­­Introduction to Engineering Design

Final Report for

**­­**

**EZ Feed**

Team: Delta

Section 8

Instructors: Glenn Gross and George Gela

Version 3.0 (12/11/2018)

Prepared by

Lily O’Halloran (Junior/ Mechanical)

Brandon Herrada (Junior/ Computer & Systems Engineering)

Daniel Schwank (Junior/ Industrial Management Engineering)

Periklis Sophocleous (Junior/ Civil Engineering)

Daniel Gay (Senior/ Materials Engineering)

Jimmy Li (Senior/ CSE & CS)

**Executive Summary**

The objective of the EZ Feed is to create a cat feeder that is capable of recognizing a cat and then feed it a set amount. The purpose is help the average cat owner control the amount of food the cat is receiving. This can help cats with dietary restrictions and combat obesity. The team began the initial brainstorm for the design by researching current products. From these products, a list of the features was made then put into a concept selection matrix. The results of the concept selection matrix helped create the initial design. The product was then split into subsystems and divided among the group.

As the prototype was being created, unforeseen issues became apparent and many redesigns were implemented. There was testing of the subsystems as they were created. When the subsystems were complete, integrating the systems began. During this process, more issues surfaced and some subsystems had to be modified. The complete prototype had working subsystems along with successful integration. With continued efforts and funds, the team would be capable of fabricating a fully functional cat feeder along with an app.

Table of Contents

[1 Introduction 6](#_Toc532297889)

[2 Project Objectives & Scope 6](#_Toc532297890)

[2.1 Mission Statement 6](#_Toc532297891)

[2.2 Customer Requirements 6](#_Toc532297892)

[2.3 Technical Specifications 7](#_Toc532297893)

[3 Assessment of Relevant Existing Technologies 8](#_Toc532297894)

[4 Professional and Societal Consideration 9](#_Toc532297895)

[5 System Concept Development and Selection 10](#_Toc532297896)

[5.1 Access Control 10](#_Toc532297897)

[5.2 Refilling Mechanism 10](#_Toc532297898)

[5.3 Cat Recognition 11](#_Toc532297899)

[5.4 Concept Selection 11](#_Toc532297900)

[5.5 Proposed Solution 13](#_Toc532297901)

[6 Subsystem Analysis and Design 13](#_Toc532297902)

[6.1 Subsystem 1: Physical Container 14](#_Toc532297903)

[6.1.1 Initial Design 14](#_Toc532297904)

[6.1.2 Redesign 16](#_Toc532297905)

[6.1.3 Final Design 18](#_Toc532297906)

[6.2 Subsystem 2: Cat Detection 25](#_Toc532297907)

[6.3 Subsystem 3: Motor Control & Power 29](#_Toc532297908)

[6.4 Subsystem 4: Food Dispenser 30](#_Toc532297909)

[6.4.1 Design 30](#_Toc532297910)

[6.4.2 Testing & Analysis 31](#_Toc532297911)

[6.5 Subsystem 5: Revolving Lid 32](#_Toc532297912)

[6.5.1 Design 32](#_Toc532297913)

[6.5.2 Testing & Analysis 35](#_Toc532297914)

[6.6 Subsystem 6: Food Weight Management 35](#_Toc532297915)

[6.7 Subsystem 7: Data Management & Transfer 37](#_Toc532297916)

[6.8 Subsystem 8: Web App 40](#_Toc532297917)

[7 Results and Discussion 43](#_Toc532297918)

[7.1 Results 43](#_Toc532297919)

[7.2 Significant Technical Accomplishments 45](#_Toc532297920)

[7.2.1 Attempt to increase RFID scanner range 45](#_Toc532297921)

[7.2.2 3D Printing 45](#_Toc532297922)

[7.2.3 Laser Cutting 46](#_Toc532297923)

[7.2.5 Rclone Configuration on Raspberry Pi 46](#_Toc532297924)

[7.2.7 Web App using JavaScript, React, and NodeJS 46](#_Toc532297925)

[8 Conclusions 48](#_Toc532297926)

[8.1 General Technical Conclusions 48](#_Toc532297927)

[8.2 Possible future improvements 48](#_Toc532297928)

[9 References 51](#_Toc532297929)

[10 Appendices 53](#_Toc532297930)

[10.1 Appendix A: Selection of Team Project 53](#_Toc532297931)

[10.2 Appendix B: Customer Requirements and Technical Specifications 53](#_Toc532297932)

[10.3 Appendix C: Gantt Chart 56](#_Toc532297933)

[10.4 Appendix D: Expense Report 58](#_Toc532297934)

[10.5 Appendix E: Team Members and Their Contributions 59](#_Toc532297935)

[10.5.1 Team Member 1: Lily O’Halloran 59](#_Toc532297936)

[10.5.2 Team Member 2: Brandon Herrada 59](#_Toc532297937)

[10.5.3 Team Member 3: Daniel Schwank 59](#_Toc532297938)

[10.5.4 Team Member 4: Periklis Sophocleous 59](#_Toc532297939)

[10.5.5 Team Member 5: Daniel Gay 59](#_Toc532297940)

[10.6.6 Team Member 6: Jimmy Li 60](#_Toc532297941)

[10.6 Appendix F: Statement of Work 60](#_Toc532297942)

[10.7 Appendix G: Professional Development - Lessons Learned 60](#_Toc532297943)

[10.8 Appendix H: User Manual 60](#_Toc532297944)

[10.8.1 Setting up the device 60](#_Toc532297945)

[10.8.2 Adding cat collar tags 61](#_Toc532297946)

[10.8.3 Everyday operation 61](#_Toc532297947)

[10.8.4 Cleaning the device 61](#_Toc532297948)

[10.8.5 Track your cat 62](#_Toc532297949)

[10.8.6 Troubleshooting 62](#_Toc532297950)

[10.9 Appendix I: Schematics 63](#_Toc532297951)

[10.10 Appendix J: Code 63](#_Toc532297952)

[10.10.1 CatProfile.py 64](#_Toc532297953)

[10.10.2 convert\_to\_csv.py 66](#_Toc532297954)

[10.10.3 Hx711.py 68](#_Toc532297955)

[10.10.4 IntegrationTest.py 73](#_Toc532297956)

[10.10.5 LearnMode.py 78](#_Toc532297957)

[10.10.6 MotorControl.py 79](#_Toc532297958)

[10.10.7 RDM6300.py 81](#_Toc532297959)

[10.10.8 TestDoor.py 82](#_Toc532297960)

[10.10.9 TestFeeder.py 84](#_Toc532297961)

[10.10.10 TestMotors.py 85](#_Toc532297962)

[10.10.11 TestRFID.py 86](#_Toc532297963)

[10.10.12 TestRotation.py 90](#_Toc532297964)

[10.10.13 TestScale.py 90](#_Toc532297965)

[10.10.14 Update 92](#_Toc532297966)

**Revision History**

**Table 1** :Revisions

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Name | Reason for Changes |
| 0.01 | 11/10 | Everyone | Initial document. |
| 1.0 | 12/4 | Everyone | Added most information |
| 1.1 | 12/5 | Everyone | Added code to appendix |
| 1.2 | 12/6 | Everyone | Made User manual and added details to subsystems |
| 1.3 | 12/7 | Everyone | Added pictures of design and build procedure |
| 1.4 | 12/8 | Everyone | Added section 8: conclusions |
| 1.5 | 12/9 | Lily | Formatted document |
| 2.0 | 12/10 | Everyone | Added more information |
| 2.1 | 12/10 | Lily | Formatted again |
| 3.0 | 12/11 | Lily/ Brandon | Final document |

# 

# 1 Introduction

In the US, over 30 percent of households have cats and on average they own two cats. If 60 percent of cats in America are obese then the odds that one of the cats is eating too much is high. The only way to combat obesity is to control that amount that the cat is being fed. If there are two cats how do you know how much each cat is getting feed? The solution is to have a special feeder than can detect how much each cat is eating. Not only can this help the owner to identify the eating habits of their cats, but it can also help them with a feeding schedule. The could help with cats that have dietary need and require a strict feeding schedule. The EZ- Feeder is an autonomous feeder that can help control how much food your cat is eating. This feeder will be durable, compact and affordable.

# 2 Project Objectives & Scope

## 2.1 Mission Statement

This project will result in a working automatic cat feeder device. This device will monitor and control how much food each cat in a household is eating. Monitoring will be done by collecting data about the cats eating habits, such as weight of food eaten and time of day. Control will be achieved by blocking access to the bowl when the cat is not allowed to eat. This product will also block access to any other animals such as dogs which might reside in the same household. The device will also refill the bowl automatically whenever it is empty. The scope of this project is to produce a working prototype which can be used to further improve on its design.

## 2.2 Customer Requirements

Our target customer is the average cat owner. They can own as many of few cats as they want. Their cats can have weight issues or dietary needs as long as they get dry food. This product can be for a busy person or the overly concerned cat owner that just needs a little help with feeding their cat.

At the beginning of this project, a survey was sent out to gather information for the customer. From these result in **Appendix B** the customer requirements were obtained and used to determine the technical specifications of the project in **Table 2.** Some of the customer requirements were not viable due to budget and time constraints. The customer requirements that were not implemented were a detachable bowl, a RFID cat microchip reader and user preferences.

**Table 2**: Customer Requirements

|  |  |  |
| --- | --- | --- |
| Customer Needs | Technical Specifications | Target Values |
| Does not require a lot of power | Voltage is minimum | < 24 volts |
| Can go days without feeing | Larger food storage | 1 gallon containment unit |
| Durable | Can withstand weight of cat | Able to withstand 200 N of downward force |
| Affordable | Cost is minimal | < $75 |
| Compact | Size | < 24”x24”x24” |
| Portable | Lightweight | < 20 lbs |

## 2.3 Technical Specifications

From customer needs the following technical specifications in **Table 3** were produced. The design values were slightly off from the target values.

Table 3: Technical Specifications

|  |  |  |
| --- | --- | --- |
| Technical Specifications | Target Value | Design value |
| Minimum voltage | <24 Volt | The device will operate on 9v power provided by an AC/DC adapter |
| Large food storage | 1 gallon containment unit | A 5”x5”x10” container will be used to hold 1.1 gallons of food. |
| Can withstand weight of cat | Able to withstand 200N downward force | Wooden frame will provide more than 1000N resisting force to weights supported on top of the container |
| Minimal Cost | <$75 | Prototype cost: $120  Expected to be <$50 if mass produced |
| Size | <2’x2’x2’ | 15”x10”x5” |
| Portable | <20lbs | Final weight without food: 20 lbs |

# 3 Assessment of Relevant Existing Technologies

While researching existing designs, many products were found, but none of them use a portion control system. Some examples of commercially available designs are outlined in **Figure 1**.

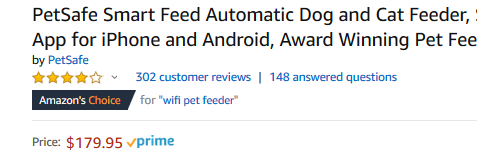


Figure 1: Examples of Existing Similar Products

The three products show how existing designs are either prohibitively expensive, hard to use, provide no dynamic portion control or access control systems. Additionally, each design provides no management over the quantity of food each cat eats, and the owner must manually set portions into the containers.

About 150,000 patents have been found with the description “(internet) (pet or cat) feeder”. The ones that most closely resembles our designs are shown in **Table 4**. They all have one or more of our four main feature requirements: Portion control, Access control, automatic refilling of bowl, and autonomous operation. Portion control is the management of how much each cat eats and block access to food after the cat has eaten its daily allowance. The access control of the food should detect which cat approaches and allow or deny access. When the food is low there is an automatic refilling of bowl that mechanically dispenses food from a big tank to the bowl. Lastly the feeder should be able to autonomously operated by being able to use without an active connection to an external computer or server

Table 4: Patents Similar to Our Proposed Design

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Key Components | | | |
| Patent number | Portion control | Access control | Automatic refill | Autonomous |
| US3340851A | N | N | Y | Y |
| US20150342143A1 | Y | Y | Y | N |
| US20040194714A1 | N | N | Y | Y |
| KR20150000981A | N | N | Y | Y |
| US20160212971A1 | N | Y | N | Y |

From this research, none of the patents in Table 4 combine all four key components of the proposed design. This design therefore does not infringe on any existing patent and thus can be produced and sold freely.

4 Professional and Societal Consideration

The purpose of the EZ feeder is to solve the first world problem of cat obesity and aiding humans in feeding their cats. Overall this does not have a great impact on the world and the environment. The feeder is food safe and does not require a lot of power. This product has a small impact on the economy because it will provide a cost affordable option for people. It will also have a small effect on society by controlling the amount of food cats get and reducing cat obesity. This product can also help busy people who don’t have a lot of time to feed their cats. The EZ Feed just ensures that their cat is getting enough food and eating properly.

**Table 5**: Engineering Solutions Impact

|  |  |  |
| --- | --- | --- |
| **Area of Impact** | **Impact** | **Description of Impact** |
| Global | N | Will not make a considerable global impact |
| Economic | Y | Affordable cat feeder option |
| Environmental | N | Will not affect the environment |
| Societal | Y | Reduce cat obesity |

# 5 System Concept Development and Selection

To being the concept development, initial researcher was done. After looking at existing products and patents, three main feature requirements were determined. For these requirements two options were picked and then put into a concept selection matrix.

## 5.1 Access Control

The two options for Access control is a rotating door or a mechanical trey shown in **Figure 2** and **Figure 3**. The rotating door allows the bowl to be stationary so the scale is not effected. The mechanical tray would require the scale to be within the tray. A concern for both is the height at which the cat will eat from.

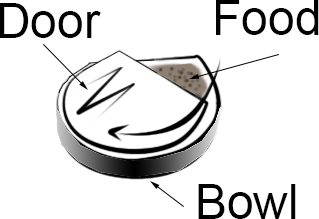
**** ****

Figure 2: Rotating Door Figure 3 : Mechanical Tray

## 5.2 Refilling Mechanism

                 Another important feature of the EZ Feed is how to refill the bowl. **Figure 4** and **Figure 5** display how the food would go from the food container to the bowl where the cat eats from. The main issue with the refilling mechanism is the hardness of the cat food. Cat food could potentially get stick and jam the mechanism.

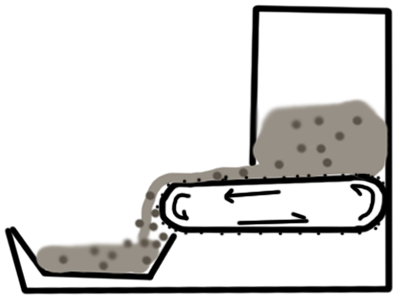
                                      

Figure 4: Conveyor belt Figure 5: Cereal dispenser type

## 5.3 Cat Recognition

The team decided to focus on two different ways to recognize a cat. A RFID tag on collar would be easy to implement by attaching an RFID tag on a cat collar which the cat must wear. This becomes an issue if the cat does not wear a collar. The other option is to use the pet microchipthat is implanted in the cats at the vet. This could an issue if the cat does not have the implant.

The team identifies that many frequencies exist for pet microchipping, the most common, 125 kHz, 128 kHz, and 134.2 kHz. Furthermore, benchmarking of product designs brought the team to recognize that the International Organization for Standardization (ISO) recognizes 134.2 kHz as the approved frequency for pet microchips. Where ISO standard 11784 defines the structure of the microchip information content, and standard 11785 determines the protocol for scanner-microchip communication. However, according to a study conducted by the USDA-APHIS in 2007, approximately 3-5% of pets within the U.S. are microchipped. Of these, 98% are implanted with 125-kHz microchips. In the interest of limited time and anticipated issues with destructive interference, the team concluded that in order to create the most effective product it would be acceptable to employ 125 kHz as the single frequency for the RFID devices.

## 5.4 Concept Selection

To determine the best concepts, the different options for the three features are put into a concept selection matrix in **Table 6**. Each feature is scored with a plus, negative, or zero for the customer needs in the factor column.

Table 6: Concept Selection Matrix

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Access control | | Refilling mechanism type | | Cat recognition | |
| Factor | Score weight | Door | Tray | Conveyor belt | Cereal dispenser | RFID tag collar | Pet Microchip |
| Cost | 3 | 1 | 0 | -1 | 1 | 0 | 0 |
| durability | 3 | 1 | 1 | -1 | 1 | 1 | 1 |
| Difficulty to manufacture | 2 | 1 | -1 | -1 | 1 | 1 | -1 |
| Visual appeal | 1 | 0 | 1 | N/A | N/A | -1 | 1 |
| Availability | 3 | 1 | 0 | -1 | 1 | 1 | 0 |
| Total | | 11 | 2 | -11 | 11 | 7 | 2 |

The bottom row of the concept selection matric is the sum total of the scores, where the highest scores are chosen. From **Table 6**, the rotating bowl door, Cereal dispenser mechanism and RFID tag on collar are the features for the design.

To decide which microcontroller would suit the project better, an Arduino Uno and Raspberry Pi are put into a selection matrix.

Table 7: Microcontroller Concept Selection Matrix

|  |  |  |
| --- | --- | --- |
| Factor | Arduino Uno | Raspberry Pi 3 Model B |
| Cost | 0 | 0 |
| Size | 1 | 1 |
| Random Access Memory (RAM) | -1 | 1 |
| Clock Speed | -1 | 1 |
| Wireless Capability (Wi-Fi) | -1 | 1 |
| Multitasking | -1 | 1 |
| Power requirements | 1 | 0 |
| Extensible Flash Memory | -1 | 1 |
| Programing Languages Supported | 0 | 1 |
| **Total** | **-3** | **7** |

From the selection matrix, the Raspberry Pi was selected because it is fully compatible with all functionalities needed by the team, such as wireless data transmission and upgradeable flash memory allowing for locally stored data. The team's familiarity with Linux-based systems is also a plus.

The software system includes a Web App that wirelessly receives data from the microcontroller. The use of Xbee Socket and UART (Universal Asynchronous Receiver/ Transmitter) for wireless transmission of data give the Raspberry Pi a huge advantage over the Arduino because those systems are built-in, whereas the Arduino requires extra physical components and configurations to achieve the same goal.

## 5.5 Proposed Solution

The configuration chosen through the selection matrix were integrated into one CAD model to give an idea of what the prototype would look like.



**Figure** 6**:** CAD Design Proposal

# 6 Subsystem Analysis and Design

The initial design was split up into 8 subsystems. Each subsystem is given a category of physical, mechanical, electronics, or information system.

Table 8: Subsystems

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Subsystems** | **Summary** | |
| Physical system | Physical Container | A container that stores cat food and all the hardware | |
| Mechanical system | Food dispenser | Controls food into the bowl | |
| Mechanical system | Revolving Lid | The 9V Motor and gears used to open the lid, exposing food | |
| Electronics system | Food Weight Management | The food weight, refilling the bowl and measuring how much the cat has eaten | |
| Electronics system | Cat Detection | The RFID Scanner and external antenna |
| Electronics system | Motor Control & Power | Controlling the motor and the power management | |
| Information system | Data Management & Transfer | Data conversion, updating database | |
| Information system | Web App and Database | The Web App, data collection and database | |

## **6.1 Subsystem 1: Physical Container**

The physical container of the EZ-FEED is made primarily of hardwood, with smaller portions of it made from clear acrylic and plywood. The overall box is 12 inches wide, 6 inches in length, and 18 inches in height.

### **6.1.1 Initial Design**

the container was constructed to be 12” x 6” x 18” tall. There are 2 doors in the container, one in the bottom back so that electronics could be fit into the container easily, and the other is the top piece so that cat food can be placed in the container on the funnel. An initial design was created in NX to get accurate dimensions for the box. This model is for the wood and acrylic being ¼ inch thick.

Clear acrylic was another primary material of the container, allowing for three 6” sections at the top of the container to become transparent windows. Acrylic was chosen primarily for its clear transparency. Acrylic is also strong, and although brittle, that was not a concern for this application. Additionally, acrylic is safer than glass, and will not shatter upon breaking.

