The George Washington University
School of Engineering and Applied Science
Department of Electrical and Computer Engineering
ECE 4925W
Spring 2022

Final Product Review

KEEPSAFE

Jaret Williams – Electrical Engineering
Nathan Pen – Electrical Engineering
Sean Letavish – Electrical Engineering

Professor Amir Aslani
April 6, 2022

ABSTRACT

With E-commerce quickly becoming one of the most popular forms of shopping, many people are getting more and more packages per day. When the packages are delivered to the doorstep of a house, they are sitting in the wide open for anyone to grab and unfortunately many people lose their beloved packages to theft every year. There are some options currently available but require either the delivery driver needs a special scanner or gets entrance to a person's home or garage. Our team plans on solving this issue with our product, KeepSafe. KeepSafe is a smart package locker with two compartments that are unlocked until a package is placed into one of the compartments. The locker will recognize the presence of a package from the sensor system inside, alert the user using the KeepSafe app, and then lock the box from any porch pirates. All the functions of KeepSafe will be performed by the Wi-Fi connected microcontroller that communicates with the user application and controls the various components inside each compartment. The user will then be able to unlock the package door with the app or the backup keypad. We will do extensive testing on our prototype for reliability of use, physical security, and ease of use. KeepSafe's goal is to be user friendly to the user but also the delivery driver so that the product will be utilized to its full extent. We intend to have KeepSafe be placed outside of residential homes, but it could be used for apartments or any small-scale deliveries that are left outside.

Contents

1. Introduction

1.1 Background

In the past few years online shopping has seen an exponential increase in users whether it was because of the Covid-19 pandemic or the sheer convenience. With more deliveries and online purchases comes with more opportunities for thieves to prey on innocent people in the form of porch piracy. In 2020 about 6 billion dollars' [1] worth of packages were estimated to have been stolen. With that much money loss to theft, it is only logical that people would like to protect their purchases and not have them stolen when they are delivered to their house because the consumer, delivery company, and the seller are all affected by these acts of crime. KeepSafe will not be one the first intelligent products in this field but a better and smarter solution to what is currently available.

1.2 Market Justification

Currently there are simple solutions such as USPS mail drop boxes that have no electrical feature, no alert system, bulky, and just stop thieves from taking the package. Other smart options include a padlock design that can go on any lockbox and alert the user when something has been delivered, but it requires the delivery driver to have a certain bar code scanner to properly use it. This proved to be ineffective due to technical glitches but most importantly the lack of user friendliness and complications that arose with having the delivery drivers take time to figure the barcode system. Lastly Amazon has a program called Amazon Key that allows delivery driver temporary access to someone's garage or house door to allow the delivery driver to leave the package inside. Although this is a good idea, many people are uncomfortable with a stranger have access to their home with all their belongings inside. There is a clear gap in the market that KeepSafe has the opportunity to fill.

1.3 Novelty

KeepSafe is unlike any product currently on the market since it is the full package. While the Amazon Key is one option for package security, KeepSafe does not require any delivery driver to enter a consumer's home. In fact, unlike other lock boxes, no prior access code is required for the delivery, which need to be provided when the package is ordered. KeepSafe is an automated package protection locker, where the only input required from the delivery driver is the package itself. Once the package is detected, KeepSafe automatically locks, and provides push notifications to the owner on the KeepSafe app. This further separates KeepSafe from any residential package locker on the market, with smart home capabilities and an application. KeepSafe is a combination of all the available options currently on the market. It provides the clean look of plain "bins", that simply just take the package out of view of potential thieves while not actually locking, and the security of lockers, without the need for a keycode or combination to be given to the delivery driver. The combination of aesthetic appeal and ease of use of everyone involve sets KeepSafe above the current package security products on the market.

Figure 1: Rendering of KeepSafe on a Front Porch

2. Impacts and Effects

2.1 Needs and Motivation

While Founder and CEO Jaret Williams was in middle school, his dad was active duty in the United States Navy and deployed overseas. While overseas, his dad sent home Christmas presents for Jaret and his family, but when Christmas arrived, and the delivery status for the presents said "delivered", it had become apparent that the packages were stolen. This heart-breaking story inspired the idea that is now KeepSafe, a residential package locker that helps prevent package theft, and the destruction of future Christmases. According to a 2021 study by Finder, 14% of Americans say they have been victims of package theft, and about $5.4 billion worth of packages were stolen between 2020-2021. With online shopping only predicted to continue its steady growth, so too can package theft.

2.2 Economic Impacts

By consistently preventing package theft, KeepSafe will directly save the homeowner, business, and delivery company money. For the homeowner, KeepSafe prevents the need from having to contact customer support to try and get a replacement delivered, which can often take a significant amount of time. More importantly though, if customer service is not willing to provide a replacement, the homeowner is sometimes forced to order the package again, which money out of their pocket that did not need to be spent. For the business, KeepSafe is useful for almost the same reasons, just on the other side of the coin. Customer support and providing a replacement is not needed if the package is protected from the start. Lastly, for the delivery companies, one less unnecessary stop on their routes saves time and gas, allowing drivers to get to more houses rather than revisiting those who were victims of package theft. With over $5 billion worth of packages

being stolen in the last year, this significant financial loss to both consumer and business is easily preventable with the use of KeepSafe.

2.3 Environmental

Extra deliveries needed to replace stolen packages increase the carbon footprint of delivering a single product. Preventing package theft with KeepSafe will eliminate the need for these additional deliveries and reduce the overall potential carbon emissions of individual online purchases. Alongside the removal of extra deliveries, KeepSafe can also keep porch pirates off the road as well, as many thieves use their vehicles to "hit" multiple homes in a short amount of time. If thieves know that entire neighborhoods utilize KeepSafe, the drive may no longer be worthwhile.

2.4 Safety Impacts

During the holiday season, package theft shifts from being primarily a crime of opportunity to purpose driven premeditated theft. These thieves, often called "porch pirates", go around from neighborhood to neighborhood looking for packages appearing to have value to steal. Having packages in communities protected by KeepSafe will decrease the number of these porch pirates and improve the overall safety of the community, especially when package theft is at its highest during the holiday season. Not only can KeepSafe keep these potentially dangerous criminals off of one's own property, but if entire communities were to utilize this technology, porch pirates could be completely deterred from even going near that neighborhood in the first place.

3. Overall System Requirements and Specifications

Table 1: Overall System Requirements and Specifications

Requirements	Specifications
(1) Package must be detected	(1) Any sized package must be consistently recognized by the system
(2) Package must be secured	(2a) Automatically locks the packages inside the box when delivered. (2b) Lock must engage every time with no error.
(3) User must be notified	(3) User must be alerted every time after a package is detected and secured
(4) Overall system must be protected from theft	(4a) KeepSafe will bolted to the ground when possible (4b) Provide the user will GPS coordinates of the box when it is being moved
(5) User must be able to control KeepSafe through an application	(5) User will be able to perform the following functions through the app: - Unlock KeepSafe

	- Check Package Status - Check GPS Location - Check Camera

4. Approach to Overall Design

4.1 Current Design

The current design for KeepSafe includes three main modules, package detection, package security, and the application. Package detection is based on two devices, a door switch and the load sensor network. To detect if a package is placed into KeepSafe, first the door must be open, which then wakes up the load sensor network. The load sensors then begin taking readings and determine if the weight has changed. The door switch then detects if the door is closed again, which will then start the package security module.

Before a package is even placed inside KeepSafe, it has a PIR motion sensor that is constantly active, looking for motion. When motion is detected for a continued period, the camara module is then activated to take a picture of whatever is in front of KeepSafe. If pictures were taken at every time movement is detected, the Raspberry Pi will be storing unnecessary pictures of non-threats. By including the idea of “consistent motion”, this ensures that only pictures of individuals approaching KeepSafe and standing in front of it are taken. Once the door is opened, a package is detected, and the door is closed again, a servo is activated that drives the locking mechanism to secure KeepSafe. KeepSafe is also equipped with a 1x4 keypad that allows the user to unlock KeepSafe if they do not have access to the app. A GPS is also included in case thieves try and steal the entire unit. The GPS is only activated though with the use of gyroscopic sensor, that detects if the leveling of KeepSafe changes, indicating that someone is picking it up and changing its orientation. Upon detection of physical motion of the box, indicating a potential theft, the user is notified and can access the exact current location of the box provided by the GPS module, overlayed onto a map user interface embedded in the app.

