need to be calibrated to account for the vast temperature changes that occur in the Kalahari.

Most scales use a tare feature in which the scale's 0g value is re-calibrated. To account for the debris build up on the perch, a tare value updater was used to re-zero the load cell after a certain time interval.

Scales are generally designed to accurately weigh objects that are stagnant when placed on the device, however because the Hornbill's movements are unpredictable, it was expected that the bird

would be moving when on the perch. This would result in dynamic weight readings recorded by the load cell. To account for this, the weight data would need to be processed (on the supplied micro-controller detailed in chapter 6) to estimate the bird's actual weight using the noisy readings from the load cell.

A Kalman filtering data processing technique was selected to deal with the dynamic readings. The Kalman filter is an estimating tool that uses mathematical equations recursively to obtain estimates from noisy inputs [27]. This filtering method was adapted to work with the noisy weight data recorded by the load cell.

3.2.2 The Perch

The perch design must be long enough to reach the entrance of the nest box and be able to withstand the weight of a Hornbill and potential debris. The weight of the bird on the perch also needs to be translated to a weight on the load cell (this would need to be accounted for through load cell calibration in software).

3D printing was used to design test models of the perch. This is because 3D printing is cheap and it is easy to create multiple iterations of the perch for testing purposes. However, because of the harsh conditions of the Kalahari, the final perch would be made out of wood as it is more heat resistant than plastic. 3D models of the perch were created using Shapr3D, a 3D modelling software. A last design consideration regarding the perch was the torque the bird would cause on the load cell.

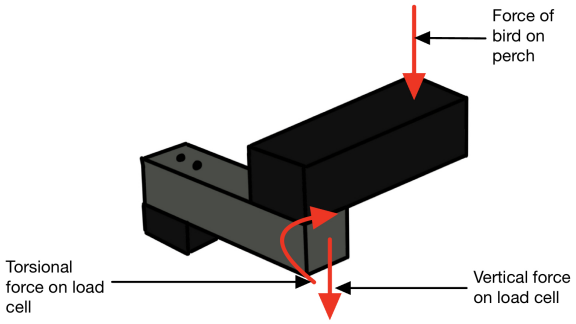


Figure 3.2: Illustration of Torsional Force on Load Cell

As illustrated in 3.2 the bird's weight will not be applied directly downwards on the load cell (the required direction of force for the load cell to read accurately) but will also cause a torque on the load cell. The perch length must account for this to ensure that the twisting force does not effect the accuracy of the weight readings significantly. The readings should not be affected by the location on the perch the bird lands.

3.2.3 The Housing

The housing would need to hold the following:

- The load cell.
- The power module used to power the whole system outlined in chapter 5.

- The camera mechanism to identify the bird outlined in chapter 4.
- Allow the perch to connect to the load cell.

Like the perch, the housing was designed using Shapr3D and 3D printed. The reasons for this are the same as for the perch and the final housing will be made of wood instead of plastic. The housing would need to attach to the nest boxes without disturbing the birds. A diagram of the nest boxes was obtained from Ben and looks as follows:

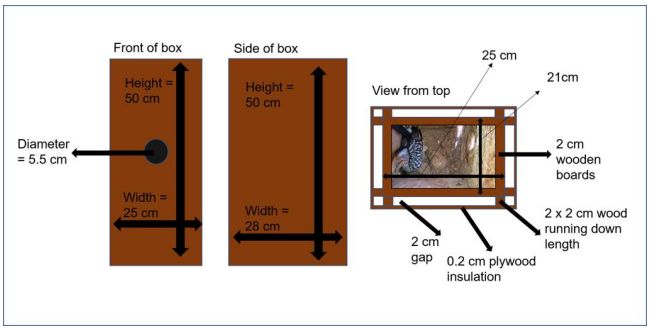


Figure 3.3: Diagram of Existing Nest Boxes

3.2.4 Specifications

The important specifications from this section are summarised in the following table:

Table 3.1: Specifications

Specification Number	Specification Description
S01	The load cell is accurate to 0.3g with noisy readings
S02	The load cell accounts for debris build up on the perch by taring
S03	The birds location on the perch does not overly affect the data readings accuracy (change in torque is negligible)
S04	The perch supports weight up to 350g
S05	The housing holds all the necessary sub-modules
S06	The housing withstands 45° temperatures, water and dust/sand exposure
S06	The housing attaches seamlessly with the nest box

3.3 Prototype Design

The order of prototyping design was slightly different to the order in which design choices were made. This was because the load cell was difficult to test without a perch or a stand to screw the load cell into. The prototype design process instead went as follows:

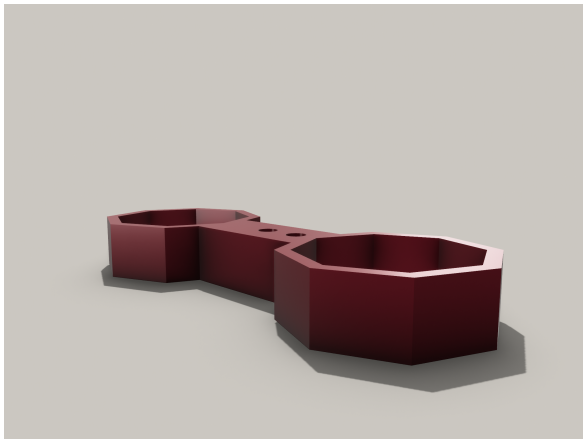
1. Design and print test perch and stand for load cell
2. Interface the load cell with the micro-controller
3. Code a tare setting that zeros the load cell after a given time interval

4. Design a Kalman filter to estimate the birds weight and implement it in code

5. Design and print a housing to hold the system and interface with the perch

3.3.1 Designing the Test Stand and Perch

The perch design was a rectangular beam 145mm long. It had two holes for screws that allowed it to be attached to the load cell. The first iteration of the stand drew reference from RRacer's load cell stand design [28]. The adaptation added a second support ring to make the stand more stable. The design looks as follows:

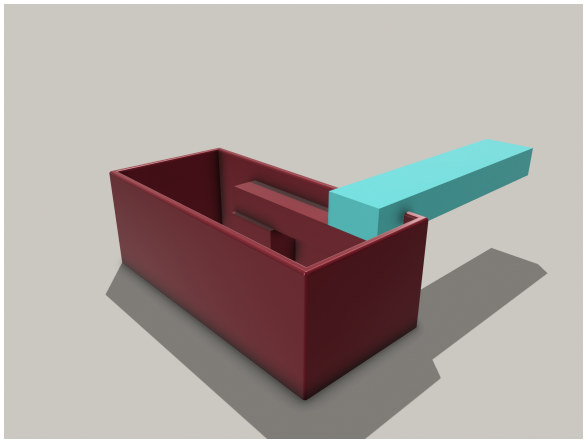


(a) Stand Design V1 Model

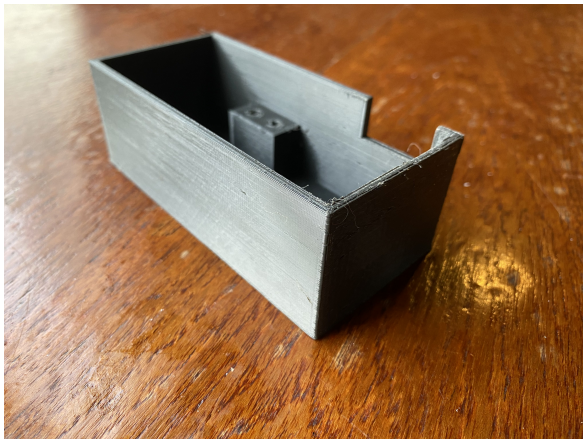
(b) Stand Design V1 3D Print

Figure 3.4: 3D Model Design and Physical Print of Stand V1

However, this design proved to be problematic when testing because it had to be held down to prevent it from tipping when a weight was placed on the perch. An updated stand was designed, which included a box to house the load cell and allow heavy objects to be placed inside to weigh the stand down.



(a) Stand Design V2 Model



(b) Stand Design V2 3D Print

Figure 3.5: 3D Model Design and Physical Print of Stand V2

This stand allowed for the load cell to be screwed into it and included an opening for the perch to fit in.

3.3.2 Load Cell Code

Interfacing the Load Cell and Micro-controller

The load was coded using the HX711 Arduino library [29] which has various built in functions designed to interface easily with the board. Included in this library is a calibration program that can be run on a micro-controller to calibrate the scale before use. This code was used alongside household objects like pencil cases and water bottles to test that the load cell was working correctly with the micro-controller. The weights of these objects were first obtained using a standard kitchen scale, before being compared with the readings obtained from the load cell.

The load cell calibration code works by first automatically taring the cell before any object is placed on it. An object of known weight is then placed on the perch and its weight is input to the micro-controller using the serial monitor on Arduino IDE. This obtains a calibration factor which the load cell stores on the eeprom of the micro-controller and uses to obtain weight readings. This calibration process needs to be repeated every time the micro-controller is restarted.

Once communication and calibration had been completed, the actual weight data transmission was coded. The load cell was coded to transmit weight readings only when the recorded weight was over a certain threshold. This was to prevent unnecessary data being transmitted to the data sub-module 6 (e.g. a series of 0.0g readings whenever there was nothing on the perch).

Designing the Tare Feature

The HX711 library also has a *loadcell.tareNodelay()* function which automatically zeros the load cell. This tare function was coded to occur repeatedly in a predetermined interval provided that the current weight reading on the load cell was less than 80g. This was to prevent the load cell from taring while the bird was on the perch, causing the scale to be zeroed to a higher weight than desired.

The Kalman Filter

The Kalman filter works as follows:

1. Set a prediction of the actual weight of the noisy data being weighed. This prediction is stored in a variable Est
2. Set a prediction of the difference between the estimate Est and the actual weight of the object. This value is stored in variable $Eest$.
3. Set a prediction of the difference between the measurement obtained from the scale and the actual weight of the object. This is stored in $Emeas$.
4. Store every scale reading in a $meas$ variable.
5. Set a Kalman gain as follows:

$$K = \frac{Eest}{Eest + Emeas} \tag{3.1}$$

6. Obtain a new estimate of the actual weight:

$$Est = Est + K * (meas - Est) \tag{3.2}$$

7. Update the estimate error:

$$Eest = (1 - K) * (Eest) \tag{3.3}$$

Steps 1-3 are done once at the start of the code, whilst steps 4-7 are repeated every time the load cell records a new measurement.

3.3.3 Designing the Housing

The housing design is essentially a more robust and bigger version of the Stand Design V2. The final housing has thicker walls (10mm) than the stand box to increase durability and is large enough to house all of the electronics used in the other sub-modules.