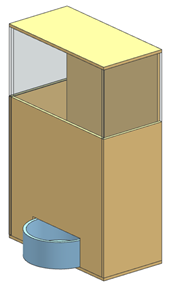


Figure 7: Initial NX model

An acrylic piece of plastic that was 12 x 12 was bought off of amazon. It was then laser cut in order to be the most accurate. Since the acrylic was cut first, the wooden pieces were resized on NX to fit these precisely. **Figure 8** is diagram of how the plastic pieces were cut.

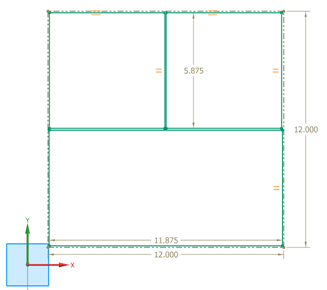


Figure 8: Laser Cut of Acrylic Plastic

The initial wood chosen for the feeder was a ¼ thick MDF panel. From the dimensions in NX, the wooden was cut using a panel saw in the Green building. After the pieces were cut to the right dimensions they were then resined that way it would be food safe.



Figure 9: Resin on the Wood

### **6.1.2** **Redesign**

After cutting the MDF wood and adding resin, there was a lot of issues with the MDF being food safe. This is because the wood would need to be completely covered in resin which would be hard to achieve. Another issue is it did not look as aesthetically pleasing as the team hoped. The redesigned mainly required wood. Maple was chosen because of the hardness. The only maple hardwood that was found was from a lumber yard and was one inch thick. This meant that the design would need to be changed because of the increased thickness

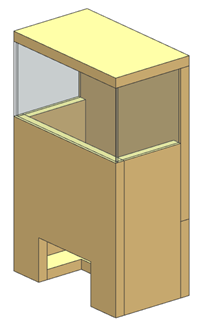


Figure 10: New Design with Thicker Wood

From the CAD design the new dimensions of the wood were determined. This can be seen in **Figure 11**.

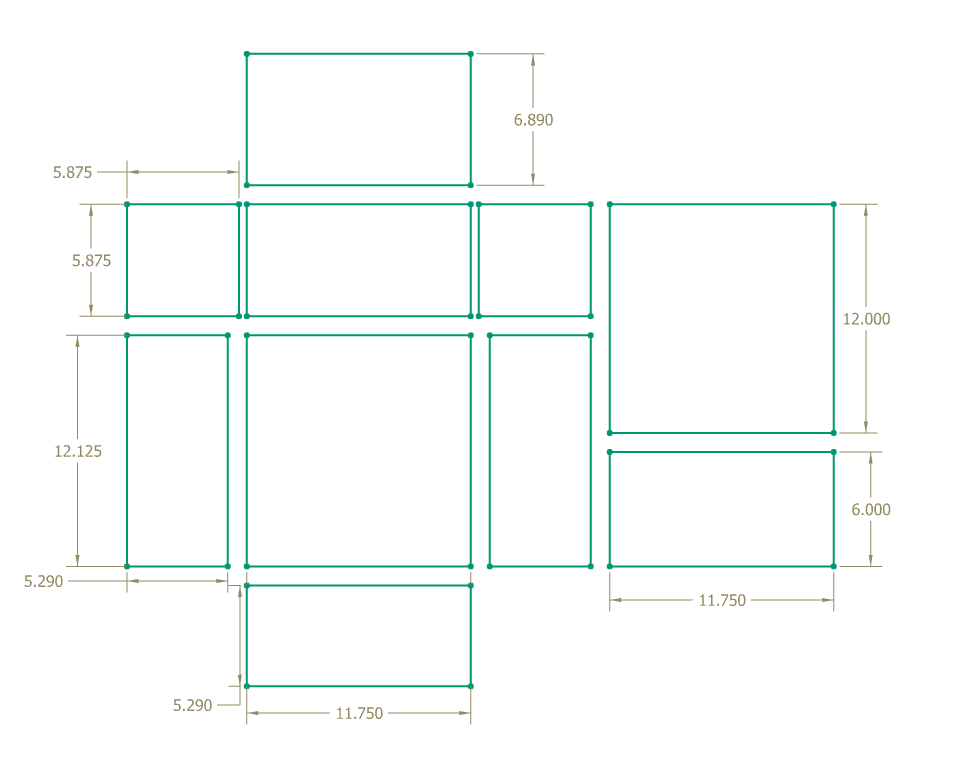


Figure 11: Dimensions of the Wood

### **6.1.3 Final Design**

The container was constructed primarily with Titebond III wood glue, as it is rated safe for indirect food use as well as being satisfactorily strong. To glue the acrylic to the wood, which cannot be done with wood glue since acrylic lacks “pores,” cyanoacrylate was used, which is also food safe when dried and cured. Waterproofing of the wood and acrylic joints was achieved with a food safe clear-drying caulk. The hex screws were used to fasten the magnetic latches in place. Mineral oil and beeswax were chosen as the finish for the container. Given the maple hardwood construction, this finish would darken the wood and make it highly water resistant.

Construction of the physical feeder began once the materials were obtained and the design was set. First, the acrylic pieces were laser cut to size. Laser cutting was done in the Processes Machine Shop and was performed by John Szczesniak, a professor at RPI. The process for this was to draw shapes of the desired dimensions in CAD, transport the file to the laser cutter, line up the cutting nozzle with the acrylic piece, and then cut.

The maple board was sawn into the specified sizes in the Fabrication Shop, again with the help of John Szczesniak. This process first included planning the board to be flat and consistently 0.8” thick. The short edges of the board were leveled with the jointer. Next a section large enough to cut all pieces from was cut from the rest of the 8 foot board. This was cut down into the specified front, back, bottom back, 2 sides, top, and bottom piece using the compound miter saw and table saw. After, the cut-outs on the bottom and front piece were done using a band saw. Cutting was done at a slow pace to ensure straight lines.



Figure 12: Front and Bottom Pieces with Cutout for the Bowl

After the container pieces were all cut, they were glued together using Titebond III, a food safe glue. First the bottom, front, and 2 side pieces were glued together by applying Titebond II to each edge surface that would be glued. Multiple clamps were used for over 30 minutes, and after 15 minutes squeeze-out from the glue was seamlessly scraped off.



Figure 13: Front View of the Initial Glue Up



Figure 14: Rear View of the Initial Glue Up

After these first pieces were glued together, the next step was to glue the upper back piece in the same fashion. This piece was the last to be glued together with the Titebond III.



Figure 15 : Front view of the second glue up - attached the rear panel

Before continuing, the wood pieces of the box were routed with a 3/16” round-over bit on all edges that were fixed and exposed. These included the top surface of the top door, the four vertical side edges, the inner edges on the front face where the bowl goes, and the outermost edge on the bottom back door. Doing this created a flowing and pleasing look in the wood, as well as making it safer by eliminating hard edges and corners.

After the wood pieces were glued together and routed, the next step was to attach the acrylic widows along the top front. To do this, cyanoacrylate adhesive was placed along the edges of the acrylic, and these were attached to the outer edge of the wood, and clamped in place.



Figure 16: Attachment of the Acrylic Pieces

After all three acrylic sides were dried, clamps were removed and a food safe clear-drying caulk was applied to all borders between the acrylic and between the acrylic and wood. This step was done to ensure waterproofing and rigidity.

Other minor materials used for the container include magnetic cabinet latches, small metal hinges, #8 x ½" hex screws, rubber feet, plastic hooks, and insulating foam. The magnetic latches were used to allow two doors in the wood to close securely and stay closed until opened. Adhesive rubber feet were placed on the bottom to prevent the box from sliding on smooth surfaces. Adhesive insulating foam was placed on the top wooden door so that a seal was made between the wood and the acrylic when the door was closed.

The final two pieces were attached next, using hinges and four #8 x ½" hex screws per hinge. Although the hex screws are self-drilling, to ensure the hardwood container did not split or crack, 1/16” pilot holes were first drilled into place for each screw, to a depth of ½". Both the lower back door and the top door were attached in this method, using two hinges each.



Figure 17: Attachment of the Hinges on the Lower Back Door

After the hinges were attached, magnetic cabinet latches were attached to the back door and the top door. Each latch is rated for a 10lb pull force, and once that limit is exceeded the magnetic gives and the door can be opened. Magnetic latches were used for ease of attachment, availability, and ease of opening the doors given no latch had to be lifted or pulled.



Figure 18: Container After Hinges and Magnetic Latches have been Attached

Since a major goal of the physical design was to have a pleasing aesthetic, a cat decal was branded into the top piece, and “EZ-FEED” was placed in the front below the acrylic. The method for doing this is called Pyrography. A pyrography tool I shaped like a pen, and reach.



Figure 19: Cat Pyrography on Top Surface - Halfway Through Production



Figure 20: EZ-FEED Pyrography on Front Face

After both pieces of artwork were completed, the container was finished on all wooden surfaces with mineral oil. After applying two coats of mineral oil, a third coat of warmed mineral oil mixed with beeswax was applied. After soaking for an hour, excess oil was wiped off, and then allowed to further cure. Applying the mineral oil and beeswax allowed the maple to take a pleasing dark color and to be made highly water resistant.



Figure 21 : Container with Final Detailing and Finishing

Plywood was the third primary material used and was only used for the inner funnel within the container. Since the funnel needed to be thin, the maple hardwood was not viable, and additionally due to the high cost of the hardwood, making the funnel from it was undesirable. Unlike the rest of the container, plywood is not very pleasing in appearance. However, given that the funnel would generally be covered by cat food when in use, this was not a concern. To further help the appearance of the funnel, the edges were sanded at 45-degree angles so they would mate flush with the inside of the container, hiding the laminated appearance of the edges of the plywood.

Following the previous steps, only minor additions were made to the container. Insulating foam with and adhesive side was attached to the bottom side of the top piece to mate with the acrylic when closed, to create a waterproof seal. Additionally, adhesive rubber feet were placed on the bottom to prevent slipping. Finally, four plastic hooks were glued into the container, to allow for the RFID antenna to be attached. After these steps, the physical container subsystem was completed. The plastic hooks were later taken out when the RFID antenna did not work properly.

The electronics will be neatly arranged in a compartmental encasing below the cat food container. Due to the possible liquid and physical damage cats are capable of, the encasing for the electronics need to be waterproof, food-safe and durable.

## **6.2 Subsystem 2: Cat Detection**

To detect the cat, a Radio-Frequency Identification (RFID) tag is placed on the front of the cat’s collar. Using this type of technology allows the feeder to keep out unwanted animals and help designate a cat’s profile. When the RFID tag is brought within a specified proximity of the RFID reader, where the proximity is determined by the antenna that is attached to the reader. After benchmarking available 125 kHz RFID scanner chips, the RDM6300 was chosen for its simplicity, compatibility with the Raspberry Pi and ability to use an external antenna. Using Universal Asynchronous Receiver/Transmitter (UART), the Raspberry Pi can read from the breakout board, by reading from the RX (UART Receiver) pin of the Raspberry Pi. The transmission of this data can be read at any specified interval of time as the data is read and transmitted asynchronously from the clock cycle of the Raspberry Pi. **Figure 22** displays how these connections would look from the RDM6300 breakout board and the Raspberry Pi GPIO Pins (Pins with user-specified functions).

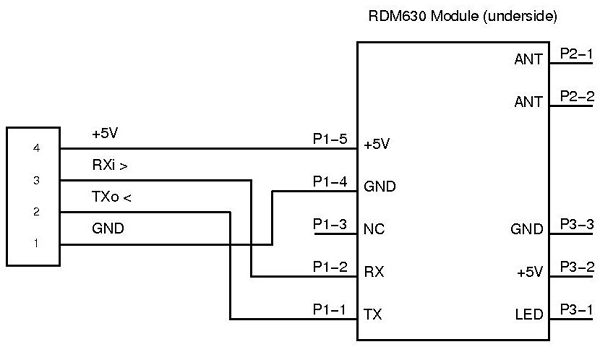


Figure 22 : RFID Reader Data Sheet

To allow a cat to eat with a little distance from the bowl, we were determined to create an external antenna that would allow up to a read range of 4 inches. The posed solution was to mount a 24-turn 10” x 10” square loop of 24 Gauge (24 AWG) Enameled Copper wire (magnet wire) on 4 nails within the box on the front-facing wall.



Figure 23: Failed External Antenna

When mounted on the wall and tested from distances of 0.1 inches, 0.5 inches, 1 inch, 1.5 inches and 2 inches. The wire loop failed from any distance above 0.5 inches. We were unable to determine the cause of this issue, but it was likely a result of one or more of the following factors outlined on page 26 of the Antenna Circuit Design for RFID Applications Manual:

1. Operating frequency
2. Q of antenna and tuning circuit
3. Antenna Orientation
4. Excitation Current
5. Sensitivity of receiver
6. Coding and decoding algorithm
7. Number of data bits and interpretation algorithm
8. Noise

Factors 1-3 relate directly to our design of the antenna, while factors 4-7 relate to the receiver, and 8, the operating environment. For factor 1, we chose to use a frequency of 125 kHz knowing well that the read range of 125 kHz is inferior to ISO standard, 13.56 MHz frequency. With the widespread use of 125 kHz microchips in the United States, it was the 125 kHz frequency was proposed to allow for readability of tags and microchips. This is likely not the issue. The Q-factor or Q, is a way in which to rate the quality or readability of an antenna surrounding the defined frequency of the reader. Using an online tool to calculate the Q-factor it was determined to be 328.03.

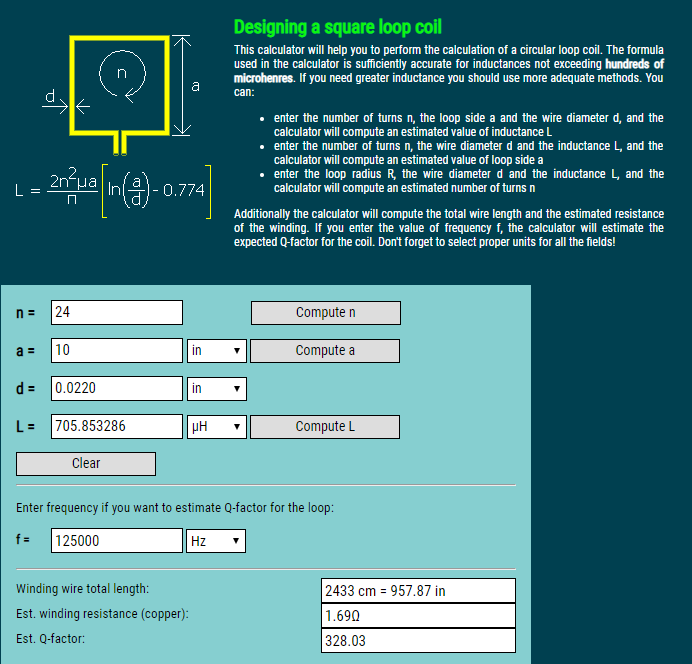


Figure 24: Q-factor Calculator

Using this Q-factor in conjunction with recommendations provided in the Antenna Circuit Design manual, it was clear that our Q-factor had to be greater than 40 in order to read from about 4 inches with our 1-inch diameter tags. In the manual as **Figure 25**, shows, the reader antenna is 3 x 6 inches, however, with aspirations to read from any point within roughly 4 inches of the front facing wall it was important to 10” x 10” antenna, so in theory this would only increase the Q-factor.

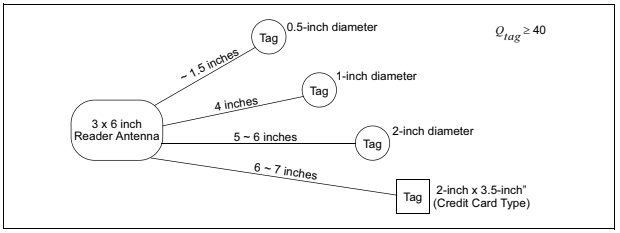


Figure 25: Expected Range of RFID Reader

With respect to factor 3, the antenna orientation, it is important to understand how an antenna works with respect to an RFID tag. The antenna used in this design is at a basic level an inductor. A time-dependent current flows through the antenna (inductor) creating a time-varying magnetic field that induces a voltage (Electomotive Force) to the antenna in the RFID tag which allows current to flow through its antenna coil and allow for transmission of the stored data to be sent to the reader. **Figure 26** defines what how this would look.

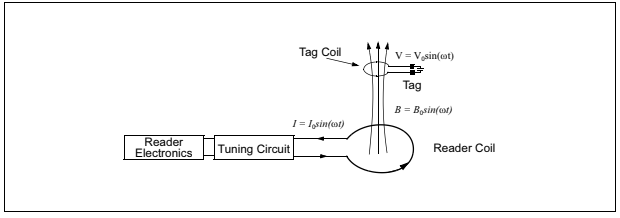


Figure 26: How RFID Works

Our orientation was defined by the ability to read in front of the front facing wall. This meant that the reader antenna had to induce a current perpendicular to the front facing wall. Which meant that in order to induce a voltage in the tag, it would have to be parallel to the reader. Mounting the external reader antenna to the inside front-facing wall was the best option to allow for sleek exterior design and permit readability. Thus, this was a design feature that had no other way of reproduction without compromising the look or readability of the device. Factors 4-7 as stated above were out of the team’s hands when designing the device. The team was constrained by the limitations of the reader’s internal circuitry. Any issues with the accuracy of the reader’s interpretation, excitation current or sensitivity would only be fixed with replacement by a higher end part. Factor 8 details constraints of noise, but the operating environment of this product is ill-defined, as there are many factors that could cause issues with readability. One major factor is the presence of magnets near the front facing wall of the antenna. Thus, placement within 12 inches of an outlet, refrigerator or magnet containing product is not recommended.

Therefore, with the issue of the external antenna not working to design specifications it was a team decision to utilize the antenna included with the RDM6300 breakout board. This antenna limited the range of readability to about 1.5 inches parallel to the induced magnetic field vector of the antenna. When the RFID Reader is listening for a tag it will output its interpretation of the tag, if no 125 kHz tag is present and within range, “00000000000” will be produced. If there is a 125 kHz tag in range, it will be output as a 10-digit Hexadecimal number. This in conjunction with the data management subsystem allows the feeder to differentiate between cat profiles to determine if a cat is within range and able to eat.

## 6.3 Subsystem 3: Motor Control & Power

The microcontroller used in this design in this system was chosen to be a Raspberry Pi 3 Model B. The motors used in both the revolving lid system and the food distribution system are controlled with an L298 Motor Driver Module which allows up to 12 volts to be supplied to the chip. An AC to DC power supply converts 120-Volt Alternating Current (AC) from the wall outlet, into 9-Volt Direct Current (DC), drawing 600 mA current. Mounted on the Motor Driver is a voltage regulator that brings the voltage down from 9-Volts which is then used to power the Raspberry Pi, load cell, RFID Reader, limit switches, and push button. For use of the Raspberry Pi, it must always receive 5-Volts and thus the power supply must always be plugged into the wall. An alternative solution that could be implemented in the future would be to use a capacitor bank to store power and discharge slowly to the raspberry pi.