The last module of the current design is the web-based application. The application is run through an Apache server, with the back-end being Python and the front-end being HTML and CSS. The application allows the user to check the camera, see if a package is inside, how much it weighs, see where KeepSafe using its GPS, and allows the user to unlock and lock KeepSafe.

4.2 Evolution of the Design

Each module of the design went through a series of changes for multitude of various reasons. The biggest change made was regarding package detection. When KeepSafe was first incepted, the idea for package detection relied on the use of photosensitive resistors. Using an array of these resistors, it would be easy to determine if the light level changes, indicating that a package had been placed on top of them. While this idea was proven to work properly, this design would have been more complex to manufacture, and to create this array, a large number of parts would have also been necessary. With the load cell network, only four components are necessary, and

not only can the package be detected, but the exact weight of the package can also be displayed to the user which adds significant value.

For package security, multiple components have been added to enhance KeepSafe's security and enhance its value. Initially, the door trigger and servo were the only components used for package security. The camera, GPS, PIR sensor, and gyroscope were all added as the design evolved. Additionally, the physical construction of the box has changed from what we originally had planned. The team originally wanted to have a multiple compartment design that could hold most packages but due to machine shop restrictions, the system was reduced to a single compartment and much smaller design. The materials used also were adjusted to make the construction of the KeepSafe prototype easier. At first a metal design was proposed to ensure high levels of package security, but this changed to a wooden design out of plywood which made the construction significantly easier.

5. Subsystems Requirements and Specifications

Figure 2: Overall Level 0 Block Diagram

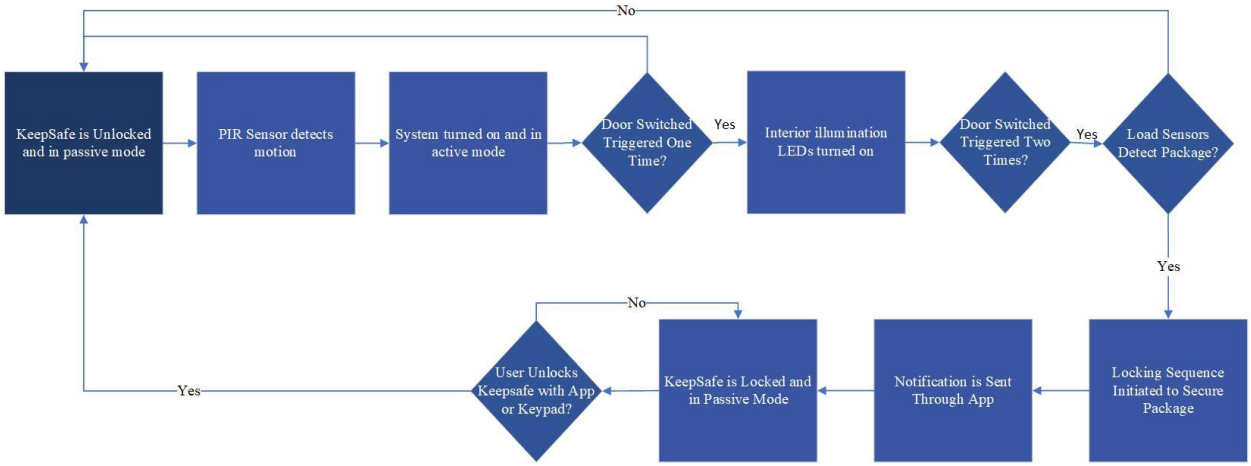


Figure 3: Overall Level 1 Block Diagram

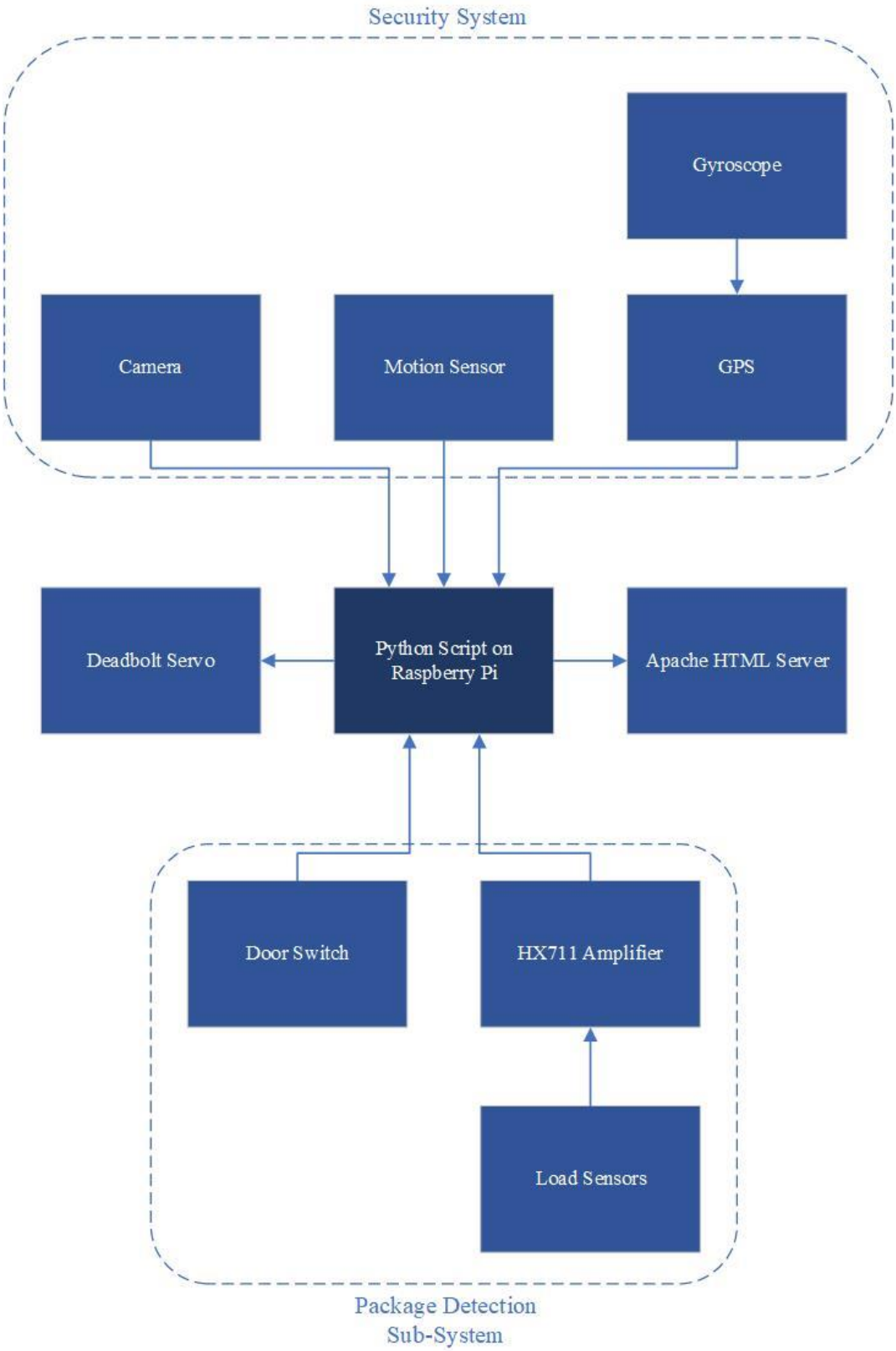


Figure 4: Overall KeepSafe Operation

5.1 Electrical Subsystem

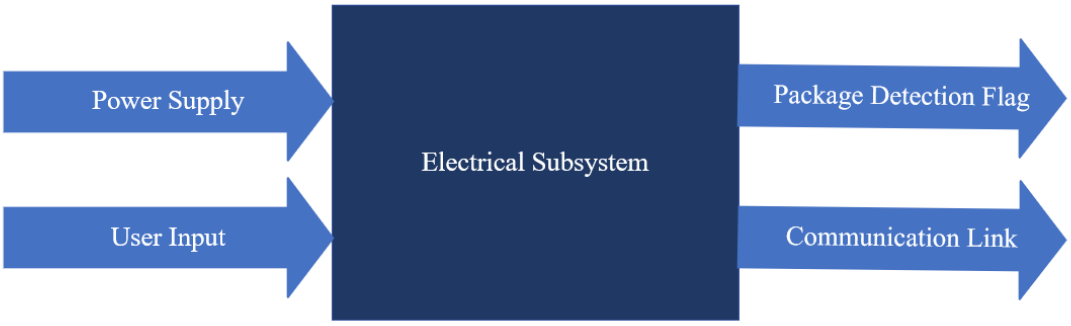


Figure 5: Electrical Subsystem Level 1 Block Diagram

Table 2: Electrical Subsystem Requirements and Specifications

Requirements	Specifications
1. The device must illuminate inside of box when opened	1. Line the inside of the box with bright low power LEDs to make nighttime use easier
2. The device must be equipped with an electronic door switch	2. Device knows the state of the door to then check for packages
3. The device must be equipped with an electronic keypad for manual unlock and package retrieval	3. Packages can be retrieved by anyone in the household if they do not have access to the app