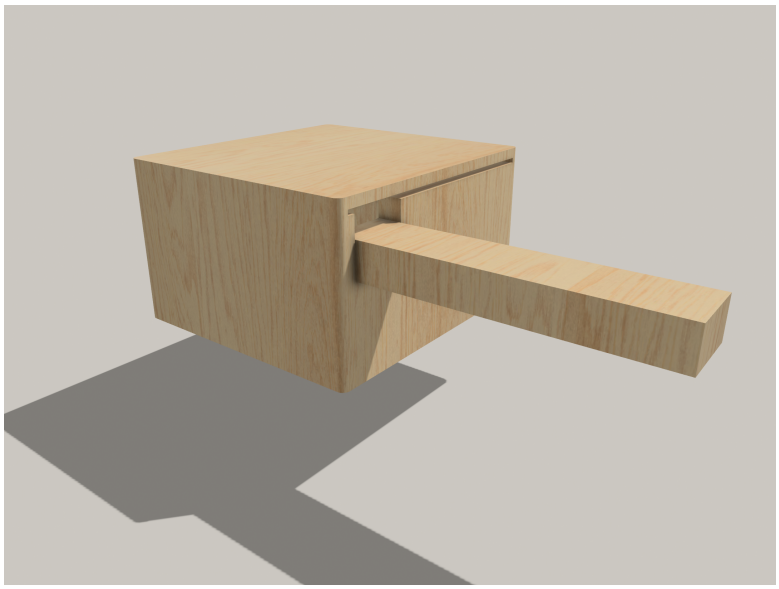


Figure 3.6: 3D Render of Final Housing

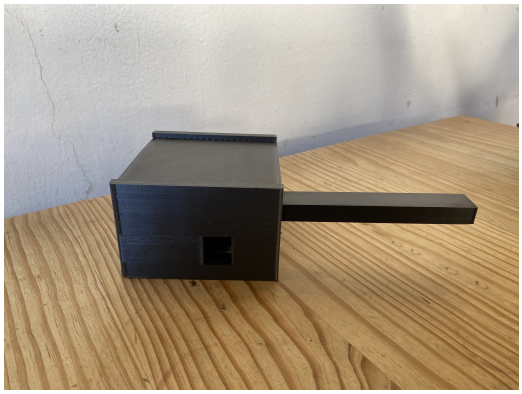


Figure 3.7: Final Housing 3D Print

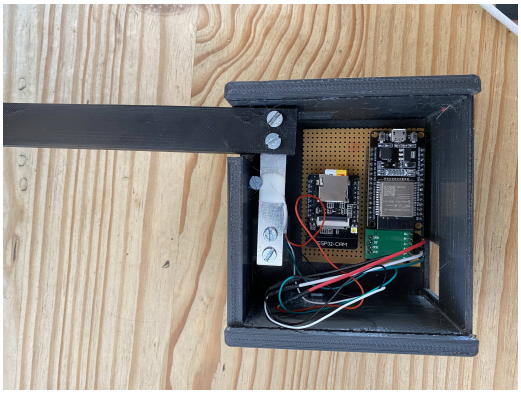


Figure 3.8: Inside of Final Housing

3.4 Testing and Results

3.4.1 Acceptance Tests

The following acceptance tests were derived from the specifications in Table 3.1:

Table 3.2: Acceptance Tests

Test Number	Test Description	Pass/Fail
AT01	Scale Accurate to 0.3g	Fail
AT02	Tare reading accounts for debris buildup on perch	Pass
AT03	Scale Tare's successfully every 10 seconds	Pass
AT04	Readings are unaffected by position of bird on perch	Pass
AT05	Perch can withstand weights 350g or more	Pass
AT06	Can housing hold all system modules	Pass
AT07	Housing can withstand temperatures over 45° and rain, sand/dust	Fail
AT08	Housing attaches easily to nest box	Fail

3.4.2 The Test Stand and Perch

As previously mentioned, the stand design V2 and perch design worked as an effective mechanism for setting up and testing the scale code. The box itself was fragile and could not withstand any harsh treatment but served its intended purpose.

The perch was tested by placing an object on the far end of the perch and observing the weight reading. The object was then moved to the end closest to the stand box and the reading observed. These readings were both the same (at least to the required accuracy of 0.3g) passing acceptance test 4.

A water bottle weighing 400g was successfully weighed on the perch, passing acceptance 5.

3.4.3 Load Cell Code Results

Interfacing with Micro-controller

The main feature that required experimentation for this section was the threshold value for when the weight data would be transmitted to the data sub-module in chapter 6. The threshold selected was 80g (significantly lower than the average weight of the Southern Yellow-Billed Hornbills which, as before mentioned, ranges from 132g-242g [24]). This threshold was deliberately selected to be lower than the Hornbill's weight to account for drastic weight changes. However, this has not yet been tested in the Kalahari, and a possible risk is that heavy debris (over this threshold) could land on the perch resulting in false readings. It does seem unlikely that debris weighing as much as 80g would land on the perch in a single instance, and a gradual build-up of debris will be accounted for by the tare feature.

The Tare Feature

When testing the tare feature it was found that, despite it's name (*tare.noDelay()*), the tare operation was not instantaneous. The operation duration was unclear but once it had begun and a heavy object

was placed on the perch, the tare would zero the load cell to the weight of the object. This was found to occur fairly often and it was predicted that the likelihood of a Hornbill landing on the perch mid-tare was not insignificant. To account for this, the tare operation has been made to occur very often (every 10 seconds). This means that, should a bird disrupt a tare, the load cell would only be incorrectly zeroed for 10 seconds before it would fix itself by taring again. This was tested by deliberately disrupting the tare and waiting 10 seconds before using the perch again. The load cell would not pick up the object used to disrupt the tare within those 10 seconds, but after re-taring (without the object on it) it began recording the object's weight correctly. This mechanism successfully fixes incorrect tares, minimising data loss thus passing acceptance test 3.

Furthermore, the effects of debris build up on the scale was successfully mitigated by the tare. This was tested by weighing an object of known weight on the perch. This object was then removed and a new object placed on the perch. The original object was then replaced on the perch after one tare operation had been completed. It was then observed that the new readings matched the original readings, despite there being two objects on the perch. This passed acceptance test 2.

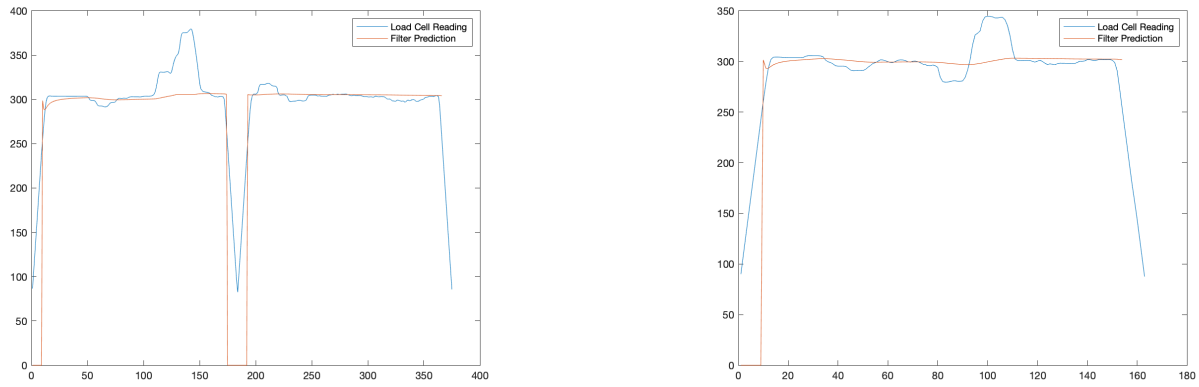
The Kalman Filter

Unfortunately, testing data using actual birds was not obtained and therefore noisy weight data was created. This was done by simulating movement on the perch using an inanimate object. The object used was a 304g glass cup, which was placed on the perch and moved up and down slightly, causing the weight readings to increase and decrease dynamically. This simulated data was not ideal, and had fairly extreme deviations in weight but was adequate for testing the Kalman filter.

The steps outlined in 3.3.2 were followed to design the filter:

1. An estimated weight of $Est = 350$ was chosen to test that estimates significantly different to the actual weight still result in a good filter.
2. The error in the estimate was selected to be $Eest = 200$. This was made unnecessarily large to test that the filter would still work with large errors.
3. The error in measurement was chosen to be $Emeas = 50$.

Steps 4-7 were then iterated through to obtain a weight estimates for each reading. The actual load cell readings and the weight estimates were plotted in MATLab.



(a) Filter with First Set of Noisy Data

(b) Filter with Seconds Set of Noisy Data

Figure 3.9: Kalman Filter Results

The noisy data seen in Figure 3.9a was made to simulate the bird jumping on and off of the perch. It can be seen that the filter deals with the fluctuations very well and has a stable estimate around 304g. Its estimates range from 299g-304.396g.

The noisy data seen in Figure 3.9b was less extreme than in Figure 3.9a but still had large fluctuations. However, the filter struggled to achieve a correct estimate, settling on a final value of 302.02g. This is likely due to the noisy recording in this example always being below the 304g value. It is unlikely that the noisy data obtained from the bird will always be below the bird's actual weight.

The above results indicate that the filter was not able to accurately predict the weight of the cup for all use cases, failing acceptance test 1.

3.4.4 The Housing

The housing is large enough hold all of the electronic components used in all sub-modules and allows for the perch to be connected to the load cell. This housing design passes acceptance test 6.

However, the housing has small openings where the perch is connected. This resulted in water and dust accumulating inside the housing when left outside. This failed acceptance test 7.

Lastly, the housing was not able to attach easily to the nest box, as this design was incomplete due to time constraints. This failed acceptance test 8. A render of what the housing would look like attached to the nest box was generated and is shown below:

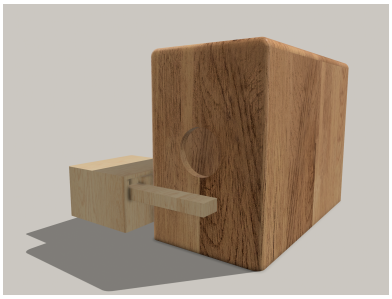


Figure 3.10: Housing Integrated with Nest Box

3.4.5 Bill of Materials

As a result of the limited budget for this project, the housing and scale prototype was made as cheap as possible. The final prototype used the following materials:

Table 3.3: Bill of Materials for Scale and Housing sub-module

Component	Cost
Load Cell	R49
HX711	R60
3D Printing Filament	R20
Total	R129

3.5 Conclusion

The scale and housing sub-module successfully captures weight readings from moving objects on the perch. These dynamic readings are then filtered, resulting in a reasonably accurate approximation of the object's weight with a maximum error of 2g. However, the achieved accuracy falls short of Ben's requirements and could be further improved. The perch design satisfies all specified requirements and can be considered a success. While the housing design provides adequate space for the devices, it does not provide optimal protection. Recommendations for enhancing the scale and housing designs can be found in section 7.1.

Chapter 4

Camera Design

Why did the bird bring a camera to the tree?

– *Because it wanted to capture some 'tweet' memories!*

This section was completed by Alex van Rijswijck, VRJALE001

4.1 Introduction

The camera submodule serves as the image acquisition component of the system. The goal of the subsystem is to identify the birds as they are being weighed. This is an important part of the larger system as it gives meaning to the weight data obtained. Specifically, the camera sub-module focuses on identifying the sex of the Hornbill species by analyzing the colours of the rings attached to their feet.

The subsystem consists of the ESP32-CAM development board shown in Figure 4.1, equipped with a OV2640 camera module shown in Figure 4.2 , which is used to photograph the birds as they are being weighed.

When a bird lands on the perch scale, the camera is triggered and a photo is taken of the bird. The image is then cropped to reduce computational complexity and isolate the bird's feet and processed to determine the colour of the rings.

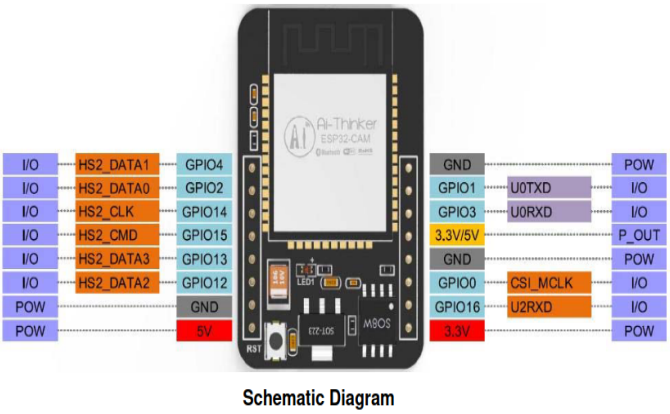


Figure 4.1: ESP32-CAM Development Board schematic

Figure 4.2: OV2640 Camera peripheral

4.2 Requirements and specifications

The following requirements and specifications are outlined for the camera subsystem.