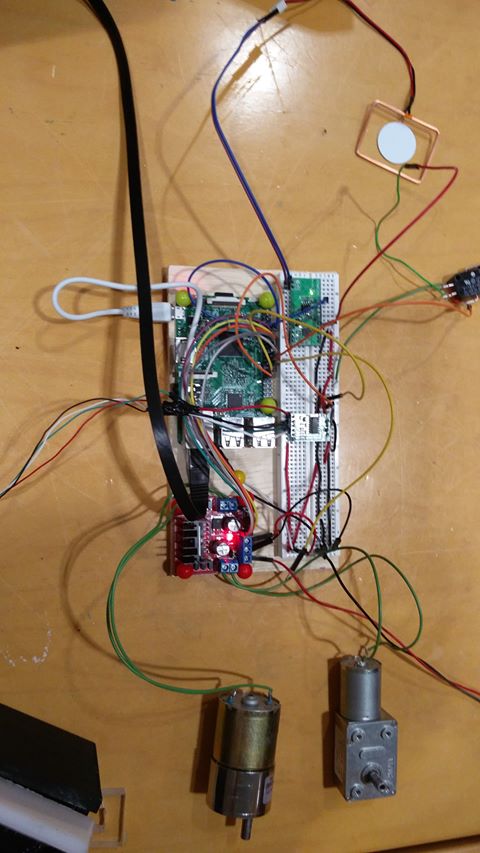


Figure 27: Hardware inside the feeder

The motor speed of the revolving lid and food distribution system will be controlled with Pulse Width Modulation (PWM) provided by the Raspberry Pi. Instead of an on-off signal to the motors, a signal that with frequencies between 100 – 500 Hz will alternate between high and low states for a specified percentage of the period. This is also known as the duty cycle, which refers to percentage in which the signal is logic High. The lower the duty cycle, the lower the effective voltage applied to the motors and thus the slower the motor will turn.

## 6.4 Subsystem 4: Food Dispenser

From the concept selection matrix, the cereal dispenser was selected. This will dispense a portion of food from the food storage area of the feeder.

**6.4.1 Design**

This design was inspired by a cereal dispenser. The dispenser was created with six blades that are curved for optimal rotation. The design was made in Autodesk Inventor, then 3D printed in the Forge. The initial print was a test print with a low PETG.

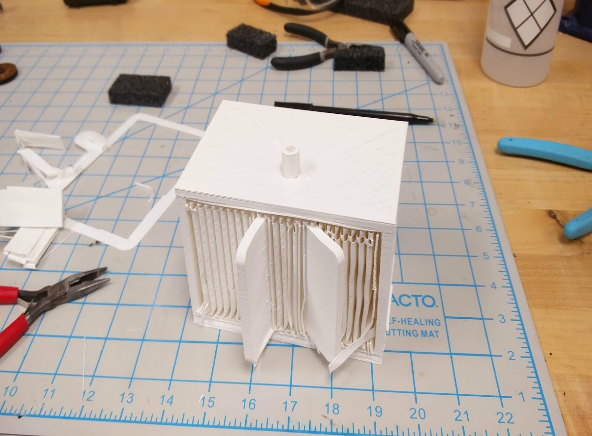


Figure 28: Initial dispenser design

The first print’s main purpose was for sizing which was too large. There was also issues with printing the walls and the blade in one model. The second print was scale down and printed with a higher PETG. The walls of the dispenser and blade were printed separately to ensure no issues.

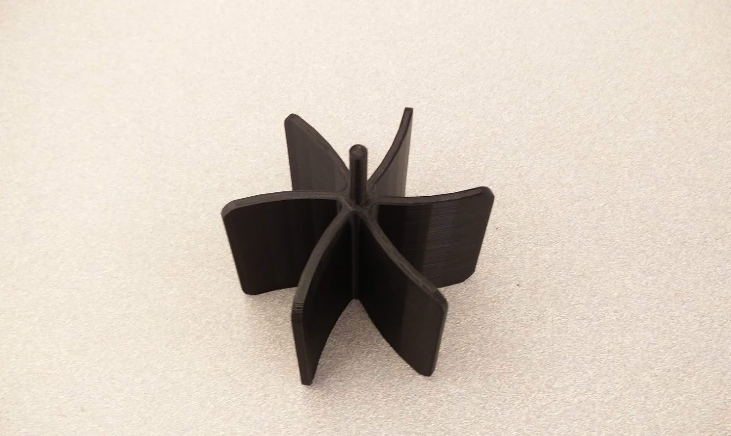
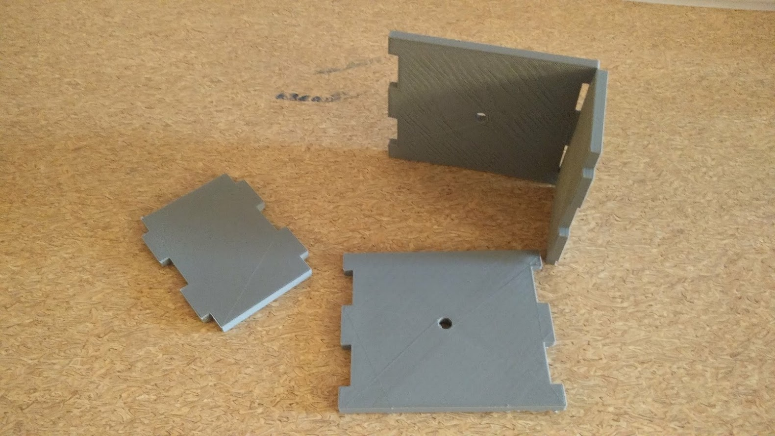
 

Figure 29: Dispenser blade Figure 30: Dispenser walls

The walls were printed in 4 separate pieces to allow the blade to be fitted into place. This also allowed for the pieces to be sanded individually. All of the pieces fit together nicely, but the walls should have been sanded more to reduce the friction between the blade and the wall

The motor used in the design of this subsystem is a 100 Rotations Per Minute (RPM) DC Motor with maximum voltage 12-Volts. The reason 100 RPM was selected is because it offers a large range from 1 RPM to 100 RPM, based on duty cycle readings. In an ideal world this would be the case, however, the effective range of the duty cycle for this motor was from 25% to 100%, at 100 Hz frequency.

### 6.4.2 Testing & Analysis

During the testing phase of this subsystem, there were extensive issues with food getting stuck in the cereal dispenser. When filling the 1-gallon storage container, it was difficult for the motor to run because the food would get stuck on both sides of the rotating blade. This was due to an overlooked design feature.

The first set of testing was finding a speed that would not cause a cat to be scared by the mechanism. The speed for this action was determined to be done with either a relatively low speed. Testing this we started from a reasonably low percentage of roughly 20 percent of the 500 Hz frequency at the time. We immediately ran into issues with this combination. The motor would produce vibrate but not turn the motor. Determined to fix the issue, we lowered the frequency to 200 Hz, resolving the issue of turning but now it was too slow. The next course of action was to increase the speed up to 25 percent, which produced a reasonable non-aggressive speed. Food was then added to the dispenser while the motor was off, when powered on at 25 percent duty cycle, the dispenser jammed. Looking into the dispenser it was easily noticeable that food being piled up on the blades caused the door to jam when food got wedged against the blades opposing the direction of motion. One suggestion to prevent the jamming was to run the motor in one direction for a short amount of time then quickly reverse the motion for a split second to unjam the food. When implemented, little progress was made to unjam the food. After a long series of unsuccessful attempts, we struck gold when one team member suggested covering one half of the dispenser allowing food to only fill one half of the blades. When food was administered to the blades, one side may get stuck but because the other direction did not have food impeding its motion, this allowed food to fall when the quick forward-reversing action from previously described test.

## 6.5 Subsystem 5: Revolving Lid

The Food container lid is used to control the amount of food consumed by each cat per day. The constraints to the design are that there is limited space to orient the motor and where the lid revolves to when the door is open. When a cat comes near the lid, the door will open slowly, to not frighten the cat and then close when the cat leaves. The lid needs to rotate slow enough that it does not scare the cat away when it opens. To ensure the safety of the cat, there is a built-in emergency release mechanism in case the door jams or the cat gets its head stuck.

**6.5.1 Design**

Since there was limited space under the bowl and in the container, the electronics, wires, sensors, and mechanical components were difficult to find the space for without being too far away from the Raspberry Pi. Since all the components were required to fit in the box, coming up with creative solutions for the placement of sensors and mechanisms was necessary. Since the load cell and the wiring of the RFID circuit were placed under the bowl **(Figure 31**), the motor could not be placed under the bowl and power the lid directly from a shaft.

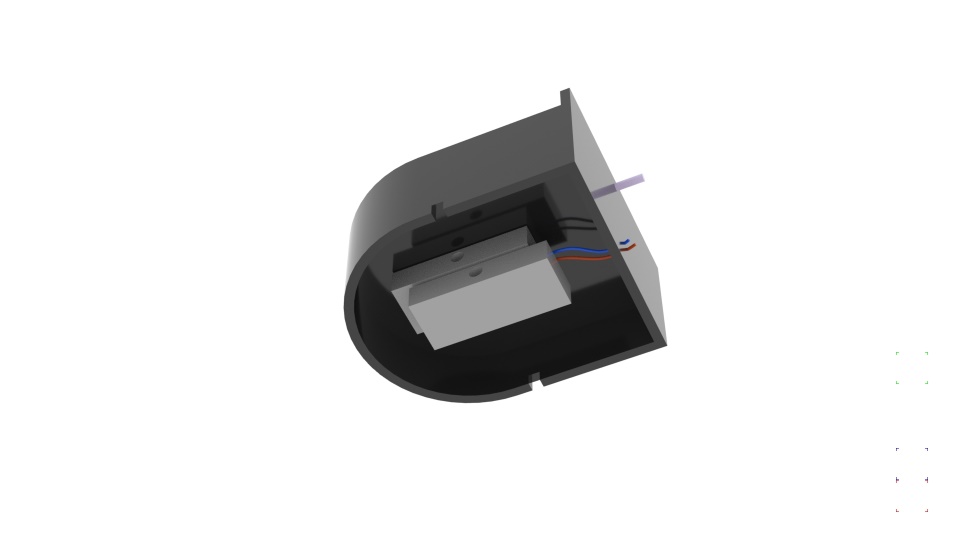


Figure 31: Bottom side of bowl with load cell

Therefore, the mechanism to open the door was constructed by using a motor that rotates the lid from a distance with the use of a 3D Printed shaft attached to a gear. This mechanism was modeled and 3D printed with a biodegradable material made from corn called Polylactic Acid (PLA). This gear then interacts with the gear mounted on the top of the lid. Since there are many types of gears used for a plethora of actions, it was important for product to incorporate one that worked well with the limited space. From benchmarking we found many options for gears such as spur gears, helical gears, gear rack, bevel gears, and worm gears. We chose bevel gears as they are the most appropriate solution to the problem of limited space. Bevel gears allow the power source, in this case the motor, to change the operating angle of the shaft and the lid. They also allow changes in mechanical advantage. Mechanical advantage is a measure of the force amplification achieved by using a mechanical system that involves mechanical components. While the mechanical system preserves the input power, and exchanges the power to output force, by adjusting the ratio of teeth on the gears, the rotational speed and torque of the output gear can be changed in relation to the input gear. The total power stays the same, but the angular speed can be sacrificed in exchange for torque, or in other words, the angular speed decreases as the torque increases. While the advantages of using a pair of bevel gears supports the subsystem design, the disadvantages also needed to be taken into consideration. The constraints and problems of using bevel gears are that bevel gears must be precisely mounted, the shafts must support a combination of normal and shear stresses, and only one gear can be used to power its complementary gear.

One of the customer requirements was that the lid needed to be safe for cat. The step motor used for the product contained a gear box that output 0.375 kg\*cm, meaning that it was designed to operate at low

Furthermore, the bowl was designed to hold the shaft and the motor by using a metal threaded insert allowing the motor to be firmly screwed into the shaft and onto the bowl. The motor, which rotates up to 100 rpms at 12V, has its angular speed controlled by the Raspberry Pi. By calculating the gear ratio of the gears on the shaft and lid, the rotation of the lid can be accurately controlled.

After designing the bevel gear using Autodesk Inventor, the best way to make them was to 3D print. There are many factors that go into 3D printing such as material, infill density, shell, and orientation of the print. The small gear mounted on the lid had a diameter of 0.35 in, while the larger gear mounted to the shaft had a diameter of 0.7 in. They were both relatively small to print and required a significant amount of detail to function. With the technology we had at our disposal, 3D printing allowed us to print with such precision and detail but costed more than necessary for prototyping. Using only ordinary FDM printers, we were able to reduce the nozzle extruding speed and nozzle moving speed thereby allowing the extruders to move at a much slower speed, capable of printing smaller parts.

To save cost and time, I printed a pair of low infill bevel gears first, tested, and adjusted the angle of the teeth and thickness of the shaft. Then we were able to print the solid shaft, gear, and the lid. We then encountered an issue, that

This RFID detection subsystem works in conjunction with this subsystem in that it will send a signal to the Door control subsystem when a registered cat is detected. The Raspberry Pi will receive this data from the RFID scanner and will allow the door to open when a registered cat is within proximity of the reader. Since the system had moving mechanical parts, special safety considerations had to be implemented. To ensure the safety of the cat, the gears were designed to slip if an opposing torque was greater than the torque applied by the motor. A limit switch was used to detect when the door is closed t stop the motor. In software, this translates to the system running the door motor while the switch is not pressed. This subsystem was tested using a python script installed on the microcontroller. This script is attached in **Appendix J**. Testing was essential in this subsystem.

### **6.5.2 Testing & Analysis**

To test the revolving lid door mechanism, we originally believed all we needed was to use some sort of pulley system attached to the motor, however, after benchmarking this concept was thrown out and replaced with a bevel gear design. In the testing phase of this subsystem there were many issues that arose. Before testing we assumed that to close the door a timer would be used and tested to find out the total time it takes to close the door. However, one team member suggested to use limit switches to ensure the door would close properly and allow for the safety of the cats. With this addition, the way in which the team tested the door was by running the motor for an amount of time or until the limit switch was engaged. If the time limit ran out, the motor would stop and wait 2 seconds then attempt again. This safety feature was thoroughly tested, this however presented another problem. Due to the force applied in the opposite direction that the motor was running, the gears would slip which is to be expected, but this caused the gear wear down. This caused the gears to slip when the door ran into the limit switch and would not engage it. To correct this issue, we continued to test and eventually scrapped the idea of the limit switch on the closing action of the door. After testing for a while without the function, it seemed to correct itself. We determined the cause of the issue to be the placement of the bowl in the box. Since the bowl is not attached directly to the box, it tends to shift after removing it from the box. Furthermore, the correct time and speed were determined after days of testing back and forth by ensuring the motor would not run too long after engaging the limit.

## 6.6 Subsystem 6: Food Weight Management

The weight scale attached to the underside of the bowl will consist of a strain-sensor type load cell connected to an HX711 chip connected to the Raspberry Pi. This combination has been chosen because the HX711 uses a synchronous data transmission interface which does not clash with the serial interface of the RFID subsystem. This subsystem was tested using a python script, attached in **Appendix F,** installed on the microcontroller. Through testing, the subsystem produced an accurate result. the weight sensor returned weight values within 5 grams of the test weights’ weight. The test procedure is outlined below:

1. Connect to the raspberry pi through SSH
2. Run the testScale.py script
3. Place a known weight onto the scale
4. Compare the actual and recorded values to see if they match.

We designed the code to default to a value of 40 grams per cat per day according to the requirements suggested by Purina cat food for an average cat.

The Food dispenser will control the amount of food that is released into the bowl. The amount of food in the bowl will be measured by a weight sensor and will release food when the bowl is below a certain weight. Possible challenges could be the dry food getting stuck in the dispenser and detachability for cleaning. This subsystem will accurately and precisely supply food for cats.

In order to control portion sizes for the cats, the team will be employing a load cell with an Analog-to-Digital Converter (A/D Converter) used to read the weight of the food. Using the following simple equation, we will be able to determine how much the cat has eaten for the current session:

= The initial weight of the food

= The final weight of the food

This data will be stored with the cat’s unique ID number taken from either the microchip or collar tag and will limit the amount of food eaten by the cat by subtracting the from the daily portion. Furthermore, the customer will get a detailed correlation mapping of the times in which their pet eats throughout the day, for those curious about their pet’s activity, further strengthening the bond between the customer and their pets.

The load cell will also be used in conjunction with a 9 Volt Motor, which will be employed to refill the bowl when the weight of the food reduces below a specified minimum weight. This specified minimum weight will be determined with further testing of the subsystem. The way in which this will occur is with a 3D-Printed gear made from Polylactic Acid (PLA) that will be created to administer an accurate amount of food. After the gear allows food to be administered to the bowl, the load cell will be used to fill food until the maximum allowed food is reached. Again, the maximum allowed food level will be determined during testing.

## 6.7 Subsystem 7: Data Management & Transfer

The web-based user interface interacts with the hardware system via two sub-systems: data conversion and data transfer. The data conversion system interpreters the analog signal generated by hardware sensors and records them in human-readable form. The data transfer system bridges the hardware system and the server that hosts the web-based interface by acting as a communication channel. The entire system runs on an update cycle once it is booted up.

A Raspberry Pi microcontroller running a Linux-based operating system is chosen for its low processing upkeep in order to host the above-mentioned subsystems (Raspberry Pi Foundation, 2012). Compared to the Windows operating system, Linux Operating Systems demand very little computing resources in order to function and can be tailored to specific needs by modifying its kernel and file system. This allows the operating system to be installed on a machine with ARM architecture, with bare minimum processing power and storage to reduce its physical dimension. Since flash memory in the form of a high capacity SD memory card is the sole storage unit in the Raspberry Pi, a custom variant of Linux is needed. The Linux operating system is optimized in the following ways: it uses ext4 to improve system stability as well as speed when running on flash memory. It also has a modified kernel that reduces excess read/write to the flash memory in exchange for its longevity. Moreover, the system is configured to only have a CLI (command line interface), further reducing the strain on both the storage flash memory and the Raspberry Pi itself. With the above preparation, the system can have the same functionality compared to a traditional PC while drawing two orders of magnitude less power (~1.0 W for Raspberry Pi compare to ~200 W for a typical PC) and sized two orders of magnitude smaller (~0.1 liter for Raspberry Pi compare to ~40 liter for typical PC). The data conversion subsystem is a conjuncture of three parts: the input data from sensors, the conversion algorithm, and the output data file.

The input data from sensors contains the time of which the sensor is triggered (a cat eating from the feeder), the identifier of the trigger in the form of RFID (which of the cats ate from the feeder), and the amount of food ate for each trigger. The input data is constructed in the format of a dictionary where RFID is the key. For each RFID, the dictionary has a string of cat's name, an array of time stamps, an array of food amount ate at each time stamp, and an integer of allowance for the cat. Each Time stamp is formatted by yyyy-ddd-hh-mm-ss. Below is an

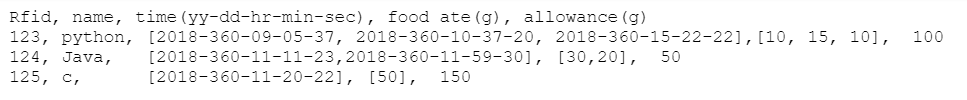


Figure 32: Sensor Data Example

The input data from sensors contains the time of which the sensor is triggered (a cat eating from the feeder), the identifier of the trigger in the form of RFID (which of the cats ate from the feeder), and the amount of food ate for each trigger. The input data is passed through Linux to the conversion algorithm. The conversion algorithm converts the input data into a data structure that associates each RFID with the name of the cat wearing it, as well as its food allowance for the day, the time in the day it eats from the feeder, and the amount of food each time. The algorithm is written in Python, because of the small amount of data and size of the data structure, allowed for less hassle without compromising functionality. While C++ provides stronger performance and more rigid low-level optimization, Python was chosen because it provides a more accessible maintenance structure, while maintaining versatile functionality. Allowing for more the possibility to easily modify and add feature upgrades to the code. The output file extrapolates the data structure into a human-readable CSV (comma-separated-value) file(Rossum, 2006). The entire process is automated so that it runs with a fixed time interval. For this project, the update interval is set to be 80 seconds.