5.1.1 Package Detection Subsystem

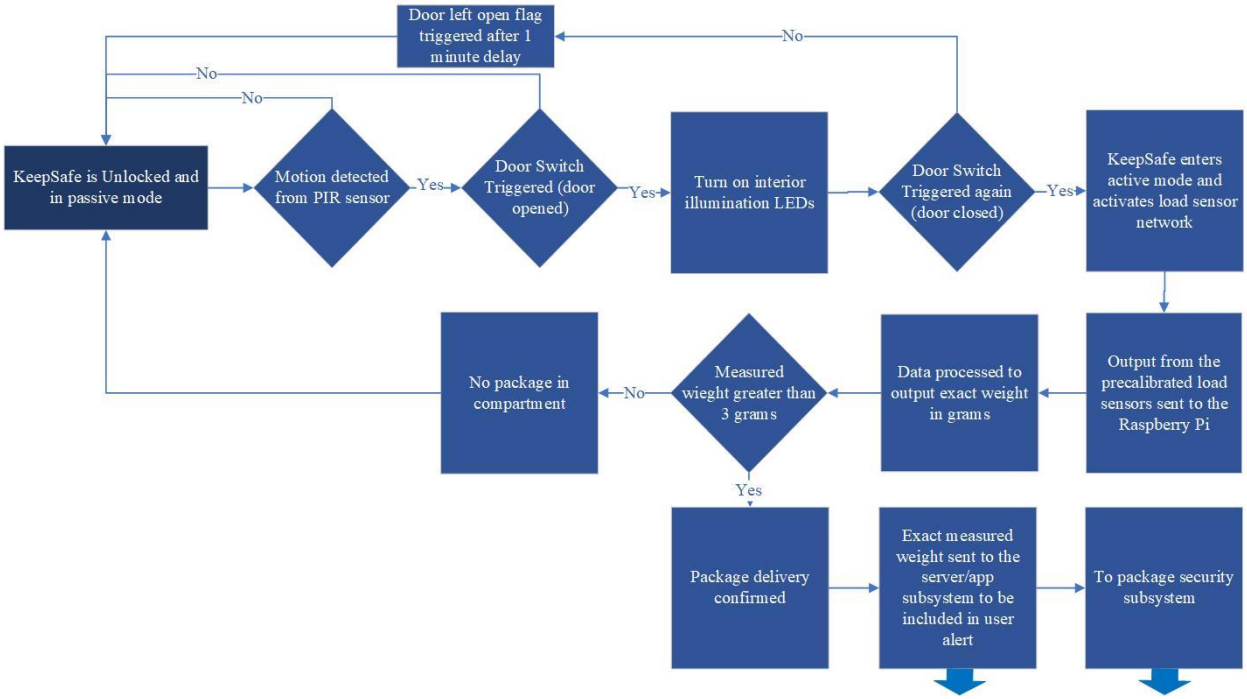


Figure 6: Package Detection Subsystem Level 2 Block Diagram

Table 3: Package Detection Subsystem Requirements and Specifications

Requirements	Specifications
1. The device must be able to detect when a package has arrived	1. Accurately detects all packages over 3 grams placed inside a compartment using very sensitive load sensors
2. Accurately inform the user of the packages weight	2. Measured weight must be accurate up to 5% error

5.1.2 System Security Subsystem

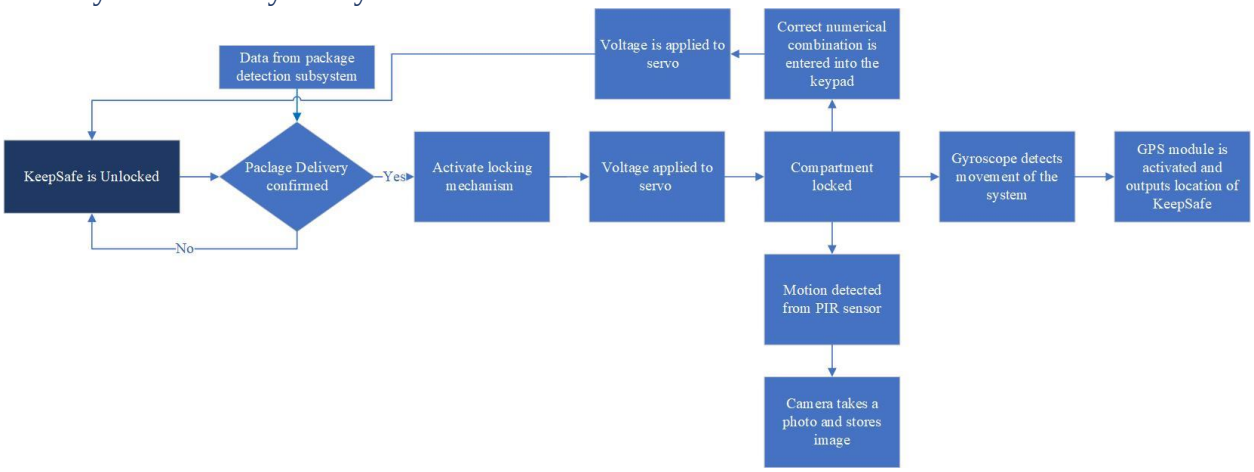


Figure 7: System Security Subsystem Level 2 Block Diagram

Table 4: System Security Subsystem Requirements and Specifications

Requirements	Specifications
1. Package secured after delivery	1. Servo is activated and locks compartment following package delivery confirmation from electrical subsystem
2. Takes a photo when there is activity around the box	2. A photo is taken from the camera when motion is detected from the PIR sensor and stores the image
3. Track the location of the system if it is stolen	3. GPS module is activated by the gyroscope when the entire system is being physically moved (i.e., potential theft)

5.2 Mechanical Subsystem

5.2.1 Package Security

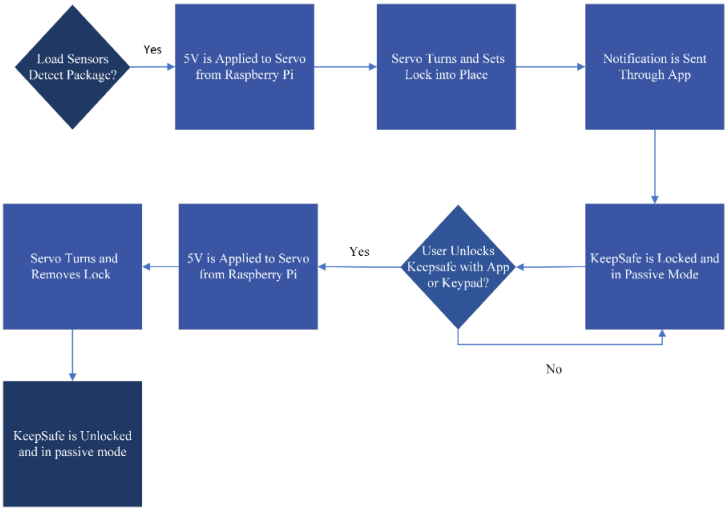


Figure 8: Locking Mechanism Level 1 Block Diagram

Table 5: Security Subsystem Requirements and Specifications

Requirements	**Specifications**
1. The device must secure the package	1a. Automatically locks the packages inside the box when delivered 1b. Locks must engage in under 0.5s 1c. User can unlock the box through phone app control or a manual keypad input on the box itself 1d. Lock must engage every time with no error
2. The device must utilize materials and design to prevent thieves potentially taking the device itself	1a. Materials used must be strong and heavy to prevent thieves from trying to pick up and steal the entire device. The device can potentially be bolted down as well to further prevent the device being stolen 1b. GPS system must be used and must be accurate within a 2m radius 1c. Motion sensors must be combined with camera to take pictures of any disturbances. When an object is within a 7m range for over 10s, camera must take picture and save the photo