Table 4.1: Subsstem Requirements

Requirement ID	Requirement	ATP ID
UR01	Capture an image of the bird being weighed	
UR02	Be able to see the ring around the birds feet	
UR03	Identify the sex of the bird	
UR01	Save the results to the SD card	

The user requirements in Table 4.1 were used to define the following specifications:

Table 4.2: Subsystem Specifications

Specification ID	Specification	ATP ID
SP01	Camera triggered when bird lands on PerchScale	
SP02	Crop image to contain feet and ring only	
SP03	Identify the ring colour and associated sex	
SP04	Configure SD card and save image/sex to it	

4.3 ESP32-cam hardware specifications

The ESP32-cam development board is a low-cost and fairly powerful development board with onboard camera connections. Its small size makes it ideal for the required application. The following specifications are relevant to the Camera subsystem:

- Built-in 520 KB SRAM, external 4MB SRAM
- 4GB SD card support
- JPEG, RGB565 image format support

- 180 mA@5V, power consumtion (Flash LED disabled)
- Deep-sleep: up to 6mA@5V, Moderm-sleep: up to 20mA@5V, Light-sleep: up to 6.7mA@5V

4.4 Design choices

Several design choices were made due to the limited memory availability on the ESP32-CAM. The constrained resources restricted the use of certain methods for colour identification. Although OpenCV is a popular image processing package, it requires a conventional operating system and consumes significant memory, making it unsuitable for the camera module. Implementing a machine learning algorithm was also not feasible due to the memory constraints.

To overcome these limitations, a less memory-intensive approach was adopted, and colour thresholding was chosen as the method for colour identification. This approach relies on the assumptions that the background image would be relatively colour-neutral (without bright colours) and that the ring colours would be known in advance. It was also assumed that only a single ring colour would be present at a time. These assumptions enable the efficient implementation of colour thresholding to determine the colour of the ring.

4.5 Design Process

4.5.1 Programming the ESP32 Cam

One of the first challenges that was faced was that the ESP32 Cam does not feature a built-in USB port for programming and communication purposes. This means that an external USB-to-UART converter was required. Common solutions to this issue is the usage of an FTDI Adaptor, however in this case this was not available and instead the a Arduino UNO was used to similar effect.

The Arduino can be used as an FTDI Adapter by setting connecting the Rx and Tx lines, with the corresponding Tx and Rx lines on the cam board. Pin connections can be seen in table 4.3

ESP32 Cam	Arduino UNO
5V	5V
GND 2	GND
GPIO3 (U0RXD)	RX0
GPIO1 (U0TXD)	TX0

Table 4.3: Pin connections for programming with the arduino uno

Additionally it is required to ground GPIO0 on the ESP32-CAM as well as grounding RESET on the Arduino UNO. This ensures that both modules are reset and able to communicate with each other. The daigram in figure 4.3 visualizes the required pin connections.

4.5.2 System overview

The diagram shown in Figure 5.9 illustrates an overview of the proposed solution.

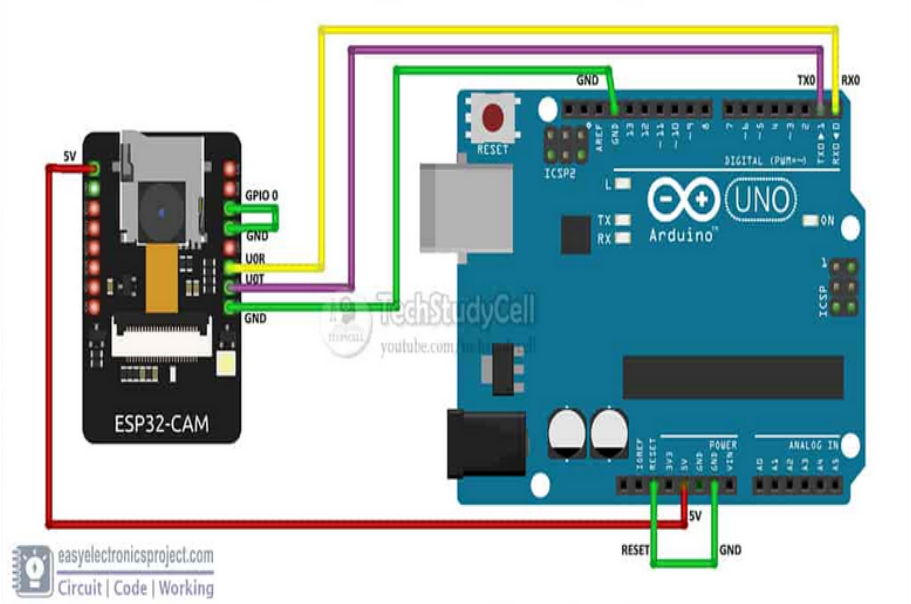


Figure 4.3: Pin connections for programming

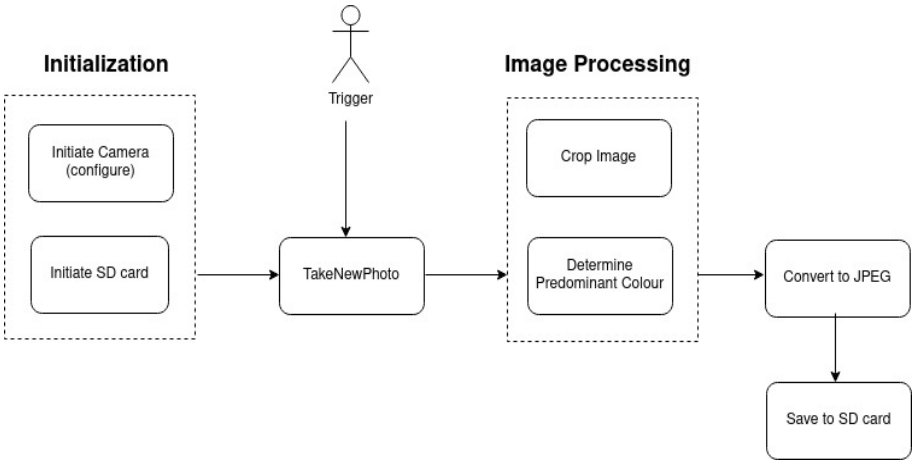


Figure 4.4: Overal system diagram

4.5.3 Initialization

The first step in the design process involves initializing both the camera and the SD card. The ESP32-CAM offers various supporting frameworks and packages, making it advantageous for our purposes. To initialize the camera, we utilize the esp_camera package. The camera_config_t structure is employed to configure the camera with specific settings. Noteworthy members of this structure include:

- pixel_format = PIXFORMAT_RGB565
- frame_size = FRAMESIZE_VGA

The pixel format chosen is RGB565, which is a 16-bit format. It consists of a color band where the most significant 5 bits represent the red pixel value, the following 6 bits represent the green pixel value, and the least significant 5 bits represent the blue pixel value. Although the camera supports the JPEG pixel format, we opted for RGB565 because it allows us to inspect each pixel value individually. In

contrast, the JPEG format contains only metadata and requires additional conversion.

Considering the available RAM limitations, we determined the frame size through trial and error. The VGA frame size, with dimensions of 640x480 pixels, was found to be the maximum size that the camera module could handle. As each pixel is represented by 2 bytes in the RGB565 format, a buffer length of 615KB was required.

For configuring the SD card, we included the FS.h file system library and used SD_MMC.h as the supporting library for the SD card.

4.5.4 Capture image

To capture an image, we utilize the camera_fb_t structure from the esp_camera package. This structure allows us to save the image into a buffer, which can be accessed via the buffer member. Additionally, the structure provides information about the length, width, and height of the image. The image data is stored in the buffer based on the camera settings initialized earlier, specifically using the RGB565 format with a frame size of 640x480 pixels.

To initiate the image capture process, the camera module needs to exit sleep mode which it undergoes during reset. During testing, a physical button was connected to the camera module. When the button is pressed, the camera module exits sleep mode and proceeds to capture the image. This replicates the scenario of a bird landing on the perch, causing the camera to capture an image.

The button can be replaced with a signal that is triggered when the scale takes a reading. When the scale detects the presence of a bird on the perch, it sends a signal to the camera module, activating the image capturing process.

4.5.5 Image processing

Once the image is captured, it is processed to crop and identify the colour of the ring. As previously mentioned the method for colour identification, is thresholding. This is a simple technique that investigates each individual pixels RGB values and determines if it falls within a desired range. This thresholding takes place alongside the cropping of the image. The following method was created:

Method: `identifyAndCropImage`

The `identifyAndCropImage` method crops an input image and identifies the color of the cropped region. It returns the identified color as a string. The method allows for precise control over the cropping region by specifying delta values for the left, right, top, and bottom edges of the image.

Signature

```
String identifyAndCropImage(const uint8_t* buf, size_t width, size_t height, size_t newBufLen,
                int deltaWidthL, int deltaWidthR, int deltaHeightT, int deltaHeightB,
                uint8_t** newBuf)
```

Parameters

- `buf` (`const uint8_t*`): The input buffer containing the original image data.
- `width` (`size_t`): The width of the original image.
- `height` (`size_t`): The height of the original image.
- `newBufLen` (`size_t`): The required length of the new buffer to hold the cropped image data.
- `deltaWidthL` (`int`): The number of pixels to crop from the left side of the image.
- `deltaWidthR` (`int`): The number of pixels to crop from the right side of the image.
- `deltaHeightT` (`int`): The number of pixels to crop from the top of the image.
- `deltaHeightB` (`int`): The number of pixels to crop from the bottom of the image.
- `newBuf` (`uint8_t**`): A pointer to the variable that will store the address of the new buffer holding the cropped image data.

Return Type

- `String`: The identified color of the cropped region in the form of a string.

Memory Management

The memory allocated for the new buffer to hold the cropped image data must be freed by the caller once it is no longer needed.

4.6 Testing and Results

4.6.1 Cropping

During the testing phase, various frame sizes were examined to determine the optimal camera resolution for image cropping. The results of the tests are presented below:

Type	Frame size	Buffer size(KB)	result
UXGA	1600x1200	3750	Failed
SXGA	1280x1024	2560	Failed
XGA	1024x768	1536	Failed
SVGA	800x600	960	Failed
VGA	640x480	614	OK!

Table 4.4: Frame sizes and their required buffer sizes

Based on the results from Table 4.4, it can be observed that the frame sizes of UXGA, SXGA, and XGA all resulted in failed image buffer allocation. This occurred because the representation of RGB565 pixels in 16-bit values required a buffer size that exceeded the available memory. As a result, capturing images at these frame sizes was not feasible.

For SVGA, the camera buffer could be set up successfully. However, attempting to crop and process the image further led to a shortage of available memory, since an additional buffer is required. Hence, it was determined that the maximum frame size capable of capturing and processing an image was VGA.

Using the VGA frame size, it was possible to crop the image. However, an additional restriction was found regarding the size of the cropped image. It was discovered that a maximum additional buffer of approximately 110KB could be allocated, which corresponded to allowing 55,000 pixels in the cropped image.

To demonstrate the cropping process, Figure 4.5, and Figure 4.6 displays the resulting image. The following values were used to determine the dimensions of the cropped image:

- deltaWidthL = 160
- deltaWidthR = 160
- deltaHeightT = 240
- deltaHeightB = 120

Figure 4.5: Original image

Figure 4.6: Cropped image

It's worth noting that the cropped image may appear misaligned with the un-cropped image. In this case, the cropping was performed on an image taken at a similar angle rather than directly cropping from the original image. Since maintaining perfect alignment with the camera can be challenging, the resulting cropped image may exhibit slight variations in alignment.

Despite the alignment differences, the cropped image still provides an approximate representation of the desired section, confirming the functionality of the cropping process. It serves as evidence that the cropping mechanism is effectively extracting the intended portion from the image.

4.6.2 Colour thresholding

Colour thresholding was used by establishing specific limits for the red, green, and blue pixel values. During the testing phase, two colours, red and yellow, were selected for evaluation. Threshold values were determined through iterative trial and error.

To confirm the accuracy of the threshold values, the identified pixels were transformed to white, and the resulting image was saved for further examination. The figures presented below showcase the outcomes achieved from this process.