The data transfer subsystem consists of three ingredients: the source, the target, and the medium. The source is the Raspberry Pi, with the output file from the conversion subsystem ready for transmission. The target is a cloud-based storage system connected to the web-based interface that allows asynchronous file access. A medium is needed to move files between the source and the target. Two choices are available on the ARM architecture: the proprietary transfer platform from Dropbox, and the open source platform Rclone. The Dropbox ecosystem offers easier integration, with the ability to automatically synchronize an entire folder across different systems, and an established codebase with minimal setup overhead. On the other hand, the Rclone platform offers much stronger versatility and available to all the major storage protocols but requires extensive knowledge of Linux to setup correctly. Moreover, the Dropbox system limits the permission of the client, and makes it so that only the uploading source has the right to write or modify the said file. Rclone was chosen for this project because of its flexibility and transparency; a piece of code can be adapted to a set of different cloud storage providers with only slight modification, and the source has the authority to restrict or relax read/write permissions on a file to file basis. Rclone is configured to use Google Drive as the target system. Rclone also allows the Raspberry Pi to pull commands from the web-based interface and modify the parameters of the cat feeder such as cat names and food allowances.

The configuration of Rclone is described below. To obtain a copy of the rclone program in a CLI only system, run the following code:

curl https://rclone.org/install.sh | sudo bash

This command downloads a copy of the installer from the rclone website, then pipe it through and run it as the administrator of the system (Rclone, 2018). Then run the command to configure:

rclone config

A list of options will then appear. Choose "n" to create a new "remote". The terminology "remote" refers to the cloud-storage service provider. Input "drive" to use Google Drive as the target system. The program will then prompt to choose the scope of permission for the data. Input "drive.readonly" to allow universal read and write access to the files created by Rclone, as well as accessibility for file management via the google drive website. Input "Y" to use auto-configuration since no advanced features are needed for this project. A browser will appear prompting to login to the google drive account. Choose "N" for team driver and finalize the configuration with a confirmation.

The two subsystems, Data Conversion and Data Transfer work together when performing the transition as new input data is received from the sensors and converted to a human-readable output file. Rclone then transfers the output file to Google drive for the web-based interface to utilize. A script run every 80 seconds, interprets a file containing a series of cat profiles, creates a human-readable output file, and deletes the old output file on Google Drive to avoid duplicates. Next the script waits 10 seconds and uploads the output file to Google Drive; waits an additional 60 seconds for the upload to complete and deletes the local copy of the output file to avoid duplicates from the next cycle; waits another 10 seconds and begin the cycle anew. Finally, in order to start the script on startup a boot file was created in the configuration directory to let the system begin the cycle as soon as it is called:

echo “@./update” >> ~/.config/lxsession/LXDE-pi/autostart

Due to the nature of the project, implementing the codes and settings after physical and hardware parts are completed would be too late. Therefore, the two subsystems are tested and modified on a separate Raspberry Pi 2 to ensure codes and rclone configuration both works. The tested versions of the two subsystems are transferred to the Raspberry Pi 3 later after both physical and embedded hardware parts are finished. The only difference appeared between Raspberry Pi 2 and Raspberry Pi 3 is the method of data transferring through internet where Raspberry Pi 2 requires ethernet cables and Raspberry Pi 3 does not. Otherwise for the subsystems Data Conversion and Data Transfer, only the system used in the device matters for the project and the two devices have the same system and version.

## 6.8 Subsystem 8: Web App

A web app was deployed to allow for user friendly viewing of data from the feeder. The original goal of the web app was to allow the customer to view analytic data about their cat feeder, and to allow the customer to select certain settings for their cat feeder. Some settings desired to control amount of food given per cat, times each cat can eat, how data is displayed on the app, and more. Unfortunately, due to the time constraints of making this prototype, not all of these goals were meant. The web app does successfully display data from the feeder, but it does not allow for any of these user-controlled features. Still, the app is a useful tool for a customer to analyze their cat’s feeding habits. In **Figure 33**, it can be seen that the “Feeder Controls” section was meant to be the user-controllable area that interacted with the feeder.

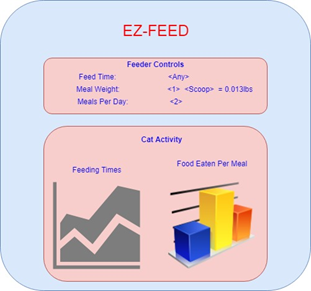


Figure 33 : Rough Sketch-up of the Original Web App Design

For the user interface, the final web app has a simplistic design that displays data collected by the cat feeder. The web app allows for analysis of the feeder’s data, by displaying certain data points on two graphs. The Customer is able to view a line graph that displays cat feeding times. The x-axis represents time of day, and the y-axis represents how much food was eaten by a cat at that time. Different cats are displayed on the same chart with different colors for clarity. The second chart is a bar graph style, and shows how much food each cat has left in their allotment for that day. The way the data is stored is so that each cat is programmed to have an allotment of food to be eaten each day, and the other subsystems are responsible for keeping track of this. When the web app is started, current data can be used to display what each allotment is and how much of that has been eaten by each cat so far. In **Figure 34**, The line chart on the left plots a cat’s feed time and amount. The bar graph on the right shows how much food each cat has remaining to eat.

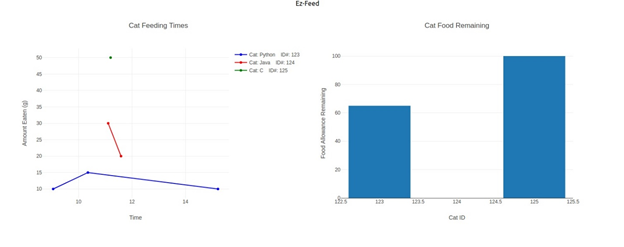


Figure 34: Screenshot of the Web App Display

Table 9 : Example Data That is Displayed in Figure 34

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RFID** | **Name** | **Time ate** | **Amount ate(g)** | **Allowance left(g)** |
| 123 | python | 9:05:37 | 10 | 90 |
| 123 | python | 10:37:20 | 15 | 75 |
| 124 | java | 11:11:23 | 30 | 20 |
| 125 | c | 11:22:22 | 50 | 100 |
| 124 | java | 11:59:30 | 20 | 0 |
| 123 | python | 15:22:22 | 10 | 65 |

As can be seen in **Figure 34**, the web app has a clean and simple view, which allows for data analytics of the cat feeder. However, the goal of allowing the customer to interact with the app and can cat feeder settings, such as the allowance limit for each cat (the rightmost column in **Table 9**, was not met. Again, the reasoning for this failure is primarily the constraint on time. App development can be done quickly, and was for the display side of this project, but for more complex things more time is needed. Since time was split between this subsystem and with **Subsystem 6.1**, the physical container, not all benchmarks were met with the app. One particular obstacle was updating the google drive .csv files that contain data. In theory, they could be edited, so through the app the user could change a cat’s food allowance, for instance. However, implementing this was not completed before the demonstration deadlines. Despite all of this, the web app does correctly display data from the physical cat feeder, and therefore does meet the benchmark of visually displaying data from the cat feeder.

The details of the implementation and development of the app are briefly described here, and was primarily done with JavaScript (JS), a high-level programming language compatible with easy web use. React was used to create the framework and skeleton of the app. React is a GUI based JS plugin that allows for simple design of the front-end (the portion of the web app the user sees). Another JavaScript utility, npm, was used for the package manager of JS. This manager, npm, allows for straightforward development and testing of the app. NodeJS and ExpressJS are two other fundamental JS packages, necessary for launching the web app and developing. A final package, PlotJS, was used to create the charts that the final web app displays. Development of the app was done on Ubuntu Linux, for ease of compatibility issues. Overall, all of these packages are used by the app to create the display that can be seen in **Figure 34**.

While the web app did not meet all benchmarks, primarily because it did not allow the user to control the cat feeder through the app’s GUI, it still successfully meets the other major benchmark of data visualization. The web app allows the customer to view their cat’s mealtimes and amounts eaten for the day, as well as how much allowance each cat has left to consume.

# 7 Results and Discussion

## 7.1 Results

The team performed well at the milestone 2. **Table 10** shows the rubric of the tests performed and the scored achieved. Overall the demonstration went well proving that our design was robust. Although the following tests were the only actions demonstrated in a formal testing environment, the feeder has the capability to perform more operations as are described in **Section 7.2**.

Table 10: Subsystem test

|  |  |  |  |
| --- | --- | --- | --- |
| **Key Function / Feature**  **Metric** | **How It Will Be Demonstrated** | **Target Specification/ Expected Determination** | **Score (4-0)** |
| Ability to withstand the weight of 1 Cat | Place a cat on top of the feeder | ~ 15 lbs | 4 |
| Water Resistance | The outer shell of the cat feeder will have 1 cup of water poured on to the TOP surface of the feeder | The towel inside the funnel will be less than 1/10th of a cup wet | 4 |
| Funnel sturdiness | full feeder | funnel can withstand food | 4 |
| RFID scanner with distance | An RFID Tag will be held at the front portion of the bowl against the lid | Prints out a non-zero tag ID on terminal screen, ¾ times | 4 |
| RFID Differentiation between cats | Place an RFID tag at the front portion of the bowl against the lid, verify that a non-zero tag id number appears on the terminal screen, remove tag. Place second tag at at the front portion of the bowl against the lid, verify that a non-zero tag id number appears on the terminal screen, verify that the second id number is not the same as the first id number. | Tags are different 3 out of 4 times | 4 |
| Load cell | A bag of cat food will be weighed on a kitchen scale then weighed on feeder scale. Verify that the weights are the same. | Weight of the food measured on the kitchen scale will be +/- 10 g from the weight measured on the feeder scale | 4 |
| Door opening | Run motor from the controller, the motor powers the shaft, and the shaft rotates the door, door opens. | Door opens and closes ½ times | 4 |
| Food Dispenser Motor will run | Run motor from the controller, the motor powers the dispenser | the dispenser will rotate 1 portion (60 degrees) | 4 |
| Food will be dispensed | Run motor from the controller, the motor powers the dispenser, and a portion of food is dispensed | Exit food weight 35g +/- 15 g | 4 |
| Visualizing | When web app is started up example cat data will be displayed on a visual graph. | Show visual graph on screen with title, axis units and labels, and data as a line graph using example data file. | 4 |
| Create .csv file from sensor data | A .csv file is created using example cat profile/sensor data. | Show a .csv file is created on the Raspberry Pi and the content is consistent with example data. | 4 |
| Send .csv to Google Drive from Raspberry Pi | An example .csv with the profile data of the cats will be uploaded to google drive. | Show old .csv file is replaced in google drive | 4 |
| Cat Learn mode | A pushbutton in the back of the feeder will be pressed and a new cat profile will be added to the database and printed to the terminal. | The cat profile will be displayed on the terminal | 4 |

## 7.2 Significant Technical **Accomplishments**

### **7.2.1 Attempt to increase RFID scanner range**

Research was carried out to find a way to increase the range of the RFID scanner. This resulted in an attempt to use a 10" by 10" 24-turn square loop coil. This large coil performed extremely badly. Testing the coil was done by placing an RFID tag near it and seeing if the RDM6300 breakout board detected it. The board only detected the tag when it was inside the coil and a few times at random distances that could not be reproduced. Eventually we opted to use the factory-made antenna which was more reliable.

### **7.2.2** **3D Printing**

3D printing is a form of additive manufacturing that is mainly used for rapid prototyping. There are many types of 3D printing, Fused Deposition Modeling, Stereolithography, Digital Light Processing, etc. Each type of 3D printing has its own unique strengths and weaknesses. Fused Deposition Modeling is the most commonly used, creates models by melting plastic filaments, extrudes thing strings of plastic onto a building plate, and stack layers of thin filaments to form the shape desired. Strengths of FDM process are cheap, relatively quick to produce, and ease of use. Weaknesses are orientation biased strength, relatively unstable performance depends on the printer setting, and relatively low-resolution prints.

For FDM printers, there are wide choices of filaments/materials to choose from for printing. Different materials have different properties, such as ABS is the commonly used cheap material for quick prints while have a good strength-weight ratio, TPU is a flexible material that can be used to absorb shocks and bend, PLA is a material derived from corn, which is used to print our project for food safety applications.

In our project, the bowl and the dispenser subsystem are 3D printed because 3D print the models directly from CAD is more reliable efficient than hand making. To achieve optimal performance and save materials, direction of printing is especially important. The orientation of how filaments layered up significantly impacts the strength of the product. I tested multiple 3D printers with different settings to have a smooth and robust product.

7.2.3Laser Cutting

This project required numerous parts that were laser cut. This was done at the Processes Lab as well as the green building. Laser cutting provided accurate and precise cuts. Through help of professors, the acrylic plastic sheet and MDF wood were cut to desire dimensions

**7.2.4 Woodworking Fabrication**

Multiple woodworking techniques and skills were learned while creating the physical container for the feeder, which was primarily done by Lily and Dan Gay. Since the final design called for hardwood construction, precise work had to be done.

First, CAD drawings were made using the already laser cut acrylic pieces’ dimensions. Once the precise size of each piece of wood was determined, RPI’s Fabrication Shop was visited with John Szczesniak. Using the planer there, the board of wood was thinned down to 0.8”, and the jointer was used to flatten the board ends. Besides this, the table saw was used to cut the wood.Titebond III was used to construct the wooden pieces, and many other processes were used to fabricate the box, such as band sawing, routing, and finishing. For a more detailed description, see **Section 6.1.3**.

### **7.2.5 Rclone Configuration on Raspberry Pi**

Rclone platform offers much stronger versatility and available to all the major storage protocols but requires extensive knowledge of Linux to setup correctly. Jimmy Li configured Rclone on Raspberry Pi to use Google Drive as the target system.

**7.2.6 Script and Startup Setting**

Jimmy Li learned to write scripts to run data conversion and updating google drive automatically in the background when triggered. Startup setting was also modified to run the script whenever Raspberry Pi is started.

### **7.2.7 Web App using JavaScript, React, and NodeJS**

Creating the web app was an entirely new skill. Despite many other Computer Science related projects, there was no experience of creating a web app on the team. To do this, an entirely new language was learned, JavaScript, and many new coding utilities were learned. The app was created primarily with React and NodeJS, both of which were being used for the first time by the team.

**7.2.8 Integration**

Brandon, Erik and Lily learned how to integrate subsystems after a long tiresome effort. In the integration test of our product, we were able to seamlessly integrate all the subsystems together. In the test, the first thing would be that the program reads in the existing cat profiles from the database and stores them in a memory-efficient Data Structure called a dictionary. The RFID reader would then listen and wait for a 125 kHz RFID tag to be within range and check to see if the RFID tag matches a cat profile in the dictionary. Each cat associated with the feeder has a cat profile object created for it stores it in the dictionary. If a match is found in the dictionary, the program checks if the cat is able to eat food based on if the cat has food remaining for the day. The food remaining for the day is saved in the Cat’s Profile along with its RFID tag, its name, a list of times the cat has eaten during the day, a list of the amount of food eaten during each feeding session in the day, and its current amount of food left for the day. Another way that the door will open is if the program identifies that the saved day is not the same day as the current system time, it will then update the saved date in the cat profile and refresh the cat’s amount of food left for the day to the original daily allowance value.

At this point, if the cat can eat, the door will open. In order to properly open the door, a limit switch must be engaged, to ensure the door did not stop before opening all the way. If the limit switch is not engaged, it will reattempt to close the door, wait 2 seconds and then try again until the limit is engaged. After the limit is engaged, the initial reading of the scale will take place and it will enter a loop that will run until the cat runs out of food or is no longer detected by the reader. Within the loop the following will take place every iteration, the scale will weigh the bowl and food, update the amount of food left, wait 1 second and listen for an RFID tag. If the RFID tag is not found it will attempt to read it again and if the same tag is not found the door will close. Otherwise the loop will continue to the next iteration.

After the loop ends, the scale will read one last time and update the cat profile object with the amount of food left for the cat. This necessary precaution will ensure that if the cat presses down on the scale and causes the door to close, a reading will take place after this fault to ensure an accurate reading is made. For the purpose of accuracy, every time the scale reads a weight, the program averages the weight of 5 successive attempts and uses the average as the weight. Furthermore, the current system time will be appended to the eating times list in the cat’s profile and add the amount of food eaten to the amount of food eaten list. The door of the feeder will then close.

In the case that a non-zero RFID tag is read by the reader, is not found in the dictionary and is accompanied by the press of the learn mode push button, then a cat profile will be created for the cat. This cat profile will be added to the dictionary of cat profiles.

Next the program will wait 2 seconds to ensure that if none of the above if-statements were run that the program does not try to re-read a tag too quickly. Then the program will export the contents of the dictionary to a .CSV file to ensure that if the program crashes, there will be a save point. The program will then check to see if the bowl contains less than 60 grams of food. If the condition is met, the feeder will run and fill the bowl until the bowl contains 60 grams of food. The program will then start over and listen for an RFID tag.

# 8 Conclusions

8.1 General Technical Conclusions

The task of designing, building and testing an automatic cat feeding device has proven to be both challenging and engaging. As shown in section 7, the finished prototype achieved the goals specified in the demonstration rubrics, as well as its final goal of feeding a real cat.

8.2 Possible future improvements

For this project to reach the commercial market, there are several areas for improvement that can make this device more accommodating to the consumer experience. Some of the additional features include but are not limited to the following:

**8.2.1 Bigger food bowl**

During the integration testing phase, the cat was encouraged to try eating from the bowl. Although it successful, the cat was forced to push its face up against the bowl and revolving door in order to eat from the bowl.



Figure 35: Cat eating from bowl

As seen in **Figure 35**, the cat had a difficult time trying to eat from the bowl. This uncomfortable eating style is attributed to the small available space that theq1 cat has between the bowl and the door. One way of making it easier for the cat to consume food, is by extending the bowl further out from the wall. This would include a significant redesign of the bowl to elongate the outward extending radius, 3D printing the bowl and doors again.

**8.2.2 Improved food replenishment mechanism**

Although the stalling problem in the “cereal dispenser” mechanism was fixed, the dispenser would still jam occasionally. Additionally, the declined slope in the back of the bowl was not steep enough to allow food to reach the front. To fix this issue, the bowl needs to be redesigned with a slope that will allow the food to roll forward, closer to the cat.

**8.2.3 Portability**

Currently, the scale is positioned directly under the bowl as to allow for the weight of the bowl and food to be directed onto the scale. However, the bowl is not connected to the bowl itself, this causes the system to dismember with the slightest attempt to move the device. A simple fix would be to add a base plate and attach the bowl-scale system onto it.

**8.2.4 Improved User Interaction & Preferences**

With limited time, the feature allowing the customer to receive notifications from the web app, was omitted. Additionally, the ability to update cat preferences was also deemed not possible. For these features to be included, we would need to develop a means of accepting user input from the web interface. This could be achieved in the form of buttons and text fields. Although, Industrial Design was a factor in our overall product design, we prioritized the Engineering design aspects as the goal was to develop a mostly-working prototype.

**8.2.5 RFID Reader Scanning Range**

The team attempted to create a 4-inch reading range for the RFID reader to accept tags, however, this attempt failed. As was addressed in the RFID section above, there were many factors that may have caused this issue. Due to this failed attempt we do not have any recommendations on how to create a working RFID antenna from magnet wire. An alternative solution to our trial and error attempt would be to buy a higher end reader that can extend the range.

# 9 References

Rclone. (2018, August 7). *CRC Wiki,* . Retrieved 22:31, December 10, 2018 from

https://wiki.crc.nd.edu/w/index.php?title=Rclone&oldid=10271.

Gareth Halfacree and Eben Upton. (2012). *Raspberry Pi User Guide* (1st ed.). Wiley Publishing.

Raspberry Pi Foundation. (2012). *Raspberry Pi Documentation.* Retrieved December 10, 2018

from https://www.raspberrypi.org/documentation/

Rossum, G. V. (2006, October 18). Python Library Reference. Retrieved December 10, 2018,

from https://docs.python.org/2.4/lib/lib.html

*Association for Pet Obesity Prevention*, Association for Pet Obesity Prevention, 2018,

petobesityprevention.org/.