3. The device must be easy to use by delivery driver to ensure consistent package security	3. No external input needed from the delivery driver, must be able to simply open the box and place the package inside to ensure the device is consistently used by the delivery driver

5.3 Application Subsystem

5.3.1 Networking Subsystem

Figure 9: Networking Subsystem Level 1 Diagram

Table 6: Networking Subsystem Requirements and Specifications

Requirements	Specifications
1. Enable message passing between server and phone or other devices	1. Connect to and transmit data over Wi-Fi
2. Transmit formatted URL with HTTP/GET format	

5.3.2 Server Subsystem

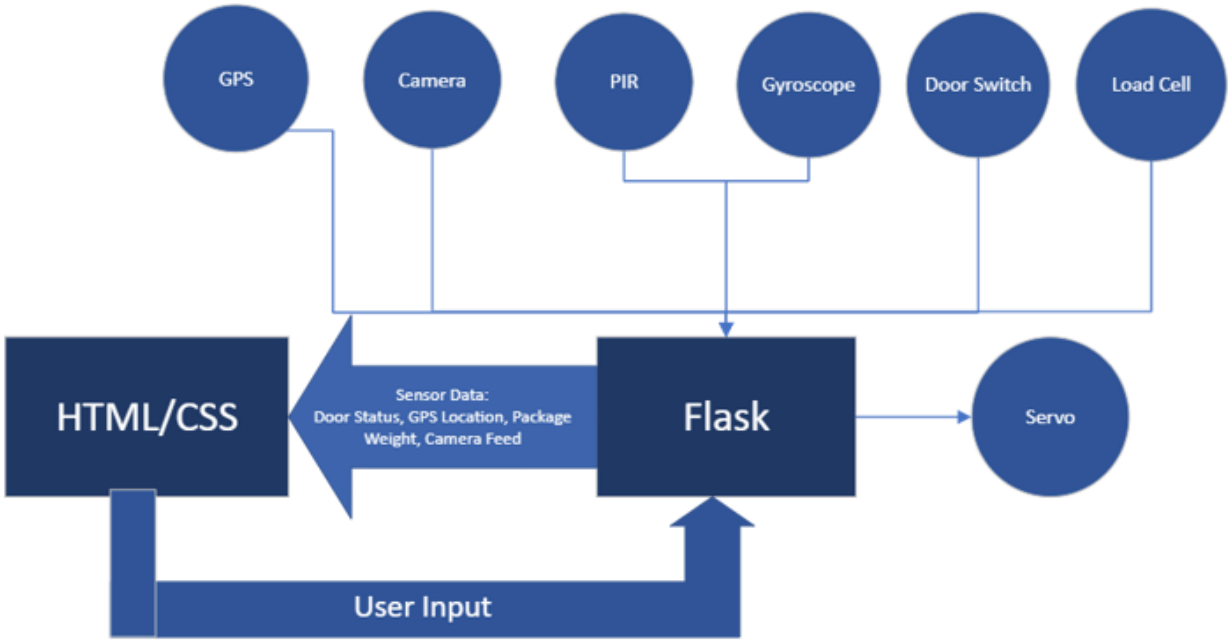


Figure 10: Application Level 1 Block Diagram

Table 7: Server/App Subsystem Specifications and Requirements

Requirements	Specifications
1. Provide notifications for package delivery, battery status, and potential security risks	1a. Easy to understand, streamlined user interface 1b. Create a notification when a package has arrived and how much the package weighs 1c. Create a notification when KeepSafe has less than a day's worth of battery remaining 1d. Create a notification regarding security (if KeepSafe has moved or otherwise been tampered with)
2. Communicate seamlessly between front end and back end of application (minimum wait time)	

6. Module Design

Overall System Overview

KeepSafe is broken down into three overall subsystems, the electrical subsystem, the mechanical subsystem, and the application. The electrical subsystem comprises all the circuitry

required for both package detection and package security. This subsystem includes the PIR sensor, the servo, the gyroscope, the GPS module, the door switch, the camera module, the load cell network, the keypad, the interior light, the Raspberry Pi itself, and how all those components connect and communicate with one another. The mechanical subsystem encompasses the body of KeepSafe and the locking mechanism used for package security. The last module is the web-based application, that allows the user to control and monitor KeepSafe from their phone or computer. The application allows the user to unlock KeepSafe, check if a package has arrived, check the camera, and delivers push notifications when a package is delivered or if there is a potential threat.

6.1 Electrical Subsystem

The electrical subsystem covers a lot of the smart functionality of KeepSafe for package detection and package security features. The modules in this system are essential for the consistent logical flow of the overall system with various inputs and outputs determining the current state of the system. The figure below shows some of the components on the breadboard located in the top electronics compartment of the box.

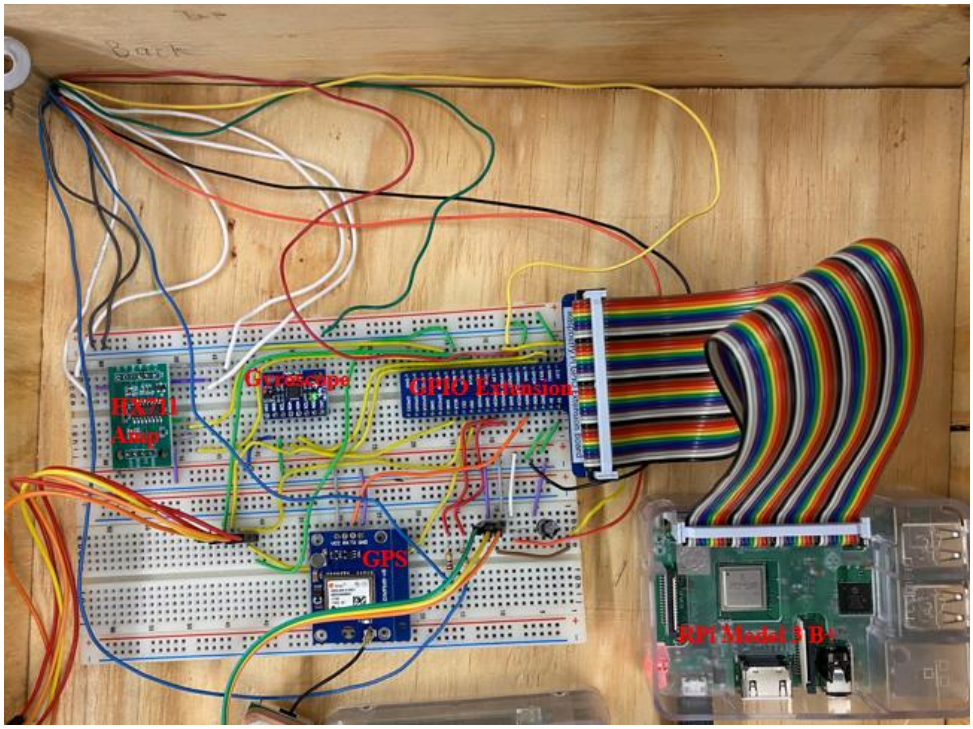


Figure 11: Electrical Subsystem Top Compartment

6.1.1 PIR Sensor

The PIR sensor's primary function is to detect when there is motion around KeepSafe and take a picture of the surroundings. The power conservation mode was not implemented in this prototype, but in a full functional system KeepSafe would utilize the PIR sensor to wake from its sleep/power conservation mode. This is a key component of the overall logic flow and the team performed testing on our current PIR sensor to ensure

that it meets the motion detection criteria. The PIR sensor works great for our current design, the only potential changes could be finding a similar PIR sensor with less of a detection range, so it only recognizes motion immediately in front of the box. For some users that would be placing KeepSafe in a busy area, the constant motion around the box would be detrimental to the overall power consumption.

Figure 12: PIR Motion Sensor [3]

Figure 13: Installed PIR Sensor

6.1.2 Servo

The high torque servo used to lock a compartment when a package is placed inside. It is only activated after there is motion in front of the box (PIR), the door is opened (door switch), a package is placed inside, and its weight is recognized (load sensors), and the door is closed again. Once these events take place, KeepSafe knows that the compartment needs to be in a locked state and the servo turned to its locked state. The locking mechanism itself will be discussed in the mechanical section.