These steps were performed to assess the effectiveness of the thresholding technique, ensuring that the identified pixels align with the desired colors. The resultant figures serve as visual evidence of the successful colour thresholding and enable further analysis of the processed image.

Figure 4.7: Original image

The optimal values found are listed below

Red Identification:

- red = 8
- green = 14
- blue = 8

Yellow Identification:

Figure 4.8: Red detection

Figure 4.9: Yellow detection

- red = 8
- green = 30
- blue = 15

4.6.3 Acceptance Tests

The following ATP's were created to test the successful implementation of the system.

Test Number	Test Description	Pass/Fail
ATP 1	Image captured successfully	PASS
ATP 2	Image can be saved to the SD card	PASS
ATP 3	Colours can be identified in the image	PASS
ATP 4	Image can be cropped	PASS
ATP 5	Background can be removed	FAIL

Table 4.5: ATP Tests for the Camera module

4.7 Conclusion

The camera sub-module successfully accomplished the tasks of capturing, processing, and saving images. However, due to memory limitations, the image quality and accuracy of colour detection were greatly compromised. Although colours were identified, there were instances of overlapping colour bands, indicating the presence of incorrectly identified pixels.

These observations highlight the necessity for improved threshold values and further iterative testing to achieve more accurate and reliable colour detection. By refining the threshold values, it is possible to enhance the precision of colour identification and mitigate the occurrence of false positives or incorrect pixel classifications.

To obtain more satisfactory results, additional experimentation and fine-tuning of the thresholding process are recommended. By iteratively adjusting the threshold values and retesting, it is possible to improve the overall accuracy and reliability of colour detection in the captured images.

Chapter 5

Power Module

If only we had an infinite power source we could use all day every day...

— *Unknown*

This section was completed by Meg Wilson, WLSMEG005.

5.1 Introduction

The Power Module needs to supply power to the entire system which includes the ESP32 development board and ESP32CAM. Since the system will be operating in the Kalahari the power module needs to withstand wide temperature ranges and harsh conditions. The Kalahari experiences long hours of direct sunlight making solar power a useful solution to alternatively power the system. The current system runs on rechargeable batteries which have to be manually replaced on a regular basis. The new system mitigates this by using solar panels to recharge the batteries.

5.2 Design Choices and Process

5.2.1 Requirements and Specifications

The user requirements have been identified below:

Requirement ID	Requirement
UR01	Power the system remotely.
UR02	The system must be cost effective.
UR03	The system must be easy to maintain and repair.

Table 5.1: System Requirements

The specifications are shown below:

5.2.2 Power connections to the ESP32 Development board

There are three options for powering ESP32 Development board:

1. 5V pin
The ESP32 Dev board has a 5V pin that can withstand voltages from 4.5V to 15V. This power supply can be unregulated because there is a built in voltage regulator on the board.

Specification ID	Specification
SP01	Power the system using rechargeable batteries.
SP02	Charge the batteries remotely using technology like solar.
SPO3	Withstand High temperatures and sandy conditions.
SP04	5V power for Micro(ESP32CAM dev board with ESP32s-micro.)

Table 5.2: Power Specifications

2. 3.3V pin

This pin only allows an input of 3.3V. It is an unregulated pin so exceeding this amount will result in damage to the ESP32 Developemnt.

3. Micro USB

The Micro USB port can be used as a power supply and it can allow serial communication between the host computer and ESP32 Development board.

All of the above power connections can be utilized. However, the micro USB is the easiest connection and provides the regulated 5V necessary to power the ESP32 Development board.

5.2.3 Power Sources to ESP32 Development Board

- Battery Power
- Solar Power

Battery Power

As explored in the Literature review there are multiple batteries that can supply the ESP32 Development Board with 5V. Choosing a battery is important as it needs to withstand the harsh conditions of the Karoo. After exploring multiple options it was concluded that a Lithium Ion battery is best suited to power the system.

Lithium Ion batteries provide a stable power supply which is ideal for powering the ESP32 Development Board. The batteries also have a low self discharge and minimal memory effect allowing them to hold power for extended periods of time. The capacity does not decrease if the battery is not fully discharged before recharging. Since the batteries are rechargeable, solar panels can be used to recharge the batteries.

Lithium 14500 3.7V 800mAh
Specs

Model: 14500
Type: Li-Ion Battery
Nominal Capacity: 800mAh
Nominal Voltage: 3.7V
Colour: Blue
Weight: 19g
Rechargeable Battery: Yes
Rechargeable Times: Up to 500 times

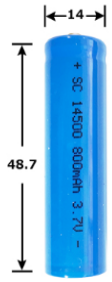


Figure 5.1: Li-ion 14500 Battery 3.7V 800mAh

Price:R38.00
This battery is easily accessible and affordable.

Solar Power

Things to consider when choosing a Solar Panel:

- Output Power
  The ESP32 Development Board needs a supply voltage of 5V meaning the output power of the solar power needs to produce 5V of power.

- Voltage drop
  When a solar panel is connected to a load the voltage drops. This is caused by the resistance of connecting cables or power conversion losses.

- Durability and Weather Conditions
  The solar panel needs to withstand large temperature ranges, wind, rain and sand in the Karoo.

Due to the voltage drop it is best to choose a solar panel with a higher voltage than 5V to ensure the ESP32 Development board is operational. The Solar panel also needs to be durable, lightweight and small.

Solar Panel 4.2W 9V

Specs

1. Rated Power: 4.2W
2. Operating Voltage: 9V
3. Operating Current: 460mA
4. Dimension: 165 x 165 x 3mm
5. Operating Temperature: -20℃ +85℃
6. Package Material: Anti-UV, resistance to yellowing, high transparent polyethylene terephthalate(PET)
7. Base Material: Resistance to deformation, high strength PCB board

Figure 5.2: Solar Panel 9V 4.2W

Price:R124.00

This solar panel was chosen as it can withstand a wide temperature range of -20℃ to +85℃. This is important as temperatures in the Karoo drop very low at night and can be extremely high during the day. The solar panel is small, strong and light making it easy to place on the bird nest and is not easily damaged.

5.2.4 Solar Power Management Board

The system can either be powered by solar panels or batteries. Combining these two power sources allows us to design a more efficient and longer supply of power. This design ensures less need for human interference with the system.

Solar panel input voltage (SOLAR IN): 6V 24V

1. Micro USB input voltage (USB IN): 5V
2. Pin header / USB output (USB OUT): 5V 1A
3. Charging cutoff voltage: 4.2V ±1 %
4. Over discharging protection voltage: 2.9V ±1 %
5. Solar panel charge efficiency:: 78%
6. USB charge efficiency:: 82%
7. 3.7V battery boost output efficiency: 86%
8. Max quiescent current: <2mA
9. Operating temperature: -40℃ 85℃

Figure 5.3: Solar Power Management Board

This management board allows the use of solar panels and a rechargeable battery to power the ESP32CAM and Development board.

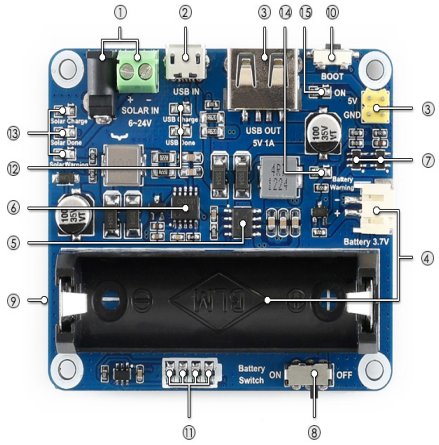


Figure 5.4: Solar Power Management Board

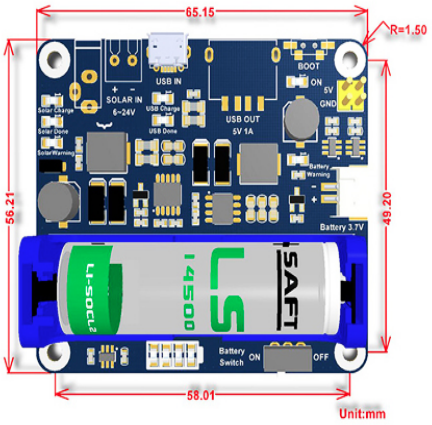


Figure 5.5: Solar Power Management Board Dimensions

There are multiple built in features to this board and the board layout numbering can be found in the datasheet for more detail.

Since the board is powered by solar panels there is a buck chip on the USB input which regulates voltages over the acceptable range to the input voltage 5V/1A. There is an LED light that lights up when the battery is charging by the solar panel and another that indicates when the battery is fully charged from the solar panel. The board is adaptable to solar panels in the range of 6V to 24V. Since the board will not be powered through the USB this will not be explored as an option.

The USB output to the ESP32 Development board is fed through a boost converter to amplify the output to the desired 5V/1A to power the ESP32 Development board.

Battery Protection:
When the battery is charging the chip carefully monitors the voltage of the battery ensuring that the voltage is not above the maximum voltage. It acts like a switch when the voltage exceeds the maximum, the battery is either disconnected from the source or the charging current limited to protect the battery from overcharging. When the battery discharges the chip ensures the voltage does not drop below a minimum threshold by disconnecting the charging current ensuring the battery isn't damaged.

Charging Level:
There are indicators showing the battery charging level. This allows the user to know when the battery is dead. If the battery is connected in reverse there is a warning light that alerts the user. There are also LED's to indicate output power from the USB.

5.3 Prototype Design

5.3.1 First Design

The first design was considered before discovering a complete battery module available on the market. After researching options for powering an ESP32 Development board the better solution was to use the existing module which has built in features to ensure the system is not damaged. The first design also uses a voltage divider circuit to read the battery percentage but this is not an accurate solution

since the resistor voltages are not exactly equal, creating inaccurate readings. Due to time constraints a coulomb counter was not implemented but can be done in future improvements.

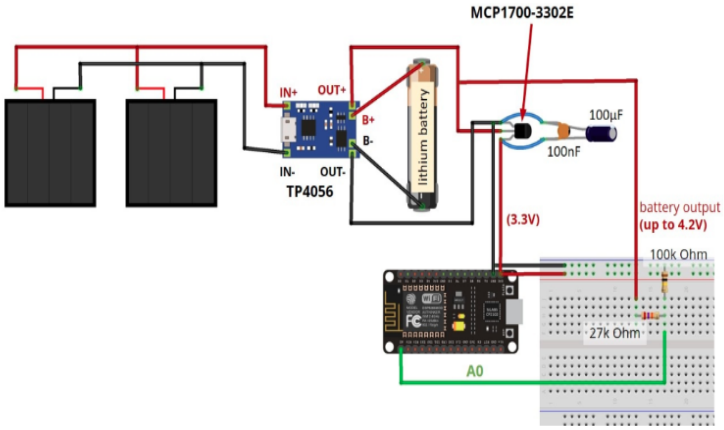


Figure 5.6: First Design Considered

5.3.2 Final Design

The final design includes a 9V 4.2W solar panel, a Lithium Ion 3.7V 800mAh battery and a Solar Power Management Board to power the system.

The final design steps taken were as follows:
1. Solder the positive wire and negative wire to the solar panel.

2. Unscrew the screws on the positive and negative terminals of battery module.

3. Connect the positive wire to the positive terminal on the battery module. Connect the negative wire to the negative terminal on the battery module.

4. Tighten the screws so the wires do not come loose.

5. Using a USB to Micro cable, connect the USB to the output terminal of the battery module and the Micro USB to the Micro Controller.

6. Place the battery in the battery holder.

The final design only makes use of one battery but in future improvements there will need to be more than one battery to enable the system to fully run for 24 hours.

The figure above shows the complete system being tested. The 9V 4.2W solar panel is connected to the solar power management board which regulates the voltage and outputs 5V/1A. The output is connected to the ESP32 Development board via a mirco USB.

5.4 Testing and Results

The figure above shows the battery percentage as can be seen by the 4 LED's next to battery. The 'Solar Charge' LED is on indicating the battery is charging through the solar panel.