Hugo, Frank, and Rocco J Logozzo. *Automatic Pet Feeder*. 12 Sept. 1967.

Lee, Ji-Hong. *Household Pet Internet Food Feeder of an Integrated Type and an Internet*

*Food Supply System Using It*.

Lee, Y., Ph.D. (2003). Antenna Circuit Design for RFID Applications. Retrieved October 12, 2018, from http://ww1.microchip.com/downloads/en/AppNotes/00710c.pdf

Lseebiz. “Automatic Pet Feeder, Dogs Cats Food Dispenser with Voice Record Remind,

Timer Programmable, Portion Control, Distribution Alarm, IR Detect, 4 Meals a Day for (Medium & Small) Dogs Cats.” *Amazon.ca*, Amazon, a.co/d/9FkVJrS.

“Microchipping of Animals.” *Avma.org*, American Veterinary Medical Association,

www.avma.org/KB/Resources/Reference/Pages/Microchipping-of-Animals-Backgrounder.aspx.

PetSafe. “Automatic Food and Crisp Dispenser Eatwell 5 Animal Feed - Programmable – LCD

Screen - BPA Free.” *Amazon.fr*, Amazon, amzn.eu/d/00gOwEK.

PetSafe. “PetSafe Smart Feed Automatic Dog and Cat Feeder, Smartphone, 24-Cups, Wi-Fi

Enabled App for IPhone and Android, Award Winning Pet Feeder.” *Amazon.com*,

Amazon, a.co/d/gN4fmEP.

Roland Hill, Nicholas Patrick. *Pet Feeders*. 28 July 2016.

Stewart, William. *In-Home Pet Feeding and Monitoring System*. 5 Sept. 2017.

SureFeed Microchip Pet Feeder. (n.d.). Retrieved from https://s3-eu-west-1.amazonaws.com/sureflap/website/support/manuals/1/multi-lang/manual.pdf

“U.S. Pet Ownership Statistics.” *Avma.org*, American Veterinary Medical Association,

2012, www.avma.org/KB/Resources/Statistics/Pages/Market-research-statistics-US-pet-

ownership.aspx.

10 Appendices

## 10.1 Appendix A: Selection of Team Project

This project was created because one of the members has two cats. They were having the issue of not being able to control the amount each cat eats.

## 10.2 Appendix B: Customer Requirements and Technical Specifications

The initial research in customer requirement began with a survey that was sent out through Facebook. From this survey, the team data from real world customers. From this data the customer requirements were developed. **Figures 36 – 39** show the data from which the team focused on.

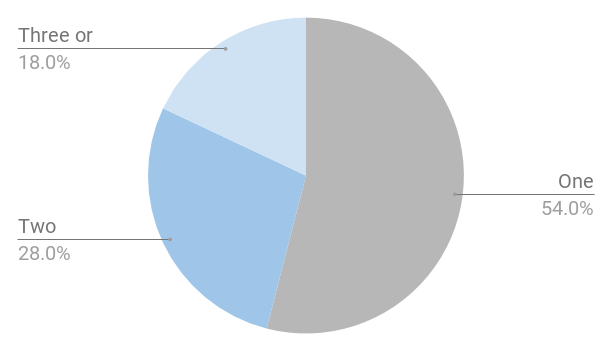


Figure 36: How Many Cats Does the surveyor own?

It is important to know how many cats the customer has because the feeder should be able to work on more than just one cat. From this data more than half of the people had only one cat so this shoes that there would still be a big need even if it worked for only one cat

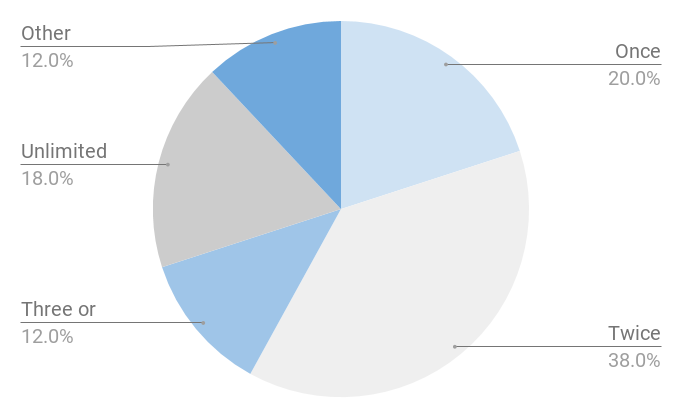


Figure 37: How Many Times a Day does the Surveyor Feed their Cat(s)?

From the data in **Figure 37**, more than half of people normally feed their cats once or twice a day. This also supports the need for the EZ feeder

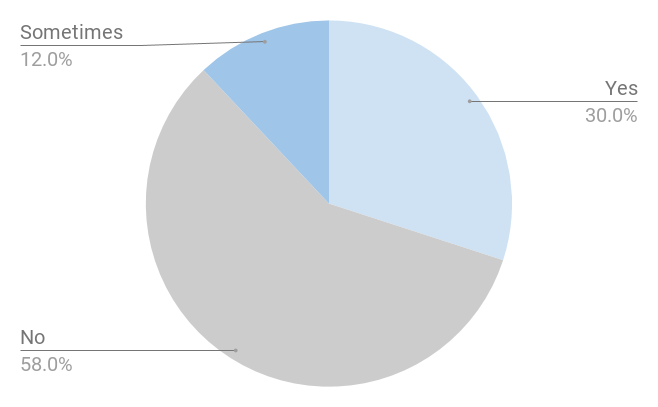


Figure 38: Does the Surveyor’s Cat Wear a Collar?

One of the biggest concerns of the group was how to recognize a cat. If a cat wears a collar, then the customer could easily attach the RFID chip to the collar. This then raised the question, how many cats wear collars? From this data there is still a thirty percent population of people who have their cats wear collars. This was a great enough percentage to continue with the project

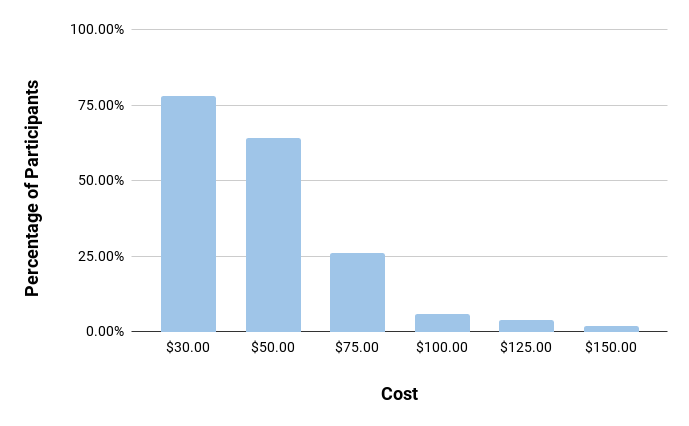


Figure 39: How much would the Surveyor Pay for the EZ Feeder

An important question to ask when creating a project is how profitable will it be. Thinking about this it was important to see how much people would be willing to pay for the EZ Feeder. From the data in **Figure 39**, most people would be willing to pay around seventy-five dollars. This is where the target value for the cost came from

Table 11 : Customer Requirements and Technical Specifications

|  |  |  |
| --- | --- | --- |
| Customer Needs | Technical Specifications | Target Values |
| Does not require a lot of power | Voltage is minimum | < 24 volts |
| Can go days without feeing | Larger food storage | 1 gallon containment unit |
| Durable | Can withstand weight of cat | Able to withstand 90 N of downward force |
| Affordable | Cost is minimal | < $75 |
| Compact | Size | < 24”x24”x2” |
| Portable | Lightweight | < 20 lbs |

From the customer requirement many technical specifications were developed then put into **Table 12**.

Table 12: Technical Specifications

|  |  |  |
| --- | --- | --- |
| Technical Specifications | Target Value | Design value |
| Minimum voltage | <24 Volt | The device will operate on 9v power provided by an AC/DC adapter |
| Large food storage | 1 gallon containment unit | A 5”x6”x11” container will be used to hold 1.6 gallons of food. |
| Can withstand weight of cat | Able to withstand 200N downward force | Wooden frame will provide more than 200N resisting force to weights supported on top of the container |
| Minimal Cost | <$75 | Prototype cost: $189.27  Expected to be <$75 if mass produced |
| Small Size | <24’’x24’’x24’’ | 12’’ x 6’’ x 18’’ |
| Portable | <20lbs | Final weight without food: 20 lbs |

## 

## 10.3 Appendix C: Gantt Chart

To keep track of the team’s progress, a Gantt chart is used. It is split into 3 sections: Documentation, Design and Build. In the documentation is the meeting minutes and all of the IED deadlines. In the design section is the projects brainstorming and concept selection. This also includes the design and material lists. The last section is physically building the product with the end goal of testing and integration. Throughout this project, the Gantt chart was updated many times. Because we had to redesign the physical portion of the feeder we had to add more sections and keep track of the time remaining in the semester.

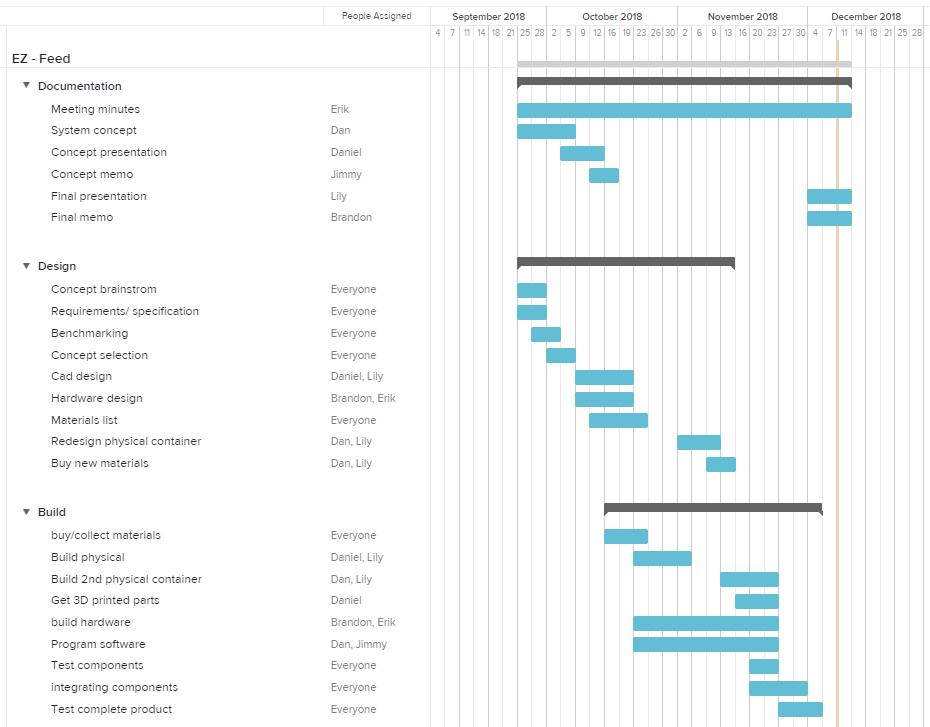


Figure 40: Gantt Chart

Not only did the Gantt chart help the team stay on track but there is also a calendar that displayed when the project started, Milestone 1 with presentation and memo, milestone 2, and milestone 3.

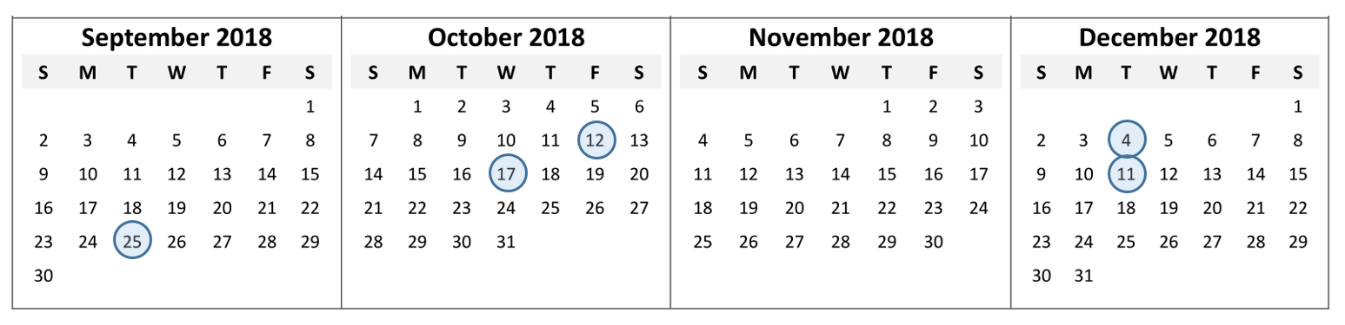


Figure 41: Four month Calendar view

## 10.4 Appendix D: Expense Report

Table 13: Project Expenses

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Item | Quantity | Unit Price | Subtotal | Shipping | Total |
| RFID Reader | 2 | $6.00 | $12.00 | $0 | $12.00 |
| Load Cell | 1 | $9.98 | $9.98 | $0 | $9.98 |
| ¼” x 2’ x 4’ MDF Board | 1 | $7.94 | $7.94 | n/a | $7.94 |
| ¼” x 12” x 12” Acrylic Sheet | 1 | $14.99 | $14.99 | n/a | $14.99 |
| #8 ½” Pan Head Screws 1/2lb. box | 1 | $4.86 | $4.86 | n/a | $4.86 |
| 1”x 15” x 4’ Maple Board | 1 | $32.80 | $32.80 | n/a | $32.80 |
| 3D Printed Gears | 4 | $2.45 | $9.81 | n/a | $9.81 |
| 3D Printed Gear Shafts | 2 | $0.50 | $1.00 | n/a | $1.00 |
| 3D Printed Dispenser | 2 | $9.00 | $18.00 | n/a | $18.00 |
| 3D Printed Geared Motor | 1 | $14.59 | $14.59 | n/a | $14.59 |
| 3D Printed Bowl Prototype | 2 | $10.00 | $10.00 | n/a | $10.00 |
| 3D Printed Bowl Lid | 1 | $6.50 | $6.50 | n/a | $6.50 |
| Copper Wire for Magnet | 1 | $11.40 | $11.40 | $0.00 | $11.40 |
| RFID Tags (Pack of 25) | 1 | $8.86 | $8.86 | $0.00 | $8.86 |
| Limit Switch | 5 | $1.39 | $6.95 | $0.00 | $6.95 |
| Food Grade Resin | 1 | $5.00 | $5.00 | n/a | $5.00 |
| Motor | 1 | $14.59 | $14.59 | $0.00 | $14.59 |
| Grand Total |  |  |  |  | $189.27 |

## 10.5 Appendix E: Team Members and Their Contributions

### **10.5.1 Team Member 1: Lily O’Halloran**

Bought materials as well as helped fabricate the physical container. Provided NX models for the physical container that gave the correct dimensions. Over saw the progress of overall project. Took part in integrating all of the subsystems before milestone 2

### **10.5.2 Team Member 2: Brandon Herrada**

Wrote the Cat Profile class to make for easy accessibility in integration. Wrote the code for the integration test in which a data structure containing cat profiles was used to help associate sensor data with the identity of a specific cat. Led hardware team in defining the electronics needed, assembly of the electronics and antenna circuit as well as software development of some of the demonstration unit test code. Led the integration of the subsystems

### **10.5.3 Team Member 3: Daniel Schwank**

Created initial design and brainstorming. Started working on mechanical components such as the food dispenser and initial design for the bowl. Purchased motors, 3D printed prototypes with low infill to test the size and the basic functionality.

### **10.5.4 Team Member 4: Periklis Sophocleous**

Provided the framework on which the final code was written. This framework includes driver scripts to interface between python code and the RFID scanner, as well as the weight scale. Assembled the weight scale. Helped in debugging the final code that was used to run the demonstration tests. Took part in assembling and troubleshooting the mechanical and electronic systems. Software development of some of the demonstration unit test code.

### **10.5.5 Team Member 5: Daniel Gay**

Purchased materials for the physical container. Constructed the physical container after the initial wood-cutting step. Created and wrote the data visualization Web App. Helped to assemble and integrate the physical container with other subsystems.

### **10.6.6 Team Member 6: Jimmy Li**

I developed the interface between the cat feeder and web-based interface as well as the interpreter for sensor data. The interface bridges the two platforms by uploading the input data from the physical system to Google Drive as well as downloading instruction from the web interface to the physical system. The interpreter takes in data generated by the physical system and convert it to a machine-friendly file type for plotting in the web interface. I also drew electronics schematic diagrams for the report. In addition, I also setup a Linux system in Raspberry Pi and provided tools such as power adapters and soldering sets for the construction of the physical system.

## **10.6 Appendix F: Statement of Work**

Team: Lily O’Halloran, Brandon Herrada, Daniel Gay, Daniel Schwank, Jimmy Li, Periklis Sophocleous

Semester’s Objectives:

1. Design a working prototype cat feeder
2. build the feeder out of high-quality materials
3. Investigate the most effective way to implement sensing a cat
4. Prototype has user defined pet preferences
5. Feeder can support more than one cat

## **10.7 Appendix G: Professional Development - Lessons Learned**

Through the semester, there were a lot of lessons that everyone learned. Not only did everyone technically improved but also grew professionally. The hardest part about this project was getting six strangers to work together as one team. We all had to learn how different people work and problem solve.

10.8 Appendix H: User Manual

### **10.8.1 Setting up the device**

1. Place the box on the floor, with the food container lid facing up.
2. Pull out the scale module from under the bowl opening
3. Place the bowl on top of the scale module
4. Push the assembled bowl + scale back into the bowl opening.
5. Plug in the device using the included 9v wall adapter

Your device is now set up and ready to fill with food. See the following sections of the user manual for further operating instructions

### **10.8.2 Adding cat collar tags**

Out of the box, the included collar tags are programmed into the system. If you need to add tags that have been purchased separately, follow these steps.

1. Make sure the bowl lid is closed and no other collar tags are near the device.
2. Place the new-collar tag on the front face of the bowl
3. Press the push button on the side of the box to register this collar tag.
4. If successful, the lid should open and close.
5. You can now attach the tag on to your cat’s collar.

### **10.8.3 Everyday operation**

When there is no food in the container:

1. Open the top lid
2. Fill up the container up to the line inscribed on the clear plastic.
3. Close the lid

The EZ-FEED will take it from here. No other interference is necessary by the user for normal use.

WARNING: Do not overfill the food container as this will damage the food dispenser.

### **10.8.4 Cleaning the device**

1. Remove all food from the container and bowl.
2. Take out the bowl by pulling it
3. Clean the bowl using dish soap and water
4. If necessary, clean the food container with dish soap and water.
5. Place the bowl back on the scale and push it into the opening.

### **10.8.5 Track your cat**

1. Open your web browser
2. Go to the website for your cat
3. Plots of most recent history of your cats’ eating time and amount should show up on your screen

WARNING: DO NOT SPRAY WATER ONTO THE DEVICE DIRECTLY. This could damage the electronics and the mechanisms inside the device. Always pat dry all components after cleaning.