Figure 14: Adafruit High Torque Servo [9]

6.1.3 Gyroscope

This sensor is used to control the activate specific elements of the security system. Once KeepSafe is locked, there must be measures in place to protect the entire box from being stolen with not all users having the ability to bolt it down. The GPS module discussed in section 6.1.4 will be used to track the location of KeepSafe if it is stolen and the gyroscope will be used to activate it. A gyroscope is used to measure orientation and angular velocity, making it the perfect fit for determining if KeepSafe is being physically moved and recognizing this security threat. Similarity to the other sensors, readings from gyroscope are checked with there is movement around the box and if the gyroscope is triggered, the GPS module is then activated to send the real time location of KeepSafe to the user. The gyroscope can be seen in figure 11 mounted on the breadboard with some of the other electrical components. This module has other functions, but the gyroscopes pins were the only ones used.

Figure 15: MPU-6050 Gyroscope [4]

6.1.4 GPS Module

When the gyroscope senses the entire system being physically moved, the GPS module will provide the location KeepSafe to the user. This is important to ensuring the security of the entire KeepSafe system and potential valuable deliveries inside. The activate of the module is activated by the gyroscope as discussed in the previous section. The GPS

module can be seen in figure 11 mounted on the breadboard with some of the electrical components.

Figure 16: GY-NEO 6M GPS Module [5]

6.1.5 Door Switch

Once motion is detected around KeepSafe, the door switch is an important logical step to determine when a package is placed into the box. The door must first be opened and then closed again for the system to check for a package and then lock. On the first trigger (door opened), the interior illumination elements are turned on to light up inside the compartment and then on the second trigger (door closed), the output on the load sensors are checked to determine if a package was delivered on that open/close sequence. Figure 18 below shows how the switch was installed to be triggered by the door every time it is closed/opened and also shows the cable management system to ensure a clean appearance inside the box.

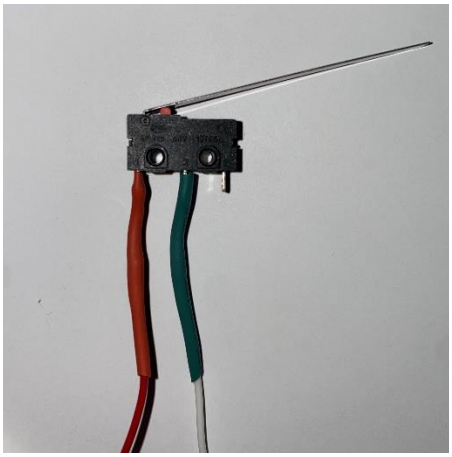


Figure 17: Door Switch Picture

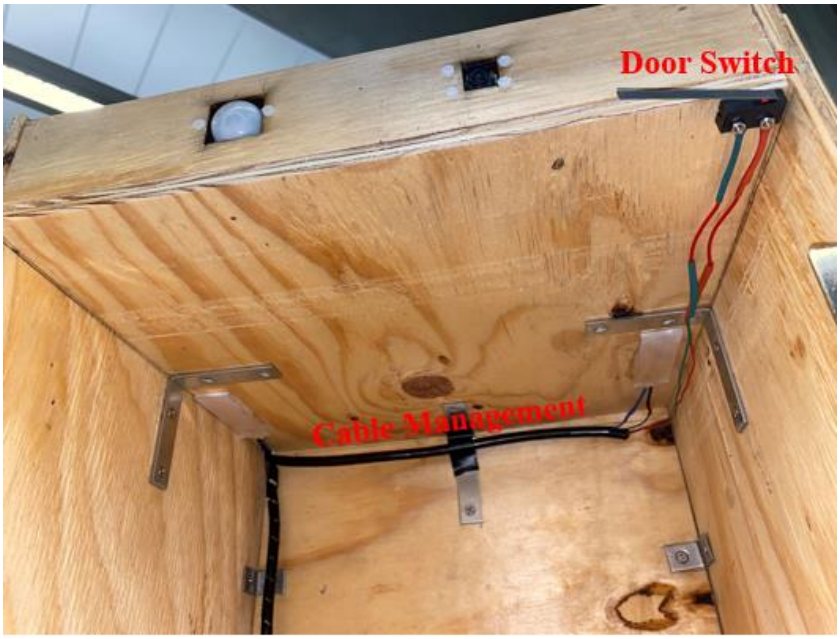


Figure 18: Installed Door Switch

6.1.6 Camera Module

The camera module is an additional feature of the security system. This allows the user to see who is interacting with the box and the images stored could be used in any theft situations. Another action resulting from motion detected in front of the box with the PIR sensor is the camera module taking the picture and saving it. This picture can be accessed from the phone application to see recent activity around the box. Additionally, the user can use the live camera feed feature in the app for the user to access at any time.

Figure 19: Raspberry Pi Camera Module [8]

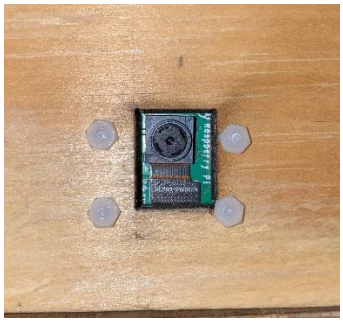


Figure 20: Installed Camera Module

6.1.7 Load Sensor Network

The load sensor network in each compartment is how KeepSafe will detect if any package is placed inside the box. Due to its sensitivity and accuracy seen in initial testing of the sensors, any sized package will be detected. Each compartment will have a false floor that will rest on top of the sensors placed inside their 3D printed mounts. When a package is placed on top of the surface the sensors not only recognize that there is a package placed inside the box but also output the exact weight of the package inside the box. This additional feature gives the user a rough idea of the package delivered was a result of the very successful implementation of the load sensor network. The load sensors are wired together and then sent to the HX711 amplifier that is then sent to the RPi pins, taking 4 load sensor input pins to the amplifier and outputting 2 amplified signals that is sent to the RPi.

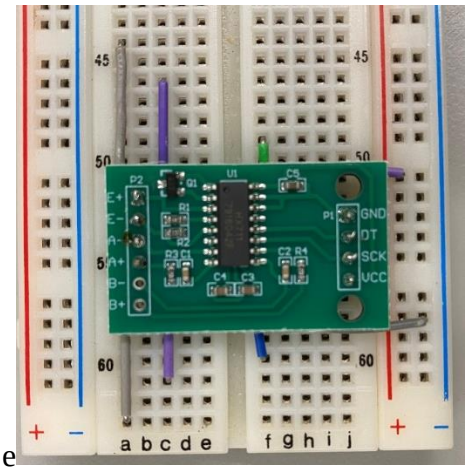
e 

Figure 21: HX711 Amplifier

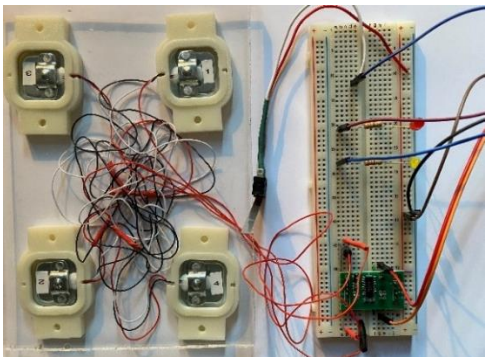


Figure 2222: Load Sensor Network Picture

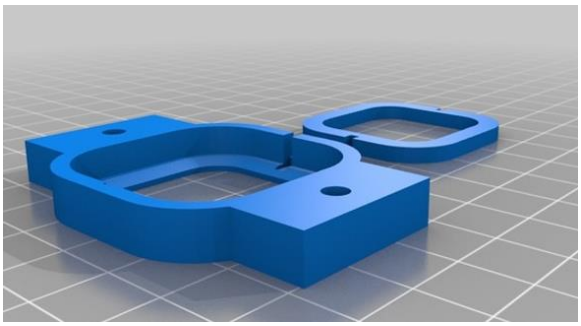


Figure 23: Load Sensor Mounts [12]