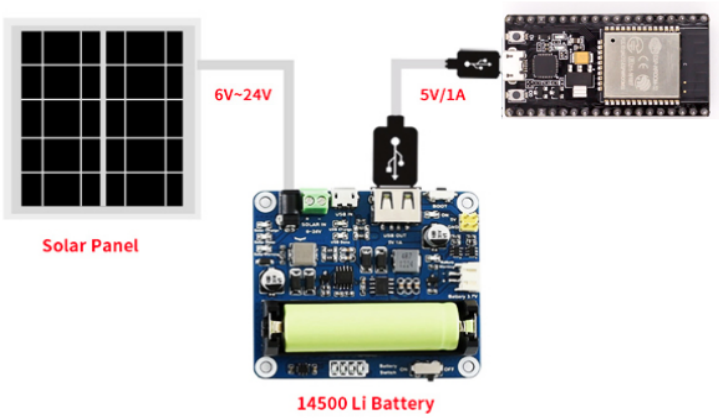


Figure 5.7: Final Power Supply Design

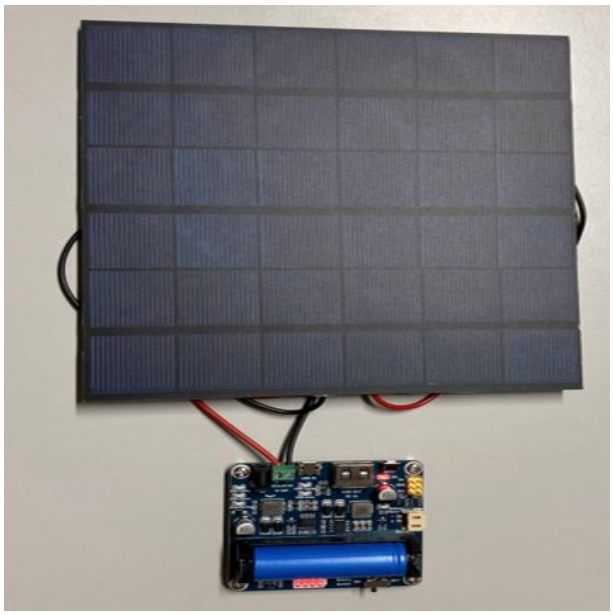


Figure 5.8: Final Power Supply System

5.4.1 Acceptance Tests

ATP01 & ATP02

To test acceptance tests 1 and 2, the solar panel was placed in direct sunlight and the Solar Charge LED turned on indicating the solar panel was supplying power to the system. The battery capacity LED indicators turned on, confirming that the battery was placed in the holder correctly.

ATP03 & ATP04

To test acceptance test 3, the output was be scoped to see if it meets the 5V/1A output. An Arduino was then connected to the system via USB and tested. This showed promising results as the solar panel and battery were both being implemented in the system.

Due to time constraints acceptance test 4 could not be tested with all the components to analyse how long the system will last and reduce the systems power consumption.

Figure 5.9: Solar Power management board LED's lit up

ATP ID	ATP	
ATP01	System is charged by rechargeable batteries.	
ATP02	Batteries are rechargeable without human interaction.	
ATP03	Power subsystem outputs regulated 5V to system.	
ATP04	Test solar charging system.	

Table 5.3: Acceptance Test Procedures

5.4.2 Results

Power Consumption Calculations

Since the birds are not active at night there is no need for the camera to be on during the evening. The camera will be in deep sleep mode until it is triggered by the birds movement and then return to deep sleep mode once it has completed the task of capturing the bird. Since there is no need for a flash as it's light during the day, the camera will take pictures in 'flash off' mode.

Current needed to run the CAM and Microcontroller:
Cam:
o Flash off: 180mA@5V.
o Deep-Sleep: as low as 6mA@5V.
Micro controller:
o Active Mode: 260mA@5V

ESP32 Development board power consumption:

The ESP32 Development board has four modes of operation. With the current system we have designed the board can only be run in active mode meaning it cannot be put into sleep mode to minimize the power consumption.

ESP32 CAM power consumption:
The camera can be operated in two mode, flash off mode and deep sleep mode, to minimize power consumption.

The following table shows the Total Systems Power Budget (24 hours for the camera and 24 hours for the ESP32 development board):

Mode	**Voltage (V)**	**Current(mA)**	**Power(mW)**	**Time(hours)**	**Daily(24 hours)**
Deep Sleep	5V	6mA	30mW	23.92 hours	717.6 mWh
Flash off	5V	180mA	900 mW	0.083 hours	74.7 mWh
Active Mode	5V	260 mA	1300 mW	24 hours	31200 mWh
Total	5V	446mA	2230mWh	44 hours	31992.3 mWh

Table 5.4: Systems Daily Power consumption

The total power consumption in 24 hours is 31992.3 mWh = $\frac{31992.3mWh}{24h}$ = 1333.01 mW.

In the table above it was assumed the camera would be triggered to take photos for 5 mins during the day. The ESP32 development board consumes a large amount of power as it currently cannot be put into sleep mode. This is very power consuming and should be further configured to be able to reduce the power consumption.

It is difficult to accurately estimate when the birds will be landing on the perch and triggering the camera as the birds are only landing on the perch when the male is feeding the chicks or when he is building the nest for the female. For accurate results it would be best to test the system in the natural conditions of the Karoo in order to get accurate data.

Battery Capacity

The system uses a 3.7V, 800mAh Lithium Ion battery. The battery can provide 2960mWh of power and using the battery boost output efficiency of 86% the battery can supply 2545.6 mWh. The power used by the system is $\frac{31992.3}{24}$ = 1333.015mWh. Therefor the battery can power the system for $\frac{2545.6}{1333.015}$ = 1.910 = 1 hour 55 minutes. Other options need to be explored and will be discussed in recommendations.

Solar Power

Using solar we can power the system during the day and charge the batteries simultaneously. Solar panels have a power rating of 4.2W. Taking into account the solar panel charging efficiency of 78%, the output power is 4200mWh*78% = 3276mWh. The battery power is 2960mWh. The batteries can be recharged by solar within $\frac{2960}{3276}$ = 0.904 hours = 54.21 minutes.

The table below shows the power consumption for the system and the power generated by the solar panel assuming there is direct sunlight for 5 hours of the day. The calculations show the battery can only supply power for 1 hour 55 minutes therefor we are only calculating the power consumption during the day. This valid as the birds are in active at night.

The excess power will be used to charge the battery. Due to the solar panels charging efficiency of 78% there will only be 2866.99*78% = 2236.25 mW excess power to charge the battery. To power the system and charge the battery simultaneously, it will require 1 hour 20 minutes to charge the battery.

Mode	Power (mW)
Solar Power	4200 mW
System	1333.01mW
Excess Power	2866.99mW

Table 5.5: Solar panel and System Daily Power Consumption

Since there are many factors effecting the amount of time the system can be powered this is an overall estimate and further improvements will be discussed in recommendations.

5.4.3 Conclusion

The Power Module was designed to supply power to the entire system while taking into consideration the need to use solar panels and manage the conditions in the Kalahari. This was achieved by using a solar panel connected to a management board which houses the rechargeable Lithium Ion battery, which is best suited for the power system. The board is then connected to the ESP32 development board via a Micro USB cable to complete the system providing the necessary power for the project.

Chapter 6

Data Retrieval Design

Why did the data analyst feel like a bird?

— *Because they were owl-ways on the lookout for valuable insights.*

This section was completed by Tristyn Ferreiro, FRRTRI001.

6.1 Introduction

This chapter focuses on the design of a data retrieval system for Perch-Scale, considering the challenging environmental conditions present in the Kalahari. The remote context of the project, significantly influences the design choices explored in this section. Moreover, the design and hardware requirements of other sub-modules must be taken into account to ensure seamless integration. The project's cost restrictions and the emphasis on simplicity, impact the decision-making process. It is worth noting that the development of this sub-module is restricted due to the concurrent development of the other sub-modules.

To address the problem effectively, this sub-module is organized into three sections, each focusing on specific aspects of the design. Design decisions are carefully deliberated and justified within each section. Subsequently, the sections are developed based on these decisions and then tested. The chapter concludes with an assessment of the design's success and provides recommendations for further development.

Based on the problem statement the following user requirements have been identified for this sub-module:

- UR01 - The solution must be cost effective
- UR02 - The retrieval must be simple and intuitive
- UR03 - The data must be stored and formatted in a useful way
- UR04 - Must function in the Kalahari a remote and harsh environment

From these user requirements, the following specifications are defined:

- SP01 - Data should be stored in a structured and organised manner.
- SP02 - Data must be time-stamped with date and time to 1 minute accuracy
- SP03 - Withstand 45°C heat and the sand

- SP04 - Communicate the battery health as a percentage

6.2 Design Choices

Perch-Scale captures important data for each weighing event, including a weight reading, color information from the captured image, and a timestamp. It is important to communicate this data in a coherent, organized, and meaningful manner. Additionally, the data should be easily accessible and retrievable. This will fulfill user requirements and ensure a user-friendly experience with the Perch-Scale system.

6.2.1 Data Storage

Data storage can be achieved through various means, ranging from a simple file storage system to a complex SQL database. Perch-scale uses the ESP32 Cam board (cam board), a light-weight micro-controller with low processing capabilities. Hence running a pure SQL database on the cam board is not feasible. Alternatively, a lightweight version of the SQL server could be considered, but it would impose additional load on the system and affect its operation and power consumption. This is especially significant since the micro-controller's resources are predominantly allocated to image processing, as discussed in chapter 4.

Another approach is to host the SQL server on a separate device and have the cam board send data updates to that server. However, this method requires both devices to be connected to the same network: the internet or a locally hosted network. Since Perch-Scale is designed for remote locations with no access to infrastructure like the internet, this is not a viable option. The cam board can host its own network which the database hosting device could connect to and then retrieve all data from. However, this would require the device to constantly be in range or an alternative storage option in the case that the database could not be accessed.

Considering the simplicity of the gathered data, which includes a timestamp, identified color, and weight for each recording event, implementing a complex database brings unnecessary complications with little benefit in this context. Instead, a file system proves to be a suitable choice since it requires minimal processing resources while reliably storing the collected data. By leveraging data formatting techniques within the files, a level of sophistication can be achieved. This aligns well with the lightweight and low-power nature of the system, making a file system the appropriate choice for the Perch-Scale application in remote locations.

Implementing the file system on the micro-SD card can be achieved without additional hardware or cost. Therefore, for these reasons, the data collected by Perch-Scale will be stored in files on a micro-SD card.

6.2.2 Facilitating Data Acquisition

To accommodate the specific requirements of the Perch-Scale system, certain considerations were made regarding the input/output pins (GPIO pins) and data transfer protocols. Initially, the plan was to use only the ESP32 Cam board for the entire system development. However, the onboard SD card of the cam board rendered all GPIO pins unusable when reading or writing data. Since the load cell for

the scale module required four GPIO pins, an additional micro-controller was introduced to handle the GPIO requirements and scale processing. The battery monitoring system required two GPIO pins.

For reliable data transfer between the micro-controllers, several protocols supported by the cam board were evaluated: UART, SPI, and ESP-Now using WiFi. UART was deemed more reliable and simpler, although it has a shorter distance limitation. SPI offered faster communication but faced issues with the cam board's SPI lines being blocked when the SD card was in use. Although SPI is a bus protocol, the cam board is known for unpredictably blocking the lines when the SD card is used. ESP-Now, a wireless protocol designed specifically for ESP micro-controllers, provided direct, quick, and low-power communication with long-distance capability. However, it introduced complexity for error handling and limited the system design to only ESP micro-controllers.

Considering cost limitations and the desire for system modularity and simplicity, UART was chosen as the communication protocol. Although ESP-Now has advantages in certain scenarios, such as distance and scalability, it was not deemed necessary for the current implementation of Perch-Scale. However, for commercial applications or future iterations, ESP-Now should be considered based on access to ESP micro-controllers and the specific application requirements.