### **10.8.6 Troubleshooting**

Table 14: Troubleshooting

|  |  |  |
| --- | --- | --- |
| Problem | Cause | Solution |
| Bowl is not automatically refilling | Food dispenser mechanism blocked | Remove all food from container. The dispenser mechanism should start spinning. Add food slowly until it stops. Now you can refill the container. |
| Lid does not open/close | Food stuck between lid and bowl | Remove the stuck piece of food and try again. |
| Lid does not open/close | Motor failure | Reset the device. If this fails, return to a qualified technician for repair. |
| Food is spilling on the side of the bowl (bowl overfilling) | Weight scale failure | Check that the scale is properly set up and that the bowl resides only on the scale. Check if there is something between the bowl and the floor and remove it. |
| Website does not show most recent plots | Internet not connected | Check to see if your computer is connected to the internet and make sure your google drive is up to date. |

## **10.9 Appendix I: Schematics**

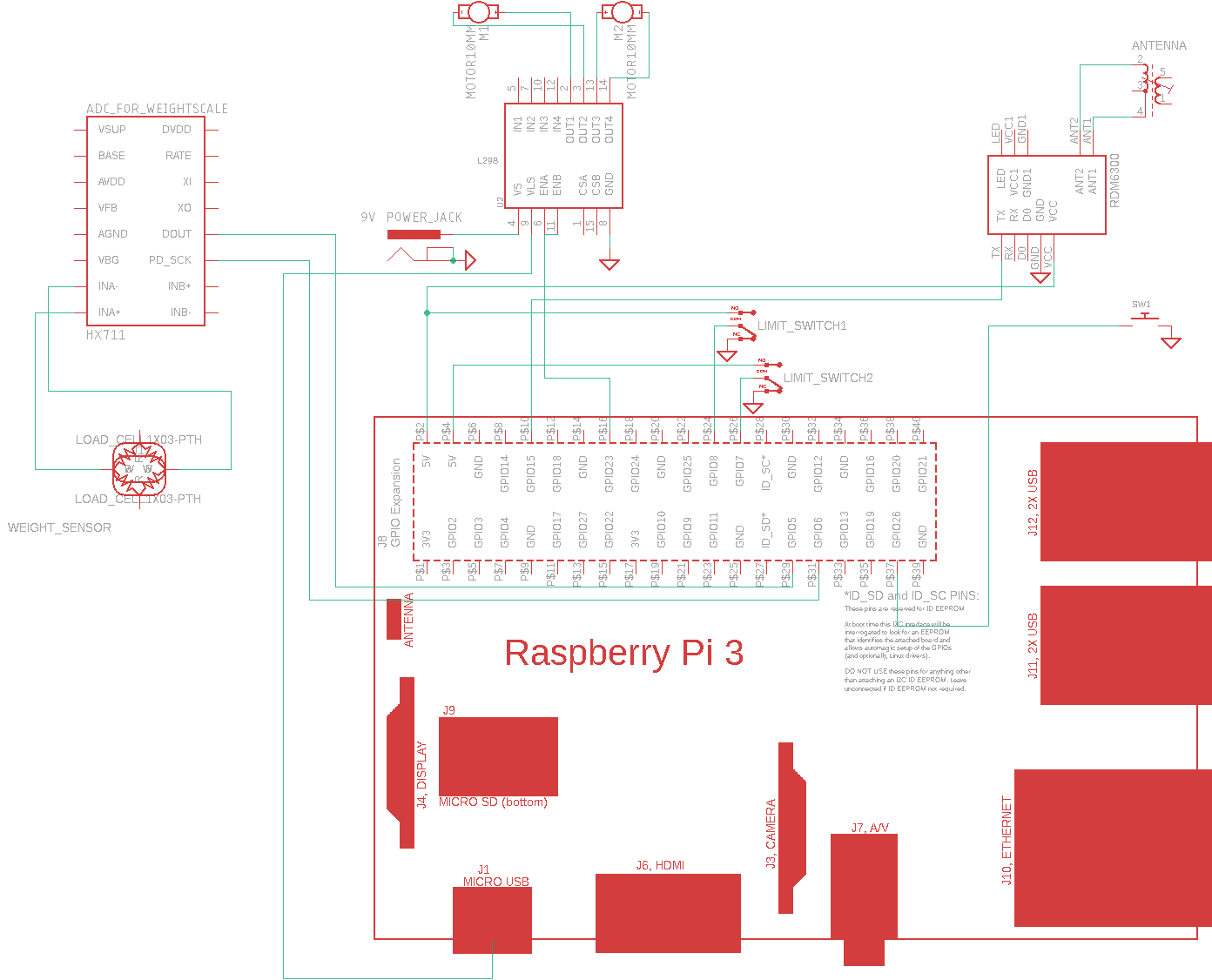


Figure 42 : Circuit diagram

## **10.10 Appendix J: Code**

The following section listed the files that the team created in order to develop a working product. With the significant software and electronic challenge given, it required many unit test functions and much debugging. Writing, testing and debugging was by the most difficult challenge in the software development. Additionally, the code was written in such a way as to provide an easy transition to integration.

### **10.10.1 C**atProfile.py

**import** time

# This cat profile is used to store and manage the cat data. It includes the times that the cat ate, the amount of food

# eaten during every session, the name of the cat, its daily food allowance, the amount of food left and its RFID tag

**class** **CatProfile:**

**def** \_\_init\_\_**(**self**,** rfid**,** name**=**''**,** food\_allowance**=**40**,** food\_left**=**40**,** times**=[],** eaten **=[]):**

self**.**RFID **=** rfid

self**.**Name **=** name

self**.**Daily\_allowance **=** int**(**food\_allowance**)**

self**.**Food\_left **=** int**(**food\_left**)**

self**.**EatingTimes **=** times

self**.**ServingEaten **=** eaten

# This allows the program to understand the current day and is used to check if the cat is able to eat

t **=** time**.**localtime**()**

timestamp **=** time**.**strftime**(**'%Y %j'**,** t**)**

timestamp **=** timestamp**.**split**(**' '**)**

timestamp **=** **[**int**(**i**)** **for** i **in** timestamp**]**

self**.**Day **=** timestamp

self**.**Able\_to\_eat **=** self**.**ableToEat**()**

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

# The following functions are created to allow access outside of the class #

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

# This can be used to change the cat's RFID tag, whether the owner wants to go from microchip to rfid tag or vice

# versa

**def** setRFID **(**self**,** setNewID**):**

self**.**RFID **=** setNewID

# Return the RFID Tag when prompted

**def** getRFID **(**self**):**

**return** self**.**RFID

# Change the daily allowance of the cat, dependent on whether the owner believes their cat should have more food

**def** setDailyAllowance **(**self**,** newAmount**):**

self**.**Daily\_allowance **=** newAmount

# Return the daily allowance

**def** getDailyAllowance **(**self**):**

**return** self**.**Daily\_allowance

# Return how much food the cat has left

**def** getFoodLeft **(**self**):**

**return** self**.**Food\_left

# Used in conjunction with the scale and is called to update how much food the cat has left

**def** setFoodLeft **(**self**,** delta**):**

#Reduce amount of food left for the day based on how much the

#The cat has eaten in the last feeding Session, based on the

#the load cells delta

self**.**Food\_left **=** delta

**return** self**.**Food\_left

# This updates the day to a new day when called

**def** setDay **(**self**,** timestamp**):**

self**.**Day **=** timestamp

self**.**Able\_to\_eat **=** **True**

# Returns the year and day of the year (out of 365)

**def** getDay **(**self**):**

**return** self**.**Day

# Checks the saved day and compares it with the current day to see if the cat can eat again

**def** updateDay**(**self**):**

t **=** time**.**localtime**()**

timestamp **=** time**.**strftime**(**'%Y %j'**,** t**)**

timestamp **=** timestamp**.**split**(**' '**)**

timestamp **=** **[**int**(**i**)** **for** i **in** timestamp**]**

day **=** self**.**getDay**()**

**if** timestamp**[**0**]** **>** day**[**0**]** **or** timestamp**[**1**]** **>** day**[**1**]:**

self**.**setDay**(**timestamp**)**

self**.**Food\_left **=** self**.**Daily\_allowance

**return** **True**

**return** **False**

# Extended user feature to add more food, at the owner's request, for the cat if the food runs out

#def owner\_override(self, amount):

# self.Food\_left += amount

# self.ableToEat()

# This checks to ensure that the day is current and that the cat has food left and if so it will allow the cat

# to eat

**def** ableToEat**(**self**):**

**if** self**.**updateDay**()** **or** self**.**Food\_left **>** 0**:**

self**.**Able\_to\_eat **=** **True**

**else:**

self**.**Able\_to\_eat **=** **False**

**return** self**.**Able\_to\_eat

# Adds the current time to the times list as well as the amount of food eaten to the servings list

**def** appendTime**(**self**,** initial\_food\_left**):**

t **=** time**.**localtime**()**

#Year, Day of the year, (24 format) Hour, Minute, Second

timestamp **=** time**.**strftime**(**'%Y-%j-%H-%M-%S'**,** t**)**

self**.**EatingTimes**.**append**(**timestamp**)**

self**.**ServingEaten**.**append**(**initial\_food\_left**-**self**.**getFoodLeft**())**

# Allows for printing of the class

**def** PrintProfile**(**self**):**

output **=** '{},{},{},{},{}'**.**format**(**self**.**getRFID**(),** self**.**Name**,** self**.**EatingTimes**,** self**.**ServingEaten**,**

self**.**getFoodLeft**())**

**print(**output**)**

# Ouputs a string of the profile, for use with writing to a .csv file

**def** PrintToTxt**(**self**):**

output **=** '{},{},{},{},{}'**.**format**(**self**.**getRFID**(),** self**.**Name**,** self**.**EatingTimes**,** self**.**ServingEaten**,**

self**.**getFoodLeft**())**

**return** output

### **10.10.2 convert\_to\_csv.py**

**import** csv

**import** sys

''' Cat Class

Class created to store specific information of each cat

Parameters: rfid - tag serial

name - cat name

time - eating time stamps

allow - total allowed food amount

ate - amount ate in respect with time stamps

Returns: none

'''

**class** **Cat:**

'''

Constructor - initialize a cat

param: rfid, name

return: none

'''

**def** \_\_init\_\_**(**self**,** rfid**,** name**):**

self**.**rfid **=** rfid

self**.**name **=** name

self**.**time **=** **[]**

self**.**allow **=** 0.0

self**.**ate **=** **[]**

'''

add time stamp to this.time

param: tstamp - array of time stamps

modifies: this.time

returns: none

'''

**def** add\_time**(**self**,** tstamp**):**

self**.**time **=** **[**x **for** x **in** tstamp**]**

'''

add ate food amount to this.ate

param: left - array of ate food amount

modifies: this.ate

returns: none

'''

**def** add\_foot\_ate**(**self**,** left**):**

self**.**ate **=** **[**x **for** x **in** left**]**

'''

change total allowance

param: allow - float(total allowance)

modifies: this.allow

returns: none

'''

**def** change\_allow**(**self**,** allow**):**

self**.**allow **=** allow

#main class

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

### Open sensor data, and output file

inputfile1 **=** sys**.**argv**[**1**]**

outputfile **=** sys**.**argv**[**2**]**

datafile **=** open**(**inputfile1**)**

output **=** open**(**outputfile**,** 'w'**)**

### Read sensor data

inputreader **=** csv**.**reader**(**datafile**,** delimiter**=**','**)**

inputreader**.**next**()** # Skip header

cat **=** **[]**

# Read line by line, store information

**for** row **in** inputreader**:**

**if** len**(**row**)** **==** 0**:**

**continue**

catid **=** str**.**strip**(**row**[**0**])**

name **=** str**.**strip**(**row**[**1**])**

time **=** **[]**

amount **=** **[]**

**for** i **in** range**(**2**,**len**(**row**)-**1**):**

**if** '-' **in** row**[**i**]:** #it is a time stamp

t **=** str**.**strip**(**row**[**i**])**

t **=** t**.**replace**(**"["**,** ''**)**

t **=** t**.**replace**(**"]"**,** ''**)**

t **=** t**.**split**(**'-'**)**

time**.**append**(**t**[**2**]+**'-'**+**t**[**3**]+**'-'**+**t**[**4**])**

**else:** # it is a food amount

a **=** str**.**strip**(**row**[**i**])**

a **=** a**.**replace**(**'['**,** ''**)**

a **=** a**.**replace**(**']'**,** ''**)**

amount**.**append**(**float**(**a**))**

allowance **=** float**(**row**[-**1**])**

c **=** Cat**(**catid**,** name**)**

c**.**change\_allow**(**allowance**)**

c**.**add\_time**(**time**)**

c**.**add\_foot\_ate**(**amount**)**

cat**.**append**(**c**)**

datafile**.**close**()**

### Output to csv

outputstr **=** "#rfid, name, time ate, amount ate (g), allowance left (g)\n"

output**.**write**(**outputstr**)**

**for** c **in** cat**:**

rfid **=** c**.**rfid

name **=** c**.**name

allow **=** c**.**allow

**for** i **in** range**(**len**(**c**.**time**)):**

allow **-=** c**.**ate**[**i**]**

outputstr **=** rfid **+** "," **+** name **+** ","**+** c**.**time**[**i**]+**","**+**str**(**c**.**ate**[**i**])** **+** "," **+** str**(**allow**)+**"\n"

output**.**write**(**outputstr**)**

output**.**close**()**

### **10.10.3 Hx711.py**

**import** RPi**.**GPIO **as** GPIO

**import** time

**import** numpy # sudo apt-get python-numpy

**import** sys

**class** **HX711:**

#==========================================

# Constructor:

# dout: digital input pin

# pd\_sck: digital output pin

#==========================================

**def** \_\_init\_\_**(**self**,** dout**,** pd\_sck**,** gain**=**128**):**

self**.**PD\_SCK **=** pd\_sck

self**.**DOUT **=** dout

# Tell the OS what pins will be used and how

GPIO**.**setmode**(**GPIO**.**BOARD**)**

GPIO**.**setup**(**self**.**PD\_SCK**,** GPIO**.**OUT**)**

GPIO**.**setup**(**self**.**DOUT**,** GPIO**.**IN**)**

# Set constants and variables as defined by manufacturer

self**.**GAIN **=** 0

self**.**REFERENCE\_UNIT **=** 1 # The value returned by the hx711 that corresponds to your reference unit AFTER dividing by the SCALE.

self**.**OFFSET **=** 1

self**.**lastVal **=** long**(**0**)**

self**.**LSByte **=** **[**2**,** **-**1**,** **-**1**]**

self**.**MSByte **=** **[**0**,** 3**,** 1**]**

self**.**MSBit **=** **[**0**,** 8**,** 1**]**

self**.**LSBit **=** **[**7**,** **-**1**,** **-**1**]**

self**.**byte\_format **=** 'LSB'

self**.**bit\_format **=** 'MSB'

self**.**byte\_range\_values **=** self**.**LSByte

self**.**bit\_range\_values **=** self**.**MSBit

self**.**set\_gain**(**gain**)**

time**.**sleep**(**1**)**

#==========================================

# Method is\_ready:

# Checks if the specified input pin DOUT is low

#==========================================

**def** is\_ready**(**self**):**

**return** GPIO**.**input**(**self**.**DOUT**)** **==** 0

# Method set\_gain:

# setter for self.GAIN

**def** set\_gain**(**self**,** gain**):**

**if** gain **is** 128**:**

self**.**GAIN **=** 1

**elif** gain **is** 64**:**

self**.**GAIN **=** 3

**elif** gain **is** 32**:**

self**.**GAIN **=** 2

GPIO**.**output**(**self**.**PD\_SCK**,** **False)**

self**.**read**()**

#==========================================

# Method createBoolList

# creates a list of boolean values

# params:

# size:

# The size of the list to create

#==========================================

**def** createBoolList**(**self**,** size**=**8**):**

ret **=** **[]**

**for** i **in** range**(**size**):**

ret**.**append**(False)**

**return** ret

#==========================================

# Method read:

# Reads a value from the scale

#==========================================

**def** read**(**self**):**

**while** **not** self**.**is\_ready**():**

#print("WAITING")

**pass**

dataBits **=** **[**self**.**createBoolList**(),** self**.**createBoolList**(),** self**.**createBoolList**()]**

dataBytes **=** **[**0x0**]** **\*** 4

**for** j **in** range**(**self**.**byte\_range\_values**[**0**],** self**.**byte\_range\_values**[**1**],** self**.**byte\_range\_values**[**2**]):**

**for** i **in** range**(**self**.**bit\_range\_values**[**0**],** self**.**bit\_range\_values**[**1**],** self**.**bit\_range\_values**[**2**]):**

GPIO**.**output**(**self**.**PD\_SCK**,** **True)**

dataBits**[**j**][**i**]** **=** GPIO**.**input**(**self**.**DOUT**)**

GPIO**.**output**(**self**.**PD\_SCK**,** **False)**

dataBytes**[**j**]** **=** numpy**.**packbits**(**numpy**.**uint8**(**dataBits**[**j**]))**

#set channel and gain factor for next reading

**for** i **in** range**(**self**.**GAIN**):**

GPIO**.**output**(**self**.**PD\_SCK**,** **True)**

GPIO**.**output**(**self**.**PD\_SCK**,** **False)**

#check for all 1

#if all(item is True for item in dataBits[0]):

# return long(self.lastVal)

dataBytes**[**2**]** **^=** 0x80

**return** dataBytes

#==========================================

# Method get\_np\_arr8\_string:

# returns a string formatted as a numpy

# 8-bit number string

#==========================================

**def** get\_np\_arr8\_string**(**self**):**

np\_arr8 **=** self**.**read\_np\_arr8**()**

np\_arr8\_string **=** "["**;**

comma **=** ", "

**for** i **in** range**(**4**):**

**if** i **is** 3**:**

comma **=** ""

np\_arr8\_string **+=** str**(**np\_arr8**[**i**])** **+** comma

np\_arr8\_string **+=** "]"**;**

**return** np\_arr8\_string

#==========================================

# Method read\_np\_arr8:

# Reads value to a numpy

# 8-bit number string

#==========================================

**def** read\_np\_arr8**(**self**):**

dataBytes **=** self**.**read**()**

np\_arr8 **=** numpy**.**uint8**(**dataBytes**)**

**return** np\_arr8

#==========================================

# Method read\_np\_arr8:

# Reads a value to a long integer

#==========================================

**def** read\_long**(**self**):**

np\_arr8 **=** self**.**read\_np\_arr8**()**

np\_arr32 **=** np\_arr8**.**view**(**'uint32'**)**

self**.**lastVal **=** np\_arr32

**return** long**(**self**.**lastVal**)**

#==========================================

# Method read\_average:

# Reads <times> values and returns the average

# params:

# times:

# how many values to read

#

#==========================================

**def** read\_average**(**self**,** times**=**3**):**

values **=** long**(**0**)**

**for** i **in** range**(**times**):**

values **+=** self**.**read\_long**()**

**return** values **/** times

#==========================================

# Method get\_value:

# Reads <times> values and returns the

# average, corrected by the tare OFFSET

# params:

# times:

# how many values to read

#

#==========================================

**def** get\_value**(**self**,** times**=**3**):**

**return** self**.**read\_average**(**times**)** **-** self**.**OFFSET

#==========================================

# Method get\_weight:

# Reads <times> values and returns the

# average, corrected by the tare OFFSET

# and divided by the reference value

# params:

# times:

# how many values to read

#

#==========================================

**def** get\_weight**(**self**,** times**=**3**):**

value **=** self**.**get\_value**(**times**)**

value **=** value **/** self**.**REFERENCE\_UNIT

**return** value

#==========================================

# Method tare:

# Reads <times> values and sets it as

# the offset

# params:

# times:

# how many values to read

#

#==========================================

**def** tare**(**self**,** times**=**15**):**

# Backup REFERENCE\_UNIT value

reference\_unit **=** self**.**REFERENCE\_UNIT

self**.**set\_reference\_unit**(**1**)**

value **=** self**.**read\_average**(**times**)**

self**.**set\_offset**(**value**)**

self**.**set\_reference\_unit**(**reference\_unit**)**

**return** value**;**

#==========================================

# Method set\_reading\_format:

# Specified by the manufacturer.

# Do not edit

#

#==========================================

**def** set\_reading\_format**(**self**,** byte\_format**=**"LSB"**,** bit\_format**=**"MSB"**):**

self**.**byte\_format **=** byte\_format

self**.**bit\_format **=** bit\_format

**if** byte\_format **==** "LSB"**:**

self**.**byte\_range\_values **=** self**.**LSByte

**elif** byte\_format **==** "MSB"**:**

self**.**byte\_range\_values **=** self**.**MSByte

**if** bit\_format **==** "LSB"**:**

self**.**bit\_range\_values **=** self**.**LSBit

**elif** bit\_format **==** "MSB"**:**

self**.**bit\_range\_values **=** self**.**MSBit

# OFFSET setter

**def** set\_offset**(**self**,** offset**):**

self**.**OFFSET **=** offset

# REFERENCE\_UNIT setter

**def** set\_reference\_unit**(**self**,** reference\_unit**):**

self**.**REFERENCE\_UNIT **=** reference\_unit

# HX711 datasheet states that setting the PDA\_CLOCK pin on high for >60 microseconds would power off the chip.

# I used 100 microseconds, just in case.

# I've found it is good practice to reset the hx711 if it wasn't used for more than a few seconds.

**def** power\_down**(**self**):**

GPIO**.**output**(**self**.**PD\_SCK**,** **False)**

GPIO**.**output**(**self**.**PD\_SCK**,** **True)**

time**.**sleep**(**0.0001**)**

**def** power\_up**(**self**):**

GPIO**.**output**(**self**.**PD\_SCK**,** **False)**

time**.**sleep**(**0.0001**)**

**def** reset**(**self**):**

self**.**power\_down**()**

self**.**power\_up**()**

**def** listen**(**self**):**

# Reset the input pins

self**.**reset**()**

# Read a value

w **=** self**.**get\_weight**(**5**)**

**def** cleanAndExit**(**self**):**

**print(**"Cleaning..."**)**

GPIO**.**cleanup**()**

**print(**"Bye!"**)**

sys**.**exit**()**

### **10.10.4 IntegrationTest.py**

**import** RPi**.**GPIO **as** GPIO

**import** time

**import** sys

**from** rdm6300 **import** RDM6300

**import** time

**import** serial

**from** catProfile **import** **\***

**import** convert\_to\_csv

**from** MotorControl **import** **\***

**from** hx711 **import** HX711

# Inputs: Text file reader object

# Outputs: Returns a dictionary composed of cat Profiles

# Function Description: This allows us to read in from a .csv file containing sensor data pertaining to cats and create a dictionary