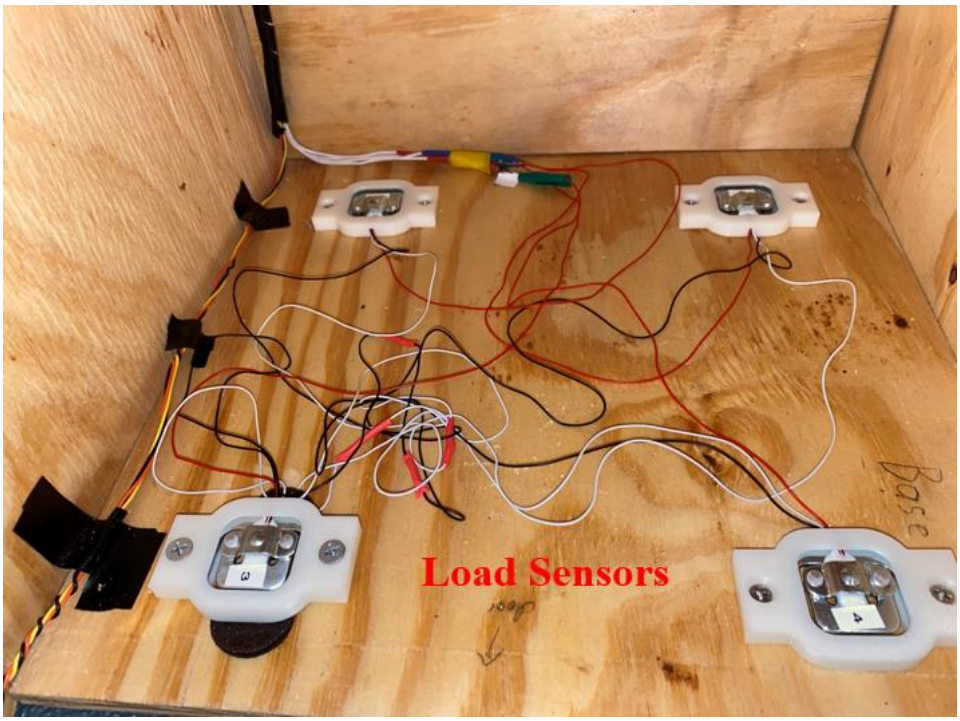


Figure 24: Load Sensor Network Installed

During the installation process of the load sensor network on the bottom of the box the team encountered some challenges with ensuring that the false floor that sits on tops of

the load sensors remains level for accurate package detection and weight measurement. This was caused by the bottom piece of plywood not being completely flat. To make up for this inconsistency in the wood, the height of the bottom left load sensor was adjusted in a trial-and-error process to find the height at which the floor perfectly sits level on top of the sensors.

Various other sensors were considered in the early design stages of the project such as ultrasonic, IR, and photoresistors. From the tests performed, the team recognized that these sensors could detected most packages and could be relatively functional but lacked the full consistent coverage desired. Some of the sensors had blind spots and lower accuracy making little to no error impossible. Additionally, the number of each of these alternative sensors needed inside the compartment was a function of the compartment size. This is far from ideal considering the number of sensors that would be needed to maximize coverage across the entire compartment floor with added complexity and many additional pins needed on the RPi. The load sensor system was the perfect solution to consistent package detection while only needing two GPIO pins on the RPi.

6.1.8 Keypad

The user will also be given the option to manually unlock the box not using the app. This will be done by using a small 1x4 keypad located on the side of KeepSafe. The user will have a unique 4-digit combination that will unlock the compartments on the box when it is entered correctly. When the correct combo is entered, the servos will turn the bolt to the unlocked positions allowing the user to retrieve the packages (locking mechanism further discussed in mechanical section).

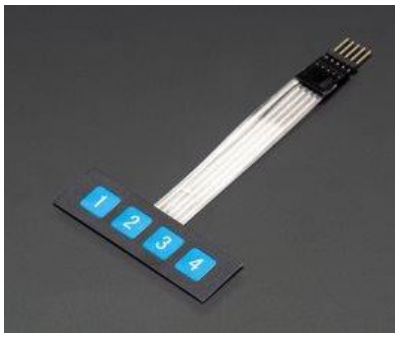


Figure 24: Adafruit 1x4 [7]

Figure 25: Keypad Installed

6.1.9 Interior Light

When the door is opened, interior lights will turn on automatically to ensure a smooth delivery process. This interior lighting system will consist of two LED modules on the top of each compartment that will completely illuminate the inside when a package is being placed inside by the delivery driver. The lights will turn off when the door is closed (second door switch trigger).

Figure 26: Adafruit LED Backlight Module [13]

6.1.10 Raspberry Pi

The Raspberry Pi controls the operations of the entire system. With many different components and a complicated logic flow, this central control unit is effective in performing the sequential operations needed through all possible inputs and outputs. In total, 15 GPIO pins are used on the RPi.

Figure 27: Raspberry Pi Microcontroller [10]

6.2 Mechanical Subsystem

6.2.1 Locking Mechanism

The locking mechanism consists of a high torque servo driving a deadbolt to secure the door. This command is triggered from the logic flow in the electrical subsystem once a package delivery has been confirmed inside a compartment. Different designs were considered such as using a solenoid to drive the deadbolt, but the servo was chosen due to its electrical properties.

The locking mechanism consists of a rack and pinon design. This can be seen in the figure below where the servo turns a gear the drives the rack (bolt) which is pushed into a hole on the side of the box. This was designed in SolidWorks then 3D printed to be installed on the door of the box for locking.

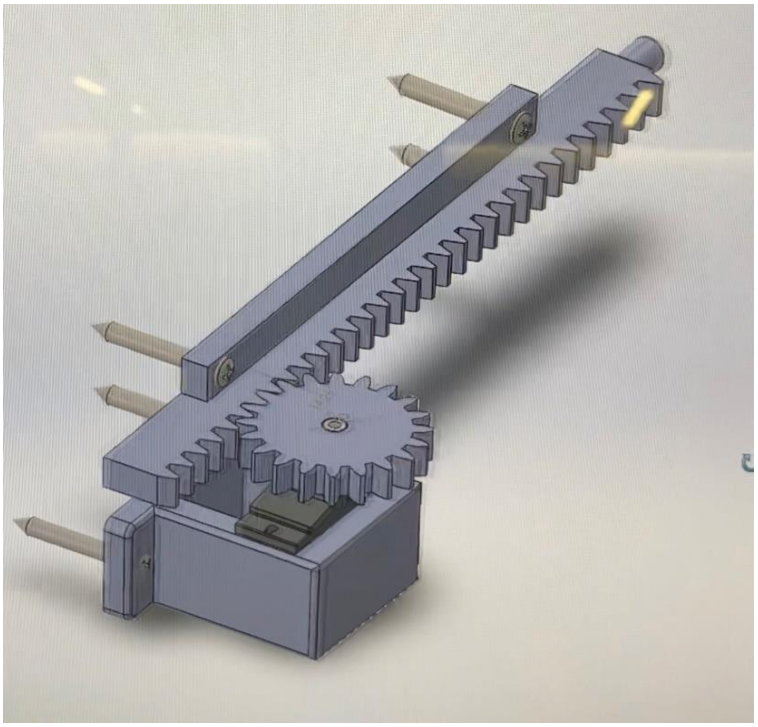


Figure 28: Locking Mechanism CAD

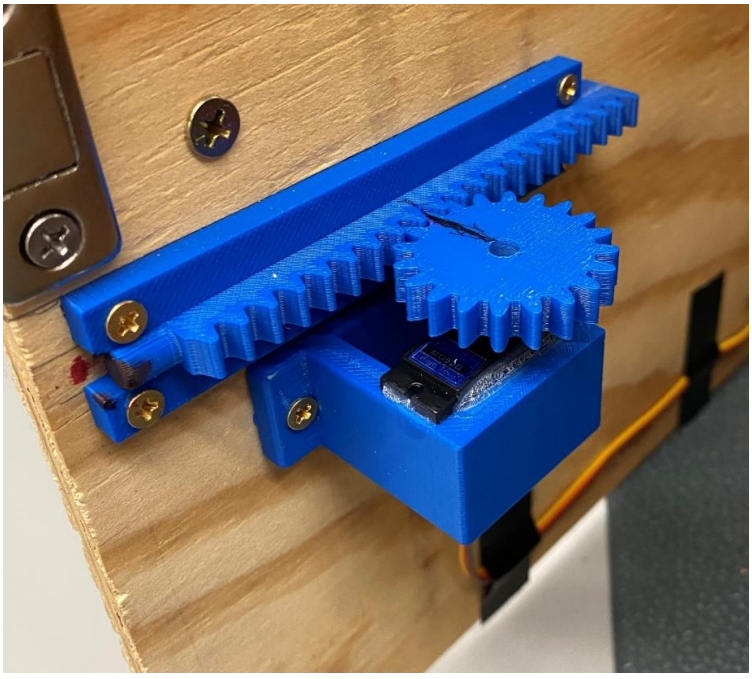


Figure 29: Locking Mechanism Installed

6.2.2 Box Design

This semester, the team had to integrate the existing and functional electronic components into a box to prototype KeepSafe and demonstrate its various features. After meeting with the machine shop employees, the team decided to construct the box out of basic plywood due to weight, cost, and construction considerations. As seen in the sketches below, the team went to a significantly scaled down design, with the final product having dimensions of 1’ x 1’ x 1’. Although, this sizing doesn’t fit most medium to large sized packages, it is an appropriate size for the prototype developed for the capstone sequence. Many various mechanical components had to be purchased from McMaster-Carr and Amazon (see appendix 3 for detailed BOM). These components included various brackets, screws, hinges, bolts, and more the create desired design. One issue encountered during the construction process was with the quality plywood boards purchased. Not only did many of them had chips or other visual defects but the main issue came from many of them not being flat. These inconsistencies between the different boards caused some difficulties when trying to make square and secure corners of the box but the many brackets on the inside and outside edges of the box eventually pulled all the sides tightly together, resulting in a solid and secure box.