The second micro-controller, intended for handling GPIO pins and data processing, needed to meet several key requirements: support UART, have sufficient RAM for data processing, provide six free GPIO pins, and be cost-effective. Given that all data is sent to and stored on the cam board after processing, a micro-controller with large ROM was not a requirement.

Table 6.1: Key Features for Micro-controller Comparison

Features	**ESP32-S2**†	**ATMEGA8A-AU**†	**ESP32 Dev Board**†
Operating Voltage	2.8V-3.6V	2.7V-5.5V	3.3V-5.5V
Operating Temperatures (C)	–40 ∼ 105	-40 ∼ 85	-40 ∼ 85
RAM	320 KB SRAM	1KByte	520 KBytes SRAM
ROM	128 KB	512 B	448 KB
GPIO pins	43	23	34
UART	Yes	Yes	Yes
802.11 WiFi	Yes	No	Yes
Cost	$1.1705	$1.0933	R150

† *Further information is available in the data sheets.*

Two micro-controllers, ESP32-S2 and ATMEGA8, were considered for additional micro-controller. Both options are affordable, well-documented, and capable of withstanding high temperatures. However, the ESP32-S2 has larger RAM capacity, which is beneficial for data processing in the scale module. The limited ROM capacity of the ESP32-S2 is not an issue since data is not stored on the micro-controller itself but transmitted to the cam board.

Both micro-controllers provide more GPIO pins than necessary and support UART communication. Despite being slightly more expensive, the ESP32-S2 is available in larger quantities on platforms like JLCPCB, ensuring a stable supply for the design. Additionally, the ESP32-S2 can be powered through the cam board, making it more suitable for the system. For these reasons the **ESP32-S2 is chosen**

as the second micro-controller.

To expedite prototyping, instead of ordering a custom-designed PCB for Perch-Scale, an alternative approach is to finalize the schematic for a Perch-Scale HAT and utilize the ESP32 Dev Module as the second micro-controller. The ESP32 Dev Module offers UART support, sufficient RAM, and the required GPIO pins, serving as a suitable substitute for the ESP32-S2 in the prototype. Refer to Table 6.1 for a concise overview of the key features.

6.2.3 Retrieving Data from the device

In order to meet the challenging conditions of the Kalahari, Perch-Scale's housing must be designed with minimal openings, as discussed in chapter 3. Traditional physical methods of retrieving data, such as plugging a device into a port or removing the micro-SD card, present several challenges. A physical port could compromise the housing's integrity or be vulnerable to environmental factors such as filling with sand. Removing the SD card could disrupt the system and require a reset, potentially causing data corruption.

An alternative approach is to use wireless data retrieval techniques. The cam board supports Bluetooth Low Energy (BLE) and can host a local network and website. BLE consumes less power compared to hosting a web server, but it lacks a user-friendly interface. With BLE, retrieving the correct data requires sending specific requests, which can be complicated for users with varying technical expertise. However, websites offer customisable user interfaces and flexibility to adapt to different research needs. Researchers can tailor the website to their specific requirements and easily incorporate new functionalities.

Furthermore, considering the use of rechargeable batteries, as discussed in chapter 5, battery health monitoring is important. Users need a simple way to track battery status, a feature that can be easily facilitated through a website, but would require a specific request through BLE.

Simplifying data gathering for researchers is a primary objective of the system design. A user-focused and flexible data retrieval solution is preferred over one that necessitates remembering request codes. Although BLE reduces power consumption, it lacks user-friendliness. Therefore, to address the user requirements, Perch-Scale will host a local network and website for simple data retrieval.

Web Server Implementation

The cam board has the capability to host both synchronous and asynchronous web servers. While asynchronous web servers offer more functionality and allow multiple clients to access the server simultaneously, this application only expects a single client to access the server at a given time. Therefore, a synchronous web server will be implemented for this purpose.

6.3 Prototype Design

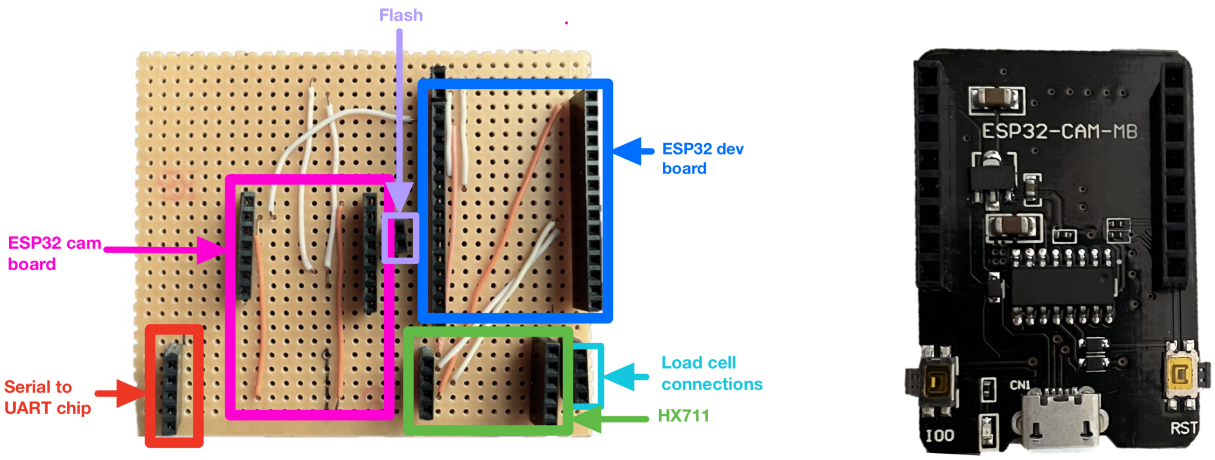
This section details how the design choices made in section 6.2 were developed.

6.3.1 Sub-module Integration

Hardware Interfacing

Initially, the system circuit design was completed on a bread board. Once the connections were confirmed to be correct, using *Hello World* testing, the circuit was designed on a veroboard. The veroboard was iteratively developed before reaching the final prototype version, Figure 6.3.

The first iteration of the veroboard design was focused on interfacing with the cam board. Its primary function was to provide a convenient method for retrieving data from the cam board during initial testing and development. Additionally, it simplified flashing the device by incorporating dedicated header pins where jumper cables could be easily plugged in and out. This version was referred to as the 'Cam Flashing veroboard'. However, due to issues with soldering workmanship, the board became faulty and unreliable. The HKD ESP32-Cam-MB was acquired to make flashing the cam board simpler and faster. These boards are shown in Figure 6.1.



(a) Veroboard v1 - Cam Board Flashing

(b) HKD ESP32-Cam-MB

Figure 6.1: ESP32 Cam Flashing Boards

The second iteration, in Figure 6.2, was designed around interfacing all the components. After designing the circuit on a breadboard, it was redesigned onto the veroboard v2. This iteration of the veroboard still included headers for flashing the cam board.

Initially, the veroboard design functioned as expected, but it encountered issues during the development process. Its faults were attributed to poor circuit design, use of low-quality copper wires and poor soldering workmanship, resulting in loose headers. As a result, the board failed to transfer data between the two micro-controllers.

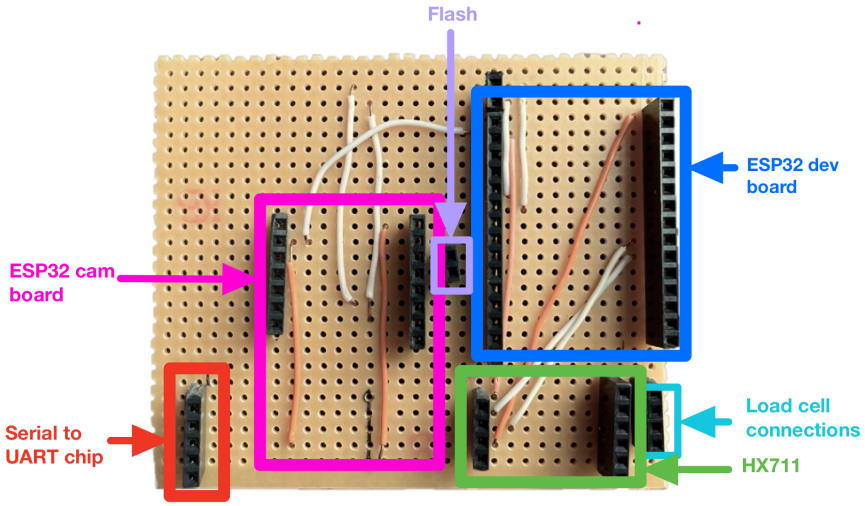
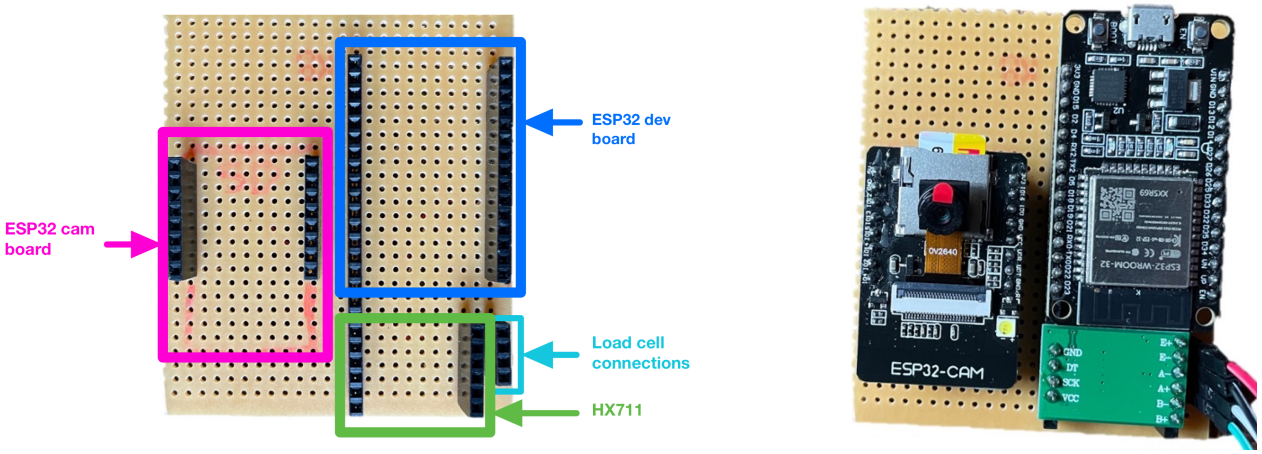


Figure 6.2: Veroboard v2 - Full System

To address the problems encountered with the second iteration, a final veroboard design was developed. The final iteration, as in Figure 6.3, made the following improvements:

- removed the serial to UART converter connections since the cam board's only UART lines are used for communication with the dev board;
- powered the cam board through the dev board rather than a separate power source. *It is important to note that a the final Perch-Scale system, the cam board will power the additional micro-controller but for the prototype this set-up makes the most sense*;
- only the UART receive line on the cam board is connected to the dev board transmission because the cam board never transmits data to the dev board;
- high quality wire wrapping was used instead of low quality copper wires;
- headers are properly secured and aligned.



(a) Veroboard v3

(b) Veroboard v3 with Components

Figure 6.3: Final Veroboard Layout and Design

This design was thoroughly tested using oscilloscopes and multi-meters to test and verify the connections. The final interfacing table for the Perch-Scale prototype veroboard in Table 6.2 gives a clearer indication of the connections.

Table 6.2: Prototype Interfacing Table

Interfacing	**Description**	**Pins/Output**	
I01	UART interfacing between the ESP32 Camera board and the ESP32 Dev board for data transmission	*ESP32 Cam*	*ESP32 Dev*
		5V	Vin
		GND	GND
		UOR	TX2
I02	Scale amplifier/converter connection	*HX711*	*ESP32 Dev*
		VCC	Vin
		GND	GND
		SCK	D33
		DT	D32
I03	Used to get the state of the battery for health monitoring	*Battery Health*	*ESP32 Dev*
		-	-
I04	Interface the HX711 with the load cell	*HX711*	*Load Cell*
		E+	Red
		E-	Black
		A-	White
		A+	Green

Software Interfacing

Two separate source codes are required, one per micro-controller. The dev board acts as a sender and the cam board acts as the receiver and controller. The dev board source code includes everything designed in chapter 3 with a slight modification of UART communication with the cam board. The camera sub-module is integrated with UART receiver source code. An overview of the system flow is shown in Figure 6.4.