# of cat profiles that allows us to reference to detect a cat and figure out if it is able to eat or not

**def** MakeProfiles**(**read\_input**):**

catProfiles **=** **{}**

read\_input**.**next**()**

**for** row **in** read\_input**:**

**if** len**(**row**)** **==** 0**:**

**continue**

row **=** row**.**split**(**','**)**

catid **=** str**.**strip**(**row**[**0**])**

name **=** str**.**strip**(**row**[**1**])**

times **=** **[]**

amount **=** **[]**

**for** i **in** range**(**2**,**len**(**row**)-**1**):**

**if** '-' **in** row**[**i**]:**

t **=** str**.**strip**(**row**[**i**])**

t **=** t**.**replace**(**"["**,** ''**)**

t **=** t**.**replace**(**"]"**,** ''**)**

times**.**append**(**t**)**

**else:**

a **=** str**.**strip**(**row**[**i**])**

a **=** a**.**replace**(**'['**,** ''**)**

a **=** a**.**replace**(**']'**,** ''**)**

amount**.**append**(**int**(**a**))**

allowance **=** int**(**row**[-**1**])**

catProfiles**[**catid**]** **=** CatProfile**(**catid**,** name**,** allowance**,** allowance**,** times**,** amount**)**

**return** catProfiles

#Pin numbers

Motor1A **=** 13

Motor1B **=** 15

PWM **=** 16

Motor2A **=** 11

Motor2B **=** 18

limit\_open **=** 24

limit\_closed **=** 26

pushbutton **=** 37

#Initialize Raspberry Pi Pins

GPIO**.**cleanup**()**

GPIO**.**setmode**(**GPIO**.**BOARD**)**

#Inilialize RFID Reader

rdm6300\_reader **=** RDM6300**(**'/dev/serial0'**)**

**print(**"Creating serial connection"**)**

serial\_connection **=** serial**.**Serial**(**rdm6300\_reader**.**getSerial**(),** rdm6300\_reader**.**getBaudrate**())**

**print(**"Serial connection created"**)**

#Iniitialize Motors to Raspberry Pi Pins

# Set motor PWM control to Pin 16 with Frequency of 200 Hz

GPIO**.**setup**(**Motor1A**,** GPIO**.**OUT**)**

GPIO**.**setup**(**Motor1B**,** GPIO**.**OUT**)**

GPIO**.**setup**(**PWM**,** GPIO**.**OUT**)**

GPIO**.**setup**(**Motor2A**,** GPIO**.**OUT**)**

GPIO**.**setup**(**Motor2B**,** GPIO**.**OUT**)**

GPIO**.**output**(**Motor1A**,** GPIO**.**LOW**)**

GPIO**.**output**(**Motor1B**,** GPIO**.**LOW**)**

GPIO**.**output**(**PWM**,** GPIO**.**LOW**)**

GPIO**.**output**(**Motor2A**,** GPIO**.**LOW**)**

GPIO**.**output**(**Motor2B**,** GPIO**.**LOW**)**

Motor\_PWM **=** GPIO**.**PWM**(**PWM**,** 200**)**

GPIO**.**setup**(**limit\_open**,** GPIO**.**IN**)**

GPIO**.**setup**(**limit\_closed**,** GPIO**.**IN**)**

Motor\_PWM**.**start**(**0**)**

#Inilaize Learn Mode Pushbutton

GPIO**.**setup**(**pushbutton**,** GPIO**.**IN**,** pull\_up\_down**=**GPIO**.**PUD\_DOWN**)**

#Inilialize Scale to Pins 29, 31

**print(**"Initializing Scale"**)**

Scale **=** HX711**(**29**,** 31**)**

Scale**.**set\_reading\_format**(**"LSB"**,** "MSB"**)**

#Set Scale Reference Value

Scale**.**set\_reference\_unit**(**390**)**

Scale**.**reset**()**

Scale**.**tare**()**

Scale**.**power\_down**()**

Scale**.**power\_up**()**

#Take initial weight of bowl with food in it

initial\_weight **=** Scale**.**get\_weight**(**5**)**

**print(**"Startup weight:"**+**str**(**initial\_weight**))**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

#sensor\_data.txt is the .csv file we read profiles from

read\_input **=** open**(**'sensor\_data.txt'**)**

#Somehow convert csv database to profiles

catProfiles **=** MakeProfiles**(**read\_input**)**

#Cat profile imported into the dictionary for use in testing and detecting rfid from distributed tags

times **=** **[**'2018-360-10-37-20'**,**'2018-360-10-37-21'**]**

amount **=** **[**50**,**10**]**

catProfiles**[**'1D00278983'**]** **=** CatProfile**(**'1D00278983'**,**'Felix'**,**'50'**,**'40'**,**times**,**amount**)**

#This is used for an extended user feature of adding upload and save times

#uploaded, saved = False, False

#This is the main loop that will run as long as the user does not do a Keyboard Interrupt (CTRL + T)

**while(True):**

**try:**

#Listen for an RFID in range of the scanner

rfid\_tag **=** rdm6300\_reader**.**listen**(**serial\_connection**)**

**print(**"RFID Tag: {}"**.**format**(**rfid\_tag**))**

#Check if there is a cat (0000 means no cat)

**if** rfid\_tag **!=** "000000000000"**:**

#Check to see if the cat

**if(**catProfiles**.**get**(**rfid\_tag**)** **!=** **None** **and** **not** catProfiles**[**rfid\_tag**].**ableToEat**()):**

**print(**"Cat not allowed to eat, door will not open"**)**

**elif** catProfiles**.**get**(**rfid\_tag**)** **!=** **None** **and** catProfiles**[**rfid\_tag**].**ableToEat**():**

cat **=** catProfiles**[**rfid\_tag**]**

#open Food Door

**print(**"RFID Tag Found in database"**)**

**print(**"Open door"**)**

speedControl**(**Motor\_PWM**,** 1**,** 100**,** 'f'**,** 0**)**

Scale**.**power\_down**()**

Scale**.**power\_up**()**

#Read the initial weight on the scale

initial **=** Scale**.**get\_weight**(**5**)**

#Set variable startingFoodLeft to cat's current food left value

startingFoodLeft **=** cat**.**getFoodLeft**()**

**print(**"Starting food left: {}"**.**format**(**startingFoodLeft**))**

**print(**"Scale reads (initial weight):"**+**str**(**initial**)+**".\n Flushing serial input to read tag"**)**

serial\_connection**.**flushInput**()**

rfid\_tag2 **=** rdm6300\_reader**.**listen**(**serial\_connection**)**

**print(**"RFID scanner reads: {}"**.**format**(**rfid\_tag2**))**

#Ensure that the cat is still able to eat by updating the amount of food it has left for the day and

#that it is within range of the scanner

**while(**cat**.**ableToEat**()** **and** cat**.**getRFID**()** **==** rfid\_tag2**):**

**print(**"Cat is within range and has food left for the day"**)**

#Read load cell data and wait until cat has eaten daily allowance or id is no longer read

new\_weight **=** Scale**.**get\_weight**(**5**)**

**print(**"New weight: {}"**.**format**(**new\_weight**))**

**print(**"Setting food left to {}"**.**format**(**startingFoodLeft**-(**initial**-**new\_weight**)))**

cat**.**setFoodLeft**(**startingFoodLeft **-** **(**initial**-**new\_weight**))**

**print(**"foodLeft="**+**str**(**cat**.**getFoodLeft**()))**

#Read once every 1 seconds

time**.**sleep**(**1**)**

serial\_connection**.**flushInput**()**

rfid\_tag2**=**rdm6300\_reader**.**listen**(**serial\_connection**)**

#If scanner cannot read cat id, it will try again and if it doesnt read it will close the door

**if(**rfid\_tag2**!=**cat**.**getRFID**()):**

**print(**"Scanned 0000, trying again in case this was an uncertainty thing"**)**

time**.**sleep**(**1**)**

serial\_connection**.**flushInput**()**

rfid\_tag2 **=** rdm6300\_reader**.**listen**(**serial\_connection**)**

**print(**"RFID scanner reads: {}"**.**format**(**rfid\_tag2**))**

Scale**.**power\_down**()**

Scale**.**power\_up**()**

#Append eating time and food left value

**print(**"Cat is done eating, saving time"**)**

cat**.**setFoodLeft**(**startingFoodLeft**-(**initial**-**Scale**.**get\_weight**(**5**)))**

**print(**"Cat has {} grams of food left for the day"**.**format**(**cat**.**getFoodLeft**()))**

cat**.**appendTime**(**startingFoodLeft**)**

#Close door

**print(**"Closing door"**)**

speedControl**(**Motor\_PWM**,** 1**,** 100**,** 'b'**,** 0**)**

#Extended User Feature: if cat runs out of food upload updated file to drive and alert the owner

#Learn mode is enabled if the push button is pressed and the rfid tag is not already in the dictionary

**elif** catProfiles**.**get**(**rfid\_tag**)** **==** **None** **and** GPIO**.**input**(**pushbutton**):**

catProfiles**[**rfid\_tag**]** **=** CatProfile**(**rfid\_tag**)**

#wait 2 seconds

time**.**sleep**(**2**)**

serial\_connection**.**flushInput**()**

#Update the .csv database file after every loop

outputstr **=** "#RFID, Name, Times Eaten, Amount Eaten (g), Daily Food Allowance Left (g)\n"

out **=** open**(**'sensor\_data.txt'**,** 'w'**)**

out**.**write**(**outputstr**)**

**for** i **in** catProfiles**:**

cat **=** catProfiles**[**i**]**

out**.**write**(**cat**.**PrintToTxt**())**

out**.**write**(**'\n'**)**

out**.**close**()**

#This will in theory upload to the google drive and update the .csv file throughout times of the day

'''

t = time.localtime()

current\_time = time.strftime('%H-%M', t)

#Upload at 5pm, this allows for it to be uploaded if the cat continues to eat more than a minute at the start of the hour

if current\_time[0:2] == '17' and uploaded == False:

#Upload to drive

uploaded = True

#Save to database every hour

if current\_time[3] == '1' or saved == False:

#Save to Database

saved = True

for i in catProfiles:

print(i)

if current\_time[3] == '59' and saved == True:

saved = False

#Verify structure of numbers

if current\_time[0:2] == '00' and uploaded == True:

uploaded = False

'''

#Refill Cat food by testing the weight

**while** **(**Scale**.**get\_weight**(**5**)** **<** **-**60**):**

**print(**Scale**.**get\_weight**(**5**))**

speedControl**(**Motor\_PWM**,** 2**,** 100**,** 'b'**,** 0.2**)**

**if** **(**Scale**.**get\_weight**(**5**)** **>** **-**60**):**

**break**

speedControl**(**Motor\_PWM**,** 2**,** 100**,** 'f'**,** 0.1**)**

#Extended User Feature: update local cat profiles if any changes have been made every hour or so

**except** **(**KeyboardInterrupt**,** SystemExit**):**

**print(**"Keyboard interrupt or system exit called"**)**

serial\_connection**.**close**()**

Scale**.**cleanAndExit**()**

### **10.10.5 LearnMode.py**

**from** catProfile **import** **\***

**import** RPi**.**GPIO **as** GPIO

**from** rdm6300 **import** RDM6300

**import** time

**import** serial

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

# Initialize Raspberry Pi Pins

GPIO**.**cleanup**()**

GPIO**.**setmode**(**GPIO**.**BOARD**)**

GPIO**.**setup**(**35**,** GPIO**.**IN**,** pull\_up\_down**=**GPIO**.**PUD\_DOWN**)**

# Initialize RFID Reader

rdm6300\_reader **=** RDM6300**(**'/dev/serial0'**)**

**print(**"Creating serial connection"**)**

serial\_connection **=** serial**.**Serial**(**rdm6300\_reader**.**getSerial**(),** rdm6300\_reader**.**getBaudrate**())**

**print(**"Serial connection created"**)**

# Create dictionary

catProfiles **=** **{}**

**while** **(True):**

**try:**

# Read curent tag

rfid\_tag **=** rdm6300\_reader**.**listen**(**serial\_connection**)**

**print(**"RFID: {}"**.**format**(**rfid\_tag**))**

# Check if there is a cat is within range(0000 means no cat)

**print(**GPIO**.**input**(**35**))**

**if** rfid\_tag **!=** "000000000000"**:**

# If the pushbutton is pressed and the cat is within range create a profile for the cat

**if** catProfiles**.**get**(**rfid\_tag**)** **==** **None** **and** GPIO**.**input**(**35**):**

**print(**"Cat Profile being created"**)**

catProfiles**[**rfid\_tag**]** **=** CatProfile**(**rfid\_tag**)**

cat **=** catProfiles**[**rfid\_tag**]**

**print(**"Cat profile"**)**

cat**.**PrintProfile**()**

time**.**sleep**(**2**)**

serial\_connection**.**flushInput**()**

**except** **(**KeyboardInterrupt**,** SystemExit**):**

**print(**"Keyboard interrupt or system exit called"**)**

serial\_connection**.**close**()**

### **10.10.6 MotorControl.py**

**import** RPi**.**GPIO **as** GPIO

**import** time

**import** sys

# Raspberry pi pins

Motor1A **=** 13

Motor1B **=** 15

PWM **=** 16

Motor2A **=** 11

Motor2B **=** 18

# Inputs: (Object) PWM motor controller, (Integer) Motor Number (1 or 2), (Integer) PWM value (0-100%),

# (String) direction that we want to go, (Float) time we want it to run

# Outputs: None

# Description: This will allow the motor to run based on what the user inputs into it, it works for both motors and will

# allow for many combinations of actions. It allows for different speeds and the amount of time to run the motor.

**def** speedControl**(**motor**,** motor\_num**,** pwm\_value**,** direction**,** duration**):**

# Set PinA & PinB to respective pin numbers set by motor number

**if** motor\_num **==** 1**:**

PinA **=** Motor1A

PinB **=** Motor1B

**else:**

PinA **=** Motor2A

PinB **=** Motor2B

# If forward assign pins to allow for forward movement and use limit switch with pin 24

**if** direction **==** 'f'**:**

**print(**"Driving motor {} Forward"**.**format**(**motor\_num**))**

GPIO**.**output**(**PinA**,** GPIO**.**HIGH**)**

GPIO**.**output**(**PinB**,** GPIO**.**LOW**)**

GPIO**.**output**(**PWM**,** GPIO**.**HIGH**)**

runtime **=** 2

# If backward assign pins to allow backward movement and use limit switch with pin 26

**elif** direction **==** 'b'**:**

**print(**"Driving motor {} Backward"**.**format**(**motor\_num**))**

limit\_switch **=** 26

GPIO**.**output**(**PinA**,** GPIO**.**LOW**)**

GPIO**.**output**(**PinB**,** GPIO**.**HIGH**)**

GPIO**.**output**(**PWM**,** GPIO**.**HIGH**)**

runtime **=** 2

**else:**

limit\_switch **=** 0

GPIO**.**output**(**PinA**,** GPIO**.**LOW**)**

GPIO**.**output**(**PinB**,** GPIO**.**LOW**)**

GPIO**.**output**(**PWM**,** GPIO**.**LOW**)**

**if** motor\_num **==** 1**:**

limit **=** **False**

motor**.**ChangeDutyCycle**(**pwm\_value**)**

# Run until limit switch is hit or time exceeds estimated time to close door

# Attempt until door closes

**while** **not** limit**:**

start **=** time**.**time**()**

motor**.**ChangeDutyCycle**(**pwm\_value**)**

**print(**"Waiting for door limit to engage"**)**

**while(**time**.**time**()-**start **<** runtime**):**

**if** GPIO**.**input**(**limit\_switch**)** **==** 1**:**

**print(**"Limit Engaged"**)**

limit **=** **True**

**break**

# if door does not close try again after 2 seconds

**print(**"Limit not broken, retrying in 2 seconds"**)**

**if** **not** limit**:**

motor**.**ChangeDutyCycle**(**0**)**

time**.**sleep**(**2**)**

**else:**

**print(**"Running Motor 2"**)**

motor**.**ChangeDutyCycle**(**pwm\_value**)**

#Read scale test by running for .5 second, reading scale then repeating until maximum is reached

time**.**sleep**(**duration**)**

**print(**"Turning off motor"**)**

motor**.**ChangeDutyCycle**(**0**)**

GPIO**.**output**(**PinA**,** GPIO**.**LOW**)**

GPIO**.**output**(**PinB**,** GPIO**.**LOW**)**

GPIO**.**output**(**PWM**,** GPIO**.**LOW**)**

### **10.10.7 RDM6300.py**

**import** time

**import** serial

**class** **RDM6300:**

SERIAL\_PORT **=** ''