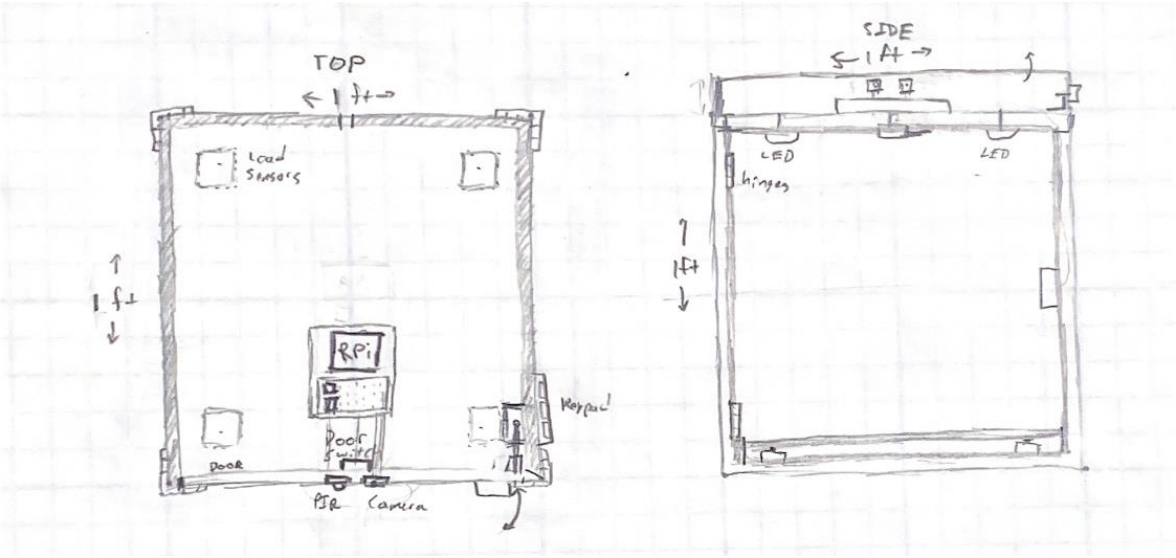


Figure 30: Box Design Sketch

Figure 31: Box Picture

6.3 Application Subsystem

The application for KeepSafe will be used to give the user full control over KeepSafe. The most difficult part of the app was choosing which programming language and structure should be used to create it. To allow all devices to access the app without issue, it was immediately decided that the application should be web based. After this decision was made, an ASP.Net structure was decided on for the backend, due to the team's prior experience with the C# language. During the initial implementation stage though, it was found that the Apache web server that the application runs on works best with FLASK based apps. This is because FLASK apps are written in Python which the Raspberry Pi is most suited for. What seemed like a limitation initially actually saved the team future troubleshooting, since translation was longer necessary between a C# based app and the Python based components in the electrical subsystem. With both systems running on Python, communication between the two was simplified immensely. Once FLASK was decided for the backend of the application, different HTML files were written for the front end, to create buttons that allow the user to navigate between the different pages and controls. These controls include unlocking KeepSafe, checking the weight of the package, seeing the GPS location, and monitoring the camera. See appendix 10 for screenshots of the app user interface.

7. Design Changes Since FDR

Since the fall, many changes have been made to KeepSafe, mostly in regard to its physical construction. These changes revolved around one idea, simplifying the final prototype. After much deliberation among the team, Professor Aslani, and the machine shop, it was decided that KeepSafe should move from a large, two compartment device, to a single compartment, 1’ x 1’ x 1’ foot prototype. By shrinking down the size of this product, the team was able to cut costs significantly, while still showing the proof of concept. The materials used to create the device also changed, with costs being the biggest factor again. Initially, it was thought that a metal exterior should be created, to add weight and improve security. While a final product may utilize metal in the future, plywood was chosen for this prototype, as it was cheap, and allowed the team to easily move it, which proved extremely useful when testing the GPS module outdoors. A plexiglass lid was also added to the top of this device, so that during the final demonstration, the audience could see what is “under the hood.” This lid design also made it extremely easy for testing and troubleshooting. It should be noted that even with these physical changes, the circuitry inside the box did not change, and all the functionality that was created in the fall was easily applied to this new body.

Another major change to the overall design was the finalization of the locking mechanism. In the fall, the final design for this mechanism was not decided upon. The team knew a servo would work best rather than a solenoid, but how the servo would drive the lock was still undecided on. Upon returning for the winter semester, the first design was a simple armature system connected to the servo, which would then turn into a designated slot, like a how a gate locks for a yard. While this design would have been extremely easy to implement, this idea was not continued after it was brought to the team’s attention that it would be extremely easy to break the servo if the armature was directly connected to it. If the armature were to break, so too would the servo. The team returned to the drawing board, and instead developed a rack and bolt type system. With this system, a gear would be attacked to the servo, with a rack moving left or right depending on the rotation of the gear. This rack would sit along a rail, with a cylinder at the end which would be inserted into the side of KeepSafe. It is important to note that this system resides on the door, so that the bolt can be driven into the side wall. This was the final design used, and the gear, rack, and rails were all 3D printed, along with a mount for the servo.

Although not necessarily changes, it is important to note that the last two modules were finally connected to KeepSafe, the gyroscope and the GPS system. While these two devices were accounted for in the planning last fall, they were not implemented until this part of the fabrication progress. Both were successfully implemented into the device, and connected seamlessly with the previous components, and the application. These two devices add to the overall security of the device, allowing the user to track the box’s location, and rest assured that if meddled with, KeepSafe can lock their possessions. Bright interior lights were also added during this part of fabrication, further adding value by illuminating the inside of KeepSafe when used at night. With all these new modules, the application was also adjusted, so that they could be utilized, and their information displayed in an easy-to-understand manner.

8. Module and System Tests

8.1 Electrical Subsystem Tests

8.1.1 PIR Sensor Tests

When there is motion immediately around KeepSafe, the system must recognize this and trigger the camera module correctly. The maximum range of the PIR sensor is 20 feet which meets the desired range for KeepSafe applications. However, reducing the range is preferred to conserve power consumption. The team will find solutions to accomplish this and perform range testing through the design process to get it closer to this specification.

8.1.2 Servo Tests

The first device that is utilized to ensure package security is a servo, which will drive the locking mechanism for KeepSafe. A servo's position is often determined by how long the duty cycle of a "high" pulse is. The longer the pulse width, the more it will turn, usually up to 180°. Although these specifications are given in the datasheet, these specifications can sometimes be misleading or wrong, so it will be important to determine different exact pulse widths for the team's specific servo to determine their corresponding angles. These tests will be done using the Raspberry Pi's GPIO controls to set the test signal's frequency and the high signal's period. The test signal will be set to a frequency of 50Hz and connected to the servo will be a single arm so that the degree change can be easily measured using a protractor. These tests will directly impact the final design of the locking mechanism and must be easily replicable in case any changes need to be made to this design in the future.

Testing must also be done regarding when the servo receives the driving signal. The goal is to secure the package and lock KeepSafe as soon as a package is detected and the door is closed, minimizing any chance of a robbery. The goal is to have KeepSafe be 100% locked 3 seconds after a package is detected and the door is closed. Minimizing this number even further will only be beneficial. To test this, the RPM of the servo must be measured, again using the GPIO controls of the Raspberry Pi. Although this value is also typically given in data sheets, it will be important to time how long the servo takes to turn 180°.