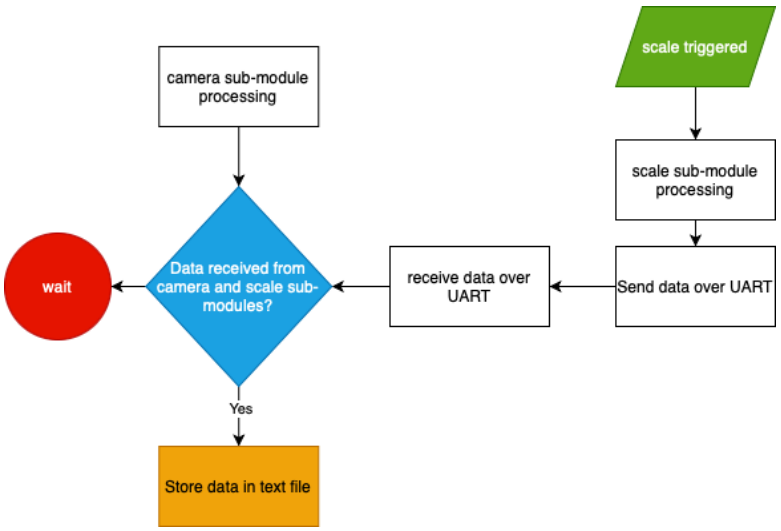


Figure 6.4: Flow Chart showing Flow of Events in Website Setup and Design

6.3.2 Data Storage

The data storage system is designed to write the data received for each weighing even on a new line in a text file on the SD card. Each line in the text file is: *timestamp, colour(s) identified, weight.* A comma-delimited list is used so that the data can easily be ported into excel or as a .csv file. This fulfills user requirement 3 of data stored usefully.

6.3.3 Website

A WiFi network and asynchronous web server is hosted by the cam board. The system is designed to work as shown in Figure 6.5.

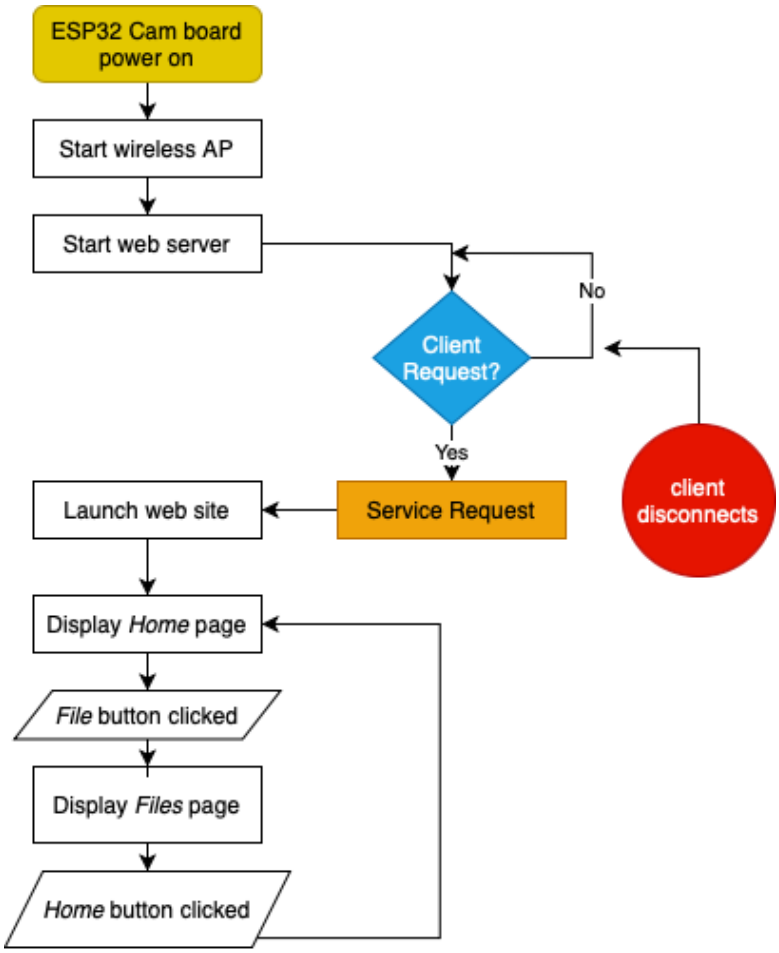
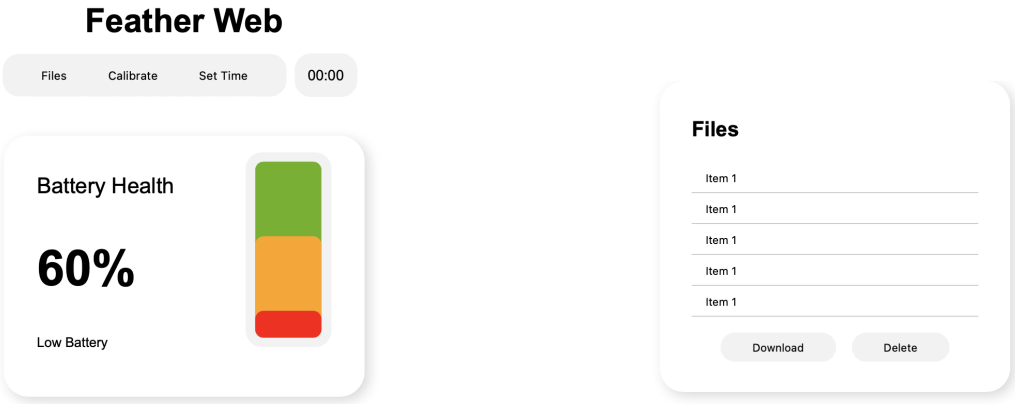


Figure 6.5: Flow Chart showing Flow of Events in Website Setup and Design

The website was designed using HTML and CSS with the aim of being simple and intuitive and is shown in Figure 6.6. The *Calibrate* and *Set Time* functionality have not been designed but are functions that can be added in future iterations. The HTML source code is integrated into the source code uploaded to the cam board.

The HTML website has not been fully integrated with the website in its current form and as such the current website has functionality but does not look like the design in Figure 6.6.



(a) Home Page

(b) Files Page

Figure 6.6: Website Design

6.3.4 Final System Flow

The final system designed works as shown in Figure 6.7. There is physical interfacing to acquire all data from sub-modules and then a wireless network and communication protocol used to facilate data retrieval for users.

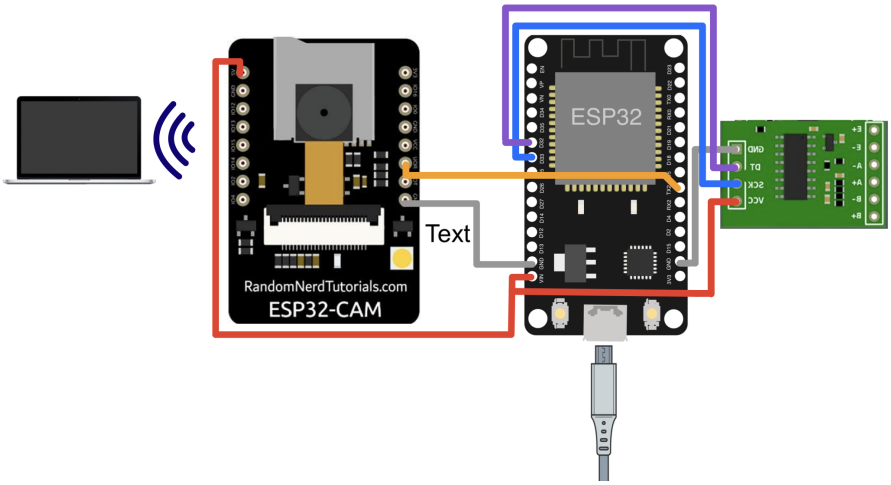


Figure 6.7: Data System Design

6.4 Testing and Results

6.4.1 Acceptance Tests (ATs)

6.4.2 Acceptance Test Procedures (ATPs)

This section details the procedures used to assess if the design meets the acceptance tests in Table 6.4.1.

ATP01 - Write to files on the SD card

Upload the *SD card testing* sketch to the cam board using the ESP32-Cam-MB. Insert the micro-SD card, open a serial monitor and press reset on the cam board. The onboard LED should flash

Table 6.3: Acceptance Tests for Data Retrieval Sub-module

Test Number	**Test Description**	**Pass/Fail**
AT01	Write to files on the SD card	Pass
AT02	Acquire data from scale	Pass
AT03	Acquire data from camera	Fail
AT04	Timestamp data	Fail
AT05	Combine all data and store in text file on SD card	Fail
AT06	Connect to locally hosted WiFi network and website	Pass
AT07	Download data file from website	Pass
AT08	View the battery health on website	Fail

intermittently indicating that 'Hello Ben' is being written to a file on the SD card. Remove the SD card and confirm that 'Hello Ben' has been written to a text file on the SD card multiple times.

ATP02 - Acquire data from scale using UART

Upload the *UART receiver* sketch to the cam board using the ESP32-Cam-MB. Plug all components into the protoype veroboard and use Table 6.2 to connect the load cell. Upload the *UART sender* onto the dev board. Insert the micro-SD card and press reset on the cam board, the onboard LED should flash once if the connection is successful. Open a serial monitor and press 'EN' on the dev board. Calibrate the scale as per the instructions. Place an object on the scale for a few seconds, watching the output on the serial monitor. Remove the object, the cam board LED should flash. Remove the micro-SD card and confirm that the recorded readings match the values in the serial monitor.

ATP03 - Acquire data from camera

Remove all files from the micro-SD card. Upload the *camera with file output* sketch to the cam board using the ESP32-Cam-MB. Insert the micro-SD card and press reset on the cam board, the onboard LED should flash once if the connection is successful. Trigger the camera and observe the onboard LED flash. Remove the SD card and confirm that a file exists on the SD card that contains the name of the colour identified in the image taken.

ATP04 - Timestamp data

Connect a RTC to the cam board and upload the *RTC timing* sketch. Press reset, the onboard LED should start flashing every 10 seconds. Remove the SD card and confirm that the text file has ten second time stamps.

ATP05 - Combine all data and store in text file on SD card

Connect a RTC to the cam board and upload the *combined system* sketch. Without uploading the UART receiver code to the cam board, follow ATP02 to setup the UART connection and calibrate the scale. Place an object on the scale for a few seconds, this should trigger the camera and the cam board LED should flash. Repeat this multiple times. Remove the SD card and confirm that the text file has data formatted as: *timestamp, colour(s), weight*. The number of lines should mach the number of times the system was triggered.

ATP06 - Connect to locally hosted WiFi network and website

Upload the *website testing* sketch to the cam board using the ESP32-Cam-MB. Press reset, and using a device that has 802.11 WiFi capabilities, look for *Hornbill Net*. Connect to the access point, open a browser, navigate to *192.168.4.1* and confirm that the *Feather Web* website exists.

ATP07 - Download data file from website

Follow ATP06 to setup the WiFi and website. Navigate to the 'Files' web page, enter the name of the desired file from the list. Confirm that the file downloads to your device.

ATP08 - View the battery health on website

Follow ATP06 to setup the WiFi and website. Confirm that the battery percentage displays on the home page.