BAUDRATE **=** 9600

RDM\_START **=** 2

RDM\_END **=** 3

RDM\_DATA\_SZ **=** 20

**def** \_\_init\_\_**(**self**,** serial\_port**):**

self**.**SERIAL\_PORT **=** serial\_port

**print** **(**"Initialized Serial Port"**)**

*@staticmethod*

**def** \_\_verify\_checksum**(**data**,** checksum**):**

**try:**

result **=** int**(**data**[**0**:**2**],** 16**)** \

**^** int**(**data**[**2**:**4**],** 16**)** \

**^** int**(**data**[**4**:**6**],** 16**)** \

**^** int**(**data**[**6**:**8**],** 16**)** \

**^** int**(**data**[**8**:**10**],** 16**)**

result **=** format**(**result**,** 'x'**)**

**except** ValueError**:**

**return** **False**

**if** result**.**lower**()** **!=** checksum**.**lower**():**

**return** **False**

**return** **True**

*@staticmethod*

**def** \_\_fix\_zeros**(**data**):**

**return** data**.**replace**(**' '**,** '0'**)**

**def** \_\_read\_sequence**(**self**,** serial\_connection**):**

serial\_connection**.**timeout**=**3

tag\_string **=** ''

byte\_read **=** serial\_connection**.**read**()**

# Check timeout

**if** byte\_read**==None** **or** byte\_read**==**""**:**

**return** **False**

# Check first byte

**if** int**(**ord**(**byte\_read**))** **!=** self**.**RDM\_START**:**

**return** **False**

expected\_len **=** 12

**while** expected\_len **is** **not** 0**:**

expected\_len **-=** 1

byte\_read **=** serial\_connection**.**read**()**

**if** int**(**ord**(**byte\_read**))** **==** self**.**RDM\_START**:**

expected\_len **=** 12

**continue**

**if** ord**(**byte\_read**)** **!=** self**.**RDM\_END**:**

tag\_string **+=** chr**(**ord**(**byte\_read**))**

**continue**

**break**

data **=** tag\_string**[**0**:**len**(**tag\_string**)** **-** 2**]**

checksum **=** tag\_string**[**len**(**tag\_string**)** **-** 2**:**len**(**tag\_string**)]**

checksum\_ok **=** self**.**\_\_verify\_checksum**(**data**,** checksum**)**

**if** **not** checksum\_ok**:**

**return** **False**

**return** self**.**\_\_fix\_zeros**(**data**)**

# Return Baudrate (Communication Transfer Rate)

**def** getBaudrate**(**self**):**

**return** self**.**BAUDRATE

# Return Serial port

**def** getSerial**(**self**):**

**return** self**.**SERIAL\_PORT

# Read RFID and return data

**def** listen**(**self**,** serial\_connection**):**

data **=** self**.**\_\_read\_sequence**(**serial\_connection**)**

# check if its an actual RFID string

**if** data **is** **False:**

**print(**"No cat detected"**)**

data **=** "000000000000"

**return** data

**else:**

**return** data

### **10.10.8 TestDoor.py**

**import** RPi**.**GPIO **as** GPIO

**from** time **import** sleep

**import** sys

**import** serial

**from** rdm6300 **import** RDM6300

**from** MotorControl **import** **\***

# Raspberry Pi Pins

Motor1A **=** 13

Motor1B **=** 15

PWM **=** 16

limit\_open **=** 24

limit\_closed **=** 26

# Initialize Raspberry Pi Pins

GPIO**.**cleanup**()**

GPIO**.**setmode**(**GPIO**.**BOARD**)**

GPIO**.**setup**(**Motor1A**,** GPIO**.**OUT**)**

GPIO**.**setup**(**Motor1B**,** GPIO**.**OUT**)**

GPIO**.**setup**(**PWM**,** GPIO**.**OUT**)**

GPIO**.**output**(**Motor1A**,** GPIO**.**LOW**)**

GPIO**.**output**(**Motor1B**,** GPIO**.**LOW**)**

GPIO**.**output**(**PWM**,** GPIO**.**LOW**)**

Motor\_PWM **=** GPIO**.**PWM**(**PWM**,** 200**)**

Motor\_PWM**.**start**(**0**)**

GPIO**.**setup**(**limit\_closed**,** GPIO**.**IN**)**

GPIO**.**setup**(**limit\_open**,** GPIO**.**IN**)**

# Initialize RFID Scanner

rdm6300\_reader **=** RDM6300**(**'/dev/serial0'**)**

**print(**"Creating serial connection"**)**

serial\_connection **=** serial**.**Serial**(**rdm6300\_reader**.**getSerial**(),** rdm6300\_reader**.**getBaudrate**())**

**print(**"Serial connection created"**)**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

# Print current Tag read

**print(**rdm6300\_reader**.**listen**(**serial\_connection**))**

**while(True):**

serial\_connection**.**flushInput**()**

# Save currently read value from RFID reader as rfid\_tag

rfid\_tag **=** rdm6300\_reader**.**listen**(**serial\_connection**)**

# If RFID Tag is nearby

**if** rfid\_tag **!=** "000000000000"**:**

**print(**"Door Opening!"**)**

# Open door

speedControl**(**Motor\_PWM**,**1**,**100**,**'f'**,**0**)**

serial\_connection**.**flushInput**()**

# As long as a non-zero RFID tag is read keep door open

**while** rdm6300\_reader**.**listen**(**serial\_connection**)** **!=** "000000000000"**:**

serial\_connection**.**flushInput**()**

# Close door when tag is no longer read

**print(**"Door Closing!"**)**

speedControl**(**Motor\_PWM**,**1**,**100**,**'b'**,**0**)**

### **10.10.9 TestFeeder.py**

**import** RPi**.**GPIO **as** GPIO

**from** time **import** sleep

**import** sys

**from** hx711 **import** HX711

**from** MotorControl **import** **\***

# Raspberry Pi Pin numbers

Motor2A **=** 11

Motor2B **=** 18

PWM **=** 16

# Initialize raspberry pi pins

GPIO**.**cleanup**()**

GPIO**.**setmode**(**GPIO**.**BOARD**)**

GPIO**.**setup**(**Motor2A**,** GPIO**.**OUT**)**

GPIO**.**setup**(**Motor2B**,** GPIO**.**OUT**)**

GPIO**.**setup**(**PWM**,** GPIO**.**OUT**)**

GPIO**.**output**(**Motor2A**,** GPIO**.**LOW**)**

GPIO**.**output**(**Motor2B**,** GPIO**.**LOW**)**

GPIO**.**output**(**PWM**,** GPIO**.**LOW**)**

# Set pulse width period to 1/100 s

Motor\_PWM **=** GPIO**.**PWM**(**PWM**,** 100**)**

# Set pwm to 0% (Always off) in the duty cycle

Motor\_PWM**.**start**(**0**)**

#Inilialize the scale

**print(**"Initializing Scale"**)**

Scale **=** HX711**(**29**,** 31**)**

Scale**.**set\_reading\_format**(**"LSB"**,** "MSB"**)**

Scale**.**set\_reference\_unit**(**390**)**

Scale**.**reset**()**

Scale**.**tare**()**

Scale**.**power\_down**()**

Scale**.**power\_up**()**

# Initial weight should be 0

initial\_weight **=** Scale**.**get\_weight**(**5**)**

**print(**"Startup weight:"**+**str**(**initial\_weight**))**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

# Run the feeder until the scale reads 22 additional grams have been added

# The purpose of this forward and backward motion is to allow for any food to be dislodged

**while(**Scale**.**get\_weight**(**5**)** **<** 22**):**

**print(**Scale**.**get\_weight**(**5**))**

# Run the feeder motor for 0.2 seconds backward

speedControl**(**Motor\_PWM**,** 2**,** 100**,** 'b'**,** 0.2**)**

# If this released more than 22 grams break out of loop

**if(**Scale**.**get\_weight**(**5**)** **>** 22**):**

**break**

# Run motor forward for 0.1 seconds forward

speedControl**(**Motor\_PWM**,** 2**,** 100**,** 'f'**,** 0.1**)**

# Wait 3 seconds

sleep**(**3**)**

**print(**"Final Food amount added(g): {}"**.**format**(**Scale**.**get\_weight**(**5**)))**

### **10.10.10 TestMotors.py**

**import** RPi**.**GPIO **as** GPIO

**from** time **import** sleep

**import** sys

#Set pinmode to GPIO number

GPIO**.**setmode**(**GPIO**.**BOARD**)**

#motor1\_init

Motor1A **=** 13

Motor1B **=** 15

PWM **=** 16

GPIO**.**setup**(**Motor1A**,** GPIO**.**OUT**)**

GPIO**.**setup**(**Motor1B**,** GPIO**.**OUT**)**

GPIO**.**setup**(**PWM**,** GPIO**.**OUT**)**

#Set Motor PWM to pin with Frequency of 500 Hz

Motor\_PWM **=** GPIO**.**PWM**(**PWM**,** 200**)**

#Set Motor1 to 0

Motor\_PWM**.**start**(**0**)**

GPIO**.**output**(**Motor1A**,** GPIO**.**LOW**)**

GPIO**.**output**(**Motor1B**,** GPIO**.**LOW**)**

GPIO**.**output**(**PWM**,** GPIO**.**LOW**)**

#motor2\_init

Motor2A **=** 11

Motor2B **=** 18

GPIO**.**setup**(**Motor2A**,** GPIO**.**OUT**)**

GPIO**.**setup**(**Motor2B**,** GPIO**.**OUT**)**

GPIO**.**output**(**Motor2A**,** GPIO**.**LOW**)**

GPIO**.**output**(**Motor2B**,** GPIO**.**LOW**)**

**def** speedControl**(**motor**,** motor\_num**,** pwm\_value**,** direction**,** duration**):**

**if** motor\_num **==** 1**:**

PinA **=** Motor1A

PinB **=** Motor1B

**else:**

PinA **=** Motor2A

PinB **=** Motor2B

motor**.**ChangeDutyCycle**(**pwm\_value**)**

**if** direction **==** 'f'**:**

GPIO**.**output**(**PinA**,** GPIO**.**HIGH**)**

GPIO**.**output**(**PinB**,** GPIO**.**LOW**)**

GPIO**.**output**(**PWM**,** GPIO**.**HIGH**)**

**elif** direction **==** 'b'**:**

GPIO**.**output**(**PinA**,** GPIO**.**LOW**)**

GPIO**.**output**(**PinB**,** GPIO**.**HIGH**)**

GPIO**.**output**(**PWM**,** GPIO**.**HIGH**)**

**else:**

GPIO**.**output**(**PinA**,** GPIO**.**LOW**)**

GPIO**.**output**(**PinB**,** GPIO**.**LOW**)**

GPIO**.**output**(**PWM**,** GPIO**.**LOW**)**

sleep**(**duration**)**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

**while(True):**

**print(**"test"**)**

**try:**

speedControl**(**Motor\_PWM**,** 1**,** 100**,** 'b'**,** 0.5**)**

#speedControl(Motor\_PWM, 1, 100, 'f', 0.25)

#speedControl(Motor\_PWM, 1, 50, 'b', 0.5)

#speedControl(Motor2, 2, 15, 'f', 0.5)

#GPIO.output(Motor1B,GPIO.HIGH)

#GPIO.output(Motor1E,GPIO.HIGH)

#GPIO.output(Motor1A,GPIO.HIGH)

#sleep(1)

**except** **(**KeyboardInterrupt**,** SystemExit**):**

GPIO**.**cleanup**()**

**break**

### **10.10.11 TestRFID.py**

#!/bin/python

**import** time

**import** serial

# RDM6300 module

# Serial output:

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# |0x02| DATA(10 ascii chars) | CHECKSUM | 0x03 |

# ===============================================

# params: 96008n1

**class** **RDM6300:**

# Interface constants

SERIAL\_PORT **=** ''

BAUDRATE **=** 9600

# Start byte and end byte constants

RDM\_START **=** 2

RDM\_END **=** 3

# DATA string size

RDM\_DATA\_SZ **=** 10

# Constructor:

#~~~~~~~~~~~~~

# Sets the serial port name

**def** \_\_init\_\_**(**self**,** serial\_port**):**

self**.**SERIAL\_PORT **=** serial\_port

**print** **(**"Initialized Serial Port"**)**

# Static method \_\_verify\_checksum:

#======================================================

# Checks if the data string received is a valid ID

# by XOR each pair of dat and compairing the result

# against the read checksum.

#

# Params:

# data:

# A list of size 10 containing each byte of

# the ID as a 2-digit hexadecimal text string

# e.g ["DE","CA","FF","C0","FF","EE",...]

# checksum:

# The checksum as read from the tag as a

# 2-digit hexadecimal text string e.g. "D3"

# Returns:

# True if checksum is ok

# False if checksum is not ok

#======================================================

*@staticmethod*

**def** \_\_verify\_checksum**(**data**,** checksum**):**

**try:**

# Calculate checksum

result **=** int**(**data**[**0**:**2**],** 16**)** \

**^** int**(**data**[**2**:**4**],** 16**)** \

**^** int**(**data**[**4**:**6**],** 16**)** \

**^** int**(**data**[**6**:**8**],** 16**)** \

**^** int**(**data**[**8**:**10**],** 16**)**

result **=** format**(**result**,** 'x'**)**

**except** ValueError**:**

**return** **False**

# Compare calculated to read value

**if** result**.**lower**()** **!=** checksum**.**lower**():**

**return** **False**

**return** **True**

# Static method \_\_fix\_zeros:

#======================================================

# Replaces spaces with zeros

#

# Params:

# data:

# A list of size 10 containing each byte of

# the ID as a 2-digit hexadecimal text string

# e.g ["DE","CA","FF","C0","FF","EE",...]

# Returns:

# The data string with zeros instead of spaces

#======================================================

*@staticmethod*

**def** \_\_fix\_zeros**(**data**):**

**return** data**.**replace**(**' '**,** '0'**)**

# Method \_\_read\_sequence:

#======================================================

# Reads a sequence of bytes from the serial

# connection

#

# Params:

# serial\_connection:

# A serial.Serial object which has been

# initialised to use the serial port that the

# scanner module is connected to.

# Returns:

# A string containing the ID read, or False if the

# bytes read are bad

#======================================================

**def** \_\_read\_sequence**(**self**,** serial\_connection**):**

serial\_connection**.**timeout**=**3

tag\_string **=** ''

byte\_read **=** serial\_connection**.**read**()**

# Check timeout

**if** byte\_read**==None** **or** byte\_read**==**""**:**

**return** **False**

# Check first byte

**if** int**(**ord**(**byte\_read**))** **!=** self**.**RDM\_START**:**

**return** **False**

# Read 12 bytes after the RDM\_START byte has been read

expected\_len **=** 12

**while** expected\_len **is** **not** 0**:**

expected\_len **-=** 1

byte\_read **=** serial\_connection**.**read**()**

# reset the loop counter if the byte is RDM\_START

**if** int**(**ord**(**byte\_read**))** **==** self**.**RDM\_START**:**

expected\_len **=** 12

**continue**

# Add the currently read byte to the tag\_string

**if** ord**(**byte\_read**)** **!=** self**.**RDM\_END**:**

tag\_string **+=** chr**(**ord**(**byte\_read**))**

**continue**

**break**

# Split tag\_string into data and checksum

data **=** tag\_string**[**0**:**len**(**tag\_string**)** **-** 2**]**

checksum **=** tag\_string**[**len**(**tag\_string**)** **-** 2**:**len**(**tag\_string**)]**

# Check checksum

checksum\_ok **=** self**.**\_\_verify\_checksum**(**data**,** checksum**)**

**if** **not** checksum\_ok**:**

**return** **False**

**return** self**.**\_\_fix\_zeros**(**data**)**

# Method \_\_read\_sequence:

#======================================================

# Prints the tags as they are read

#

#======================================================

**def** do\_work**(**self**):**

serial\_connection **=** ''

**try:**

**print** **(**"Creating serial connection"**)**

serial\_connection **=** serial**.**Serial**(**self**.**SERIAL\_PORT**,** baudrate**=**self**.**BAUDRATE**)**

**print** **(**"Serial connection created"**)**

**while** **True:**

data **=** self**.**\_\_read\_sequence**(**serial\_connection**)**

# check if its an actual RFID string

**if** data **is** **False:**

**print** **(**"No cat detected"**)**

data**=**"000000000000"

**else:**

**print** **(**data**)**

# wait for 2 seconds

time**.**sleep**(**2**)**

# reset all input buffer data

serial\_connection**.**flushInput**()**

**except** KeyboardInterrupt**:**

**print** **(**"\nKilled. Serial port was safely closed."**)**

serial\_connection**.**close**()**

# Main method:

#===============================================

# initialises RDM6300 object

# runs RDM6300.do\_work()

#===============================================

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

rdm6300\_reader **=** RDM6300**(**'/dev/serial0'**)**

rdm6300\_reader**.**do\_work**()**

### **10.10.12 TestRotation.py**

**import** RPi**.**GPIO **as** GPIO

**from** time **import** sleep

**import** sys

**from** hx711 **import** HX711

**from** MotorControl **import** **\***

# Raspberry Pi Pins

Motor2A **=** 11

Motor2B **=** 18

PWM **=** 16

# Initialize Pins

GPIO**.**setmode**(**GPIO**.**BOARD**)**

GPIO**.**setup**(**Motor2A**,** GPIO**.**OUT**)**

GPIO**.**setup**(**Motor2B**,** GPIO**.**OUT**)**

GPIO**.**setup**(**PWM**,** GPIO**.**OUT**)**

GPIO**.**output**(**Motor2A**,** GPIO**.**LOW**)**

GPIO**.**output**(**Motor2B**,** GPIO**.**LOW**)**

GPIO**.**output**(**PWM**,** GPIO**.**LOW**)**

Motor\_PWM **=** GPIO**.**PWM**(**PWM**,** 200**)**

Motor\_PWM**.**start**(**0**)**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

# Run motor for ~60 degrees of movement

speedControl**(**Motor\_PWM**,** 2**,** 30**,** 'f'**,** 0.41**)**

### **10.10.13 TestScale.py**

**import** RPi**.**GPIO **as** GPIO

**import** time

**import** sys

**from** hx711 **import** HX711

hx **=** HX711**(**29**,** 31**)**

**def** cleanAndExit**():**

**print** **(**"Cleaning..."**)**

GPIO**.**cleanup**()**

**print** **(**"Bye!"**)**

sys**.**exit**()**

**def** scaleInit**():**

# Reading format defined by manufacturer

hx**.**set\_reading\_format**(**"LSB"**,** "MSB"**)**

# Set how many units returned by the chip are equivalent to 1 gram

hx**.**set\_reference\_unit**(**390**)**

**def** testScale**():**

hx**.**reset**()**

hx**.**tare**()**

**while** **True:**

**try:**

**print** **(**"HX711 Scale tester. Calibrated for 5kg weight sensor. Displayed values are in grams"**)**

keyboard **=** raw\_input**(**"Enter a to print 20 values\nEnter c to print infinite values\n Press Ctrl+C to exit.\n>"**)**

avg\_index**=**0

current\_sum**=**0

measured\_avg**=**0

actual**=**0

**if** keyboard **==** "a"**:**

#reads 20 values and calculates the average

**for** counter **in** range **(**20**):**

# Read a value

w **=** hx**.**get\_weight**(**5**)**

**print** **(**" Got value " **+** str**(**counter**)** **+** "/20: " **+** str**(**w**))**

current\_sum **+=** w

# Reset the input pins

hx**.**power\_down**()**

hx**.**power\_up**()**

time**.**sleep**(**0.1**)**

measured\_avg**=(**current\_sum**/**20**)**

**print** **(**"Average is: "**+**str**(**measured\_avg**))**

hx**.**power\_down**()**

hx**.**power\_up**()**

time**.**sleep**(**0.1**)**

**if** keyboard **==** "c"**:**

#reads values until interuppted by ctrl+c

**while** **True:**

#Read a value

w **=** hx**.**get\_weight**(**5**)**

**print** **(**" Got value :"**+**str**(**w**))**

current\_sum **+=** w

#Reset the input pins

hx**.**power\_down**()**

hx**.**power\_up**()**

time**.**sleep**(**0.1**)**

**except** **(**KeyboardInterrupt**,** SystemExit**):**

cleanAndExit**()**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

GPIO**.**cleanup**()**

#Initialise HX711 object (see HX711.py)

hx**=**HX711**(**29**,**31**)**

scaleInit**()**

hx**.**listen**()**

testScale**()**

### **10.10.14 Update**

#!/bin/bash

# Cat feeder script

# start loop

# convert sensor data to csv file

# delete old csv file from google drive

# wait 10 sec

# copy new csv file to google drive

# wait 60 sec

# remove new csv file from local directory

# wait 10 sec

# end loop

**while** **true**

**do**

**python** convert\_to\_csv.py sensor\_data.txt out.csv

**echo** "converted"

rclone delete Cat**:**out.csv

**sleep** 10

rclone copy out.csv Cat**:**

**echo** "updated"

**sleep** 60

**rm** out.csv

**echo** "removed"

**sleep** 10

**done**