8.1.3 Gyroscope Tests

The PIR is used to detect motion around the box to wake up the RPi and operate as designed. However, outside of taking and saving a picture for security purposes, the PIR sensor isn't also fully tied to the functionality of the security system (activating the GPS module) since someone coming and trying to steal the entire box is much rarer occurrence. Therefore, the activation of the GPS module is determined by the output of the gyroscope sensor which will monitor any physical movement of the box. This sensor will be tested independently to observe the amount of movement needed to trigger it and then in the full box. To move the large two compartment box, a significant amount of force will be needed so the team is confident with the sensors ability to detect movement. This will be confirmed through testing which consists of moving the box with

various abouts of force and observing the output of the sensor to make sure that it will detect any potential attempts at stealing the entire box and can activate the GPS module. The gyroscope must activate the GPS module if the orientation of the gyroscope changes 45° or more.

8.1.4 GPS Tests

The last device that will be used for package security is the GPS module. There are many factors that can be tested and considered with this device due to its complexity. The first and potentially easiest factor that can be tested is connection time. According to the datasheet, from a "cold-start" (meaning it has not been connected at all or in a while), the GPS can take up to 27 seconds to calculate its location. With the implementation of the sleepy pi, if the GPS system is not constantly connected, it will be important to take this connection time into account, since every activation will be considered a cold start. The cold start time must be tested multiple times, by simply starting up the device with raspberry pi, starting a timer, recording how long it takes to establish a connection, turn it off, and then repeat the process. During this process, it will also be important to determine how long it takes to actually "cool down", essentially determining at what time between connections does it require that full 27 seconds. This method will also generate a cool down rate, that can be used to determine if sequential pings can be used to keep the device from cooling down in the first, and if so, how long between pings. All these factors can be extremely useful during implementation. Another aspect of this device that can be tested is its actual accuracy of its coordinates. Again, according to its data sheet, it has an error rating of 2.5 meters. To test if this number is accurate for this specific device, its coordinates can be compared to a GPS software on one of the team member's smartphones. If both devices are working as expected, their calculated positions should be the exact same. If tested multiple times, a more accurate error value can be determined in meters for the team's specific device. The last major aspect to test for this device is the antenna. Since the antenna is not extremely powerful, when placed inside or not near a window, its functionality can become severely limited. KeepSafe is designed to be outside, so this issue may not be extremely detrimental, but it will still be important to set up different test scenarios where the antenna can be impacted. Some test scenarios include placing the antenna outside, placing it inside near a window, placing it inside away from a window, placing it outside but under a roof or awning just to name a few. These conditions may affect the cold-start time as previously mentioned, so those tests will also need to be conducted in these different conditions as well. By conducting all these tests, the team will be able to determine the optimal conidiations on where to place KeepSafe and determine what kind of algorithm will be required to maximize the GPS's effectiveness on package security.

8.1.5 Door Switch Tests

The SPDT switch will be connected to KeepSafe's door and will determine whether the door is opened or closed. Because this is the first device to be activated, it is important to ensure that it is fully functional. Once wires are soldered to the device, continuity tests will be performed using a DMM to ensure that the connections are solid, and that the device

was not damaged by the heat of the soldering iron. The next test will focus on debouncing the switch, and what parameters are required to do so. With some smaller push buttons for example, it can be very easy for a user to think they have only pushed the button once, when in fact they have been pressed multiple times. For this switch, a python script will be used in conjunction with a simple circuit to determine if debouncing is necessary for this device. By connecting the switch to a 3.3V GIPO output in the RPi, in series with a resistor, the team will be able to gauge the sensitivity of the switch. The result of these tests will ensure that each time the switch is clicked, there is only one change in current, and not multiple.

8.1.6 Camera Tests

The next device is a Raspberry Pi camera module that is triggered by the PIR sensor. The camera module can be tested simply by ensuring its operation. This can be done both through the app and through separate python scripts. With the app, the images are stored in the "static" file of the server, so that they can be displayed within the html. It is important within the html coding not to try and force a resolution greater than 8 MP or else the images will appear distorted.

8.1.7 Load Cell Network Tests

The four load cells in conjunction with the HX711 amplifier are responsible for physically detecting the package and determining its weight. For these devices to work, a reference unit must be set in the code so that discernable values can be measured. A correct and precise reference unit is important to achieve the accurate weight of up to 5% error. Testing will be done to fine tune this reference unit to maximize accuracy. To complete this test process, the sensors are first zeroed with the plexiglass platform on top. Then, pre-calibration measurements are taken with an object of know weight. For the test, 200 pre-calibrations readings will be acquired and the average of them will be calculated. Finally, the averaged value will be divided by the known weight. This provides you with the reference unit which is added into the code. This reference unit value should be consistent for all tests but there is some slight degree of error. Reference unit calculations will be taken 5 different package sizes and the average will be taken to ensure the most accurate readings from the load sensor network for all cases. Using the procedure discussed above, the reference unit is calculated for each package size and the average value is used for the final reference used in the code. To confirm the subsystem meets the specifications, each package size is placed onto the sensors and tested with the final reference unit value to ensure it has less than 5% error from its actual weight from a highly accurate scale. The network must also detect anything over 3 grams, which about the weight of an empty envelope that the team measured.

8.1.8 Keypad Tests

The manual unlock of the box is important for members of the household that do not have access to the app. The functionality of this feature will be tested by inputting correct and incorrect combinations and confirming the correct response of the system. Currently, there will be a pre-determined passcode loaded but the team will

consider implementing an option to allow the user to personalize the code. These additional features will be complemented with more testing to confirm that it is working properly, and these is no potential weak points in the design flow. Debouncing may also be required for each button of the keypad, and similar tests as the door switch will also be conducted.

8.1.9 Interior Light Tests

Testing will be done on the interior lighting configuration to show that it properly illuminates the entire interior of the compartment during nighttime hours. Adjustments will be made to the design if the given lighting configuration does not accomplish this. Additionally, testing needs to be done on the response time of the lights to show that they turn on almost instantaneously following the door bring opened and turning off when the door is closed. This is important for us to confirm that the compartment will be illuminated 100% of the time packages are being placed inside but also that the lighting system following the logic flow of the system exactly.

8.2 Mechanical Subsystem Tests

8.2.1 Locking Mechanism

The physical strength of the locking mechanism must be testing due to its importance to overall package security. Once in a locked position, various amounts of force will be applied onto the door to pull it open. This will be measure with a digital or analog force gauge (whatever is accessible) to ensure that the physical strength of the door locking mechanism meets the specification of 100 pounds of force. (See also: 7.1.2) Additionally, consistent locking will be tested to ensure that the bolt of the locking mechanism is inserted into the drilled hole every time.

8.3 Application Test

The main testing for the app will mostly be focused on response time. No consumer enjoys waiting for a webpage to load, so it will be important for KeepSafe's app to have a response time under one second when navigating through the app. To test this, it will simply be a matter of clicking to the next page and timing how long the app takes to communicate with the server and return the correct homepage. If the correct page is returned in under a second, this would be considered a success.

9. Applicable Standards

9.1 IEEE Std 1888.4-2016

The first standard KeepSafe must follow is IEEE Std 1888.4-2016, the *IEEE Standard for Green Smart Home and Residential Quarter Control Network Protocol*. This document outlines how different smart devices connected to a home network must follow "protocols for measurement and control networks for home and residential quarters, so that they can achieve green, smarter functions." Devices that must follow this standard include but are not limited to: Alarm Sensors, Surveillance Cameras, Door Locks, and Door Lock Controllers. KeepSafe encapsulates all these devices, with a PIR sensor to trigger a

potential alarm, a surveillance camera to capture pictures of anyone in front of KeepSafe, a servo to lock KeepSafe, and the Raspberry Pi controlling all of these devices. This standard does not give specific ranges on how devices should be green, but instead outlines testing protocols to determine if the device is functioning properly within the GSHN, Green Smart Home Network. There are two kinds of testing that are outlined in this document, conformance testing and interoperability testing. Conformance testing of a device includes selecting the proper testing process and parameters, testing the device by itself and within the network, and analyzing the results of the test to provide diagnostic information on the device. Interoperability testing "focuses on verifying the interconnection of different products by different vendors." For KeepSafe, the main testing would focus on the connection between the router and KeepSafe to ensure their interconnection. It is important to note though that interoperability testing must be done with other devices that are already certified in following this standard.

9.2 IEEE 802.1AE-2018

The second standard KeepSafe