6.4.3 Results

Sub-module Integration Results

Acceptance tests 2 to 5 (AT02-05) are designed to test if sub-module integration was successful. Initially AT02 was tested using veroboard v2 but failed due to faulty design and connections. This lead to a complete redesign which resulted in veroboard v3, the improvements are detailed in section 6.3.1. Using the final Perch-Scale veroboard design, ATP02 was repeated and AT02 was passed. ATP03 could not be carried out because the camera sub-module was not functional within the timeline of this first prototype. For this reason the system fails AT03. Due to time constraints, the RTC was not implemented and ATP04 could not be carried out and the system failed AT04. AT05 is dependent on AT02-AT04, since only AT02 was passed ATP05 could not be tested and AT05 is failed.

Data Storage

Acceptance tests 1 and 5 (AT01 and AT05) are designed to test if the data storage system was successfully implemented and. Following the steps in ATP01, a text file containing multiple lines of 'Hello Ben' was successfully created on the SD card. This confirms that the system passes AT01. As discussed before, AT05 requires AT01-04 to be passed before it can be tested and the system fails AT05.

Retrieving Data

Acceptance tests 6 to 8 (AT06-08) are designed to test if a user can retrieve the system data. ATP06 was conducted resulting in the successful deploy of *Hornbill Net* and *Feather Web*. The system passed AT06. Then the 'Files' button was pressed which opened the a web page with a list of available files and a text box. After entering the name of the desired file, it downloaded and contained the expected data. ATP07 was successfully completed on an iPhone and laptop confirming that the system passes AT07. ATP08 could not be carried out because the battery monitoring circuitry was not functional within the timeline of this first prototype. The website blueprint has been created but not deployed on *Feather Web*. For these reasons the system fails AT08.

Bill of Materials

The final prototype used the materials outlined in Table 6.4.

Table 6.4: Bill of Materials for Data Retrieval Sub-module

Component	**Cost (R)**
ESP32 Cam board	152
ESP32 Dev board	144
Veroboard	20
Total	316

6.5 Conclusion

The data retrieval sub-module of Perch-Scale offers a convenient and user-friendly solution for retrieving weight data from the device. Users can effortlessly connect to Perch-Scale's network using any WiFi-enabled device. Once connected, users can conveniently access the Perch-Scale's web interface through their preferred browser. From the web interface, users can navigate through the available options with ease. Currently, the only functionality available is downloading data files. Additionally, users can only retrieve non-timestamped weight data.

By leveraging wireless technology, the data retrieval design not only mitigates potential risks associated with physical connectors or access points but also provides a streamlined and user-friendly experience. The simplified design ensures that data can be accessed with ease, requiring minimal intervention from the user.

Despite the failure of some acceptance tests, the data retrieval system suffices as a proof of concept and prototype for Perch-Scale. Further recommendations for development are presented in section 7.1.

Chapter 7

Conclusions and Recommendations

Why couldn't anyone see the bird?

—*Because it was in da skies.*

The primary objective of the project was to develop an improved method for collecting weight data on Yellow-Billed Hornbills. Due to time constraints the sub modules were not fully integrated and have only been tested individually. The project was divided into four sub modules, Scale and Housing chapter 3, Camera Design chapter 4 and Power Design chapter 5, Data and Micro Controller Design chapter 6. The scale and housing successfully captures weight readings from moving objects on the perch with a worst case accuracy of 2g. The perch design satisfies all specified requirements and can be considered a success. The housing falls short since it does not providing optimal protection.

The Power Module successfully supplies power to the Perch-Scale by using a solar panel connected to a management board which houses the rechargeable Lithium Ion battery. The board is then connected to the ESP32 development board via a Micro USB cable to complete the system providing the necessary power for the project.

The data retrieval sub-module of Perch-Scale offers a convenient and user-friendly solution for retrieving weight data from the device. Users can effortlessly connect to Perch-Scale's network using any WiFi-enabled device. Once connected, users can conveniently access the Perch-Scale's web interface through their preferred browser.

The camera module was able to successfully capture images when triggered, process the image to determine the predominant colour and save the cropped image to the SD card. The memory constraints posed significant limitations on the overall image quality and the accuracy of color detection. While the system was able to identify colors to some extent, there were significant inconsistencies. This resulted in the miss-classification of certain pixels, undermining the precision of the color detection process. To address this issue, further enhancements are necessary to optimize memory usage and improve colour threshold values.

In summary, the Perch-Scale project achieved its objective of designing a more effective method for gathering weight data on Yellow-Billed Hornbills. Although the complete system implementation and testing were not feasible within the given time constraints, the project's sub-modules can be integrated and implemented to realize the desired outcome given more time.

7.1 Recommendations

Based on the initial Problem Statement, achieved results and discussions, Perch-Scale can be improved in the following ways.

7.1.1 Scale

- Test the perch using actual birds. This data can then be used to create a Kalman filter that approximates the bird's weight more accurately.
- Build the housing out of wood as it is more durable than the 3D printing filament. Seal the housing to waterproof it and reduce openings to prevent dust/sand/water getting inside.
- Use a temperature sensor to observe the effects of temperature change on the accuracy of the load cell. This data can be used to create a look up table that calibrates the load cell for different temperatures.

7.1.2 Camera

- Add additional memory in order to achieve better image quality which would improve the colour detection.
- Add additional colours for identification.
- Investigate alternative methods of colour identification.
- Pre-process the image with a median filter.
- Remove the background dynamically .

7.1.3 Power

- Adding multiple batteries to the system allowing a longer battery life and run time for the system.
- Configuring the Microcontroller to enter sleep mode reducing the power consumption.
- Adding multiple solar panels to the system to charge multiple batteries.

7.1.4 Data

- Add RTC to system and time stamping capabilities
- Fully integrate all sub-modules and re-evaluate the failed acceptance tests
- Fully integrate HTML and CSS website design into functional design
- Add BIOS folder that sets the RTC to the correct time in the case of a brown out. or add RTC battery.
- Add React to better the user interface and add dynamic updating capabilities to the web server
- Calibrate scale and Reset time functionality on website

Bibliography

[1] M. J. Boisvert and D. F. Sherry, "A system for the automated recording of feeding behavior and body weight," *Physiology Behavior*, vol. 71, no. 1, pp. 147–151, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031938400003176

[2] S. Haftorn, "Seasonal and diurnal body weight variations in titmice, based on analyses of individual birds," *The Wilson Bulletin*, vol. 101, no. 2, pp. 217–235, 1989. [Online]. Available: http://www.jstor.org/stable/4162726

[3] D. F. Larios, C. Rodríguez, J. Barbancho, M. Baena, M. L. Angel, J. Marín, C. León, and J. Bustamante, "An automatic weighting system for wild animals based in an artificial neural network: How to weigh wild animals without causing stress," *Sensors (Basel)*, vol. 13, no. 3, pp. 2862–2883, Feb. 2013.

[4] G. R. Geen, R. A. Robinson, and S. R. Baillie, "Effects of tracking devices on individual birds - a review of the evidence," *Journal of avian biology.*, vol. 50, no. 2, 2019-02.

[5] W. Fiedler, "New technologies for monitoring bird migration and behaviour," *Ringing & Migration*, vol. 24, no. 3, pp. 175–179, 2009. [Online]. Available: https://doi.org/10.1080/03078698.2009.9674389

[6] W. A. Cox, M. S. Pruett, T. J. Benson, S. J. Chiavacci, and F. R. Thompson III, "Development of camera technology for monitoring nests," 2012.

[7] P. M. Vaughan, J. C. BUETTE, and B. W. Brook, "Investigating avian behaviour using opportunistic camera-trap imagery reveals an untapped data source," *Ornithological Science*, vol. 21, no. 1, pp. 3–12, 2022.

[8] P. Palencia, J. Vicente, R. C. Soriguer, and P. Acevedo, "Towards a best-practices guide for camera trapping: Assessing differences among camera trap models and settings under field conditions," *Journal of Zoology*, vol. 316, no. 3, pp. 197–208, 2022.

[9] R. S. Stahl, S. J. Werner, J. L. Cummings, and J. J. Johnston, "Computer simulations of baiting efficacy for raven management using drc-1339 egg baits," in *Proceedings of the Vertebrate Pest Conference*, vol. 23, no. 23, 2008.

[10] J. A. Galbraith, D. N. Jones, J. R. Beggs, K. Parry, and M. C. Stanley, "Urban bird feeders dominated by a few species and individuals," *Front. Ecol. Evol.*, vol. 5, Aug. 2017.

[11] C. Xu, Y. Song, and H. Zhang, "Portable and wearable self-powered systems based on emerging energy harvesting technology," Mar 2021. [Online]. Available: https://www.nature.com/articles/s41378-021-00248-z#citeas

[12] C. Alippi, R. Camplani, C. Galperti, and M. Roveri, "A robust, adaptive, solar-powered wsn framework for aquatic environmental monitoring," *IEEE Sensors Journal*, vol. 11, no. 1, pp. 45–55, 2011.

[13] P. Sridhar, A. M. Madni, and M. Jamshidi, "Hierarchical aggregation and intelligent monitoring and control in fault-tolerant wireless sensor networks," *IEEE Systems Journal*, vol. 1, no. 1, pp. 38–54, 2007.

[14] P. Raja and D. Saraswathi, "An effective two stage text compression and decompression technique for data communication," *International Journal of Electronics and Communication Engineering*, vol. 4, no. 2, pp. 233–241, 2011.

[15] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 265–278. [Online]. Available: https://doi.org/10.1145/1182807.1182834

[16] A. Gupta and S. Nigam, "A review on different types of lossless data compression techniques," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 2021.

[17] T. Szalapski and S. Madria, "On compressing data in wireless sensor networks for energy efficiency and real time delivery," *Distributed and Parallel Databases*, vol. 31, pp. 151–182, 2013.

[18] Z.-M. Lu and S.-Z. Guo, "Chapter 1 - introduction," in *Lossless Information Hiding in Images*, Z.-M. Lu and S.-Z. Guo, Eds. Syngress, 2017, pp. 1–68. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128120064000012

[19] E. Mackensen, M. Lai, and T. M. Wendt, "Performance analysis of an bluetooth low energy sensor system," in *2012 IEEE 1st International Symposium on Wireless Systems (IDAACS-SWS)*, 2012, pp. 62–66.

[20] K. Lounis and M. Zulkernine, "Bluetooth low energy makes "just works" not work," in *2019 3rd Cyber Security in Networking Conference (CSNet)*, 2019, pp. 99–106.

[21] F. Touati, O. Erdene-Ochir, W. Mehmood, A. Hassan, A. B. Mnaouer, B. Gaabab, M. F. A. Rasid, and L. Khriji, "An experimental performance evaluation and compatibility study of the bluetooth low energy based platform for ecg monitoring in wbans," *International Journal of Distributed Sensor Networks*, vol. 11, no. 9, p. 645781, 2015.

[22] J. Lin, X. Jin, and Q. Mao, "Wireless monitoring system of street lamps based on zigbee," in *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*, 2009, pp. 1–3.

[23] R. Tabish, A. Ben Mnaouer, F. Touati, and A. M. Ghaleb, "A comparative analysis of ble and 6lowpan for u-healthcare applications," in *2013 7th IEEE GCC Conference and Exhibition (GCC)*, 2013, pp. 286–291.

[24] Wikipedia contributors, "Southern yellow-billed hornbill," https://en.wikipedia.org/wiki/Southern_yellow-billed_hornbill, 2023, accessed on 21 May 2023.

[25]

[26] M. 133943, M. 856276, M. 860839, M. 863964, M. 703397, M. 770429, M. 881622, Fred4, M. 626388, M. 304462, and et al., "Sparkfun load cell amplifier - hx711." [Online]. Available: https://www.sparkfun.com/products/13879

[27] B. Alsadik, "Kalman filter," 2019. [Online]. Available: https://www.sciencedirect.com/topics/earth-and-planetary-sciences/kalman-filter

[28] R. Thingiverse.com, "Load cell testbed by rracer." [Online]. Available: https://www.thingiverse.com/thing:3129439

[29] Olkal, "Olkal/hx711$_a$dc : $Arduinolibraryforthehx71124-bitadcforweightscales.''[Online].Available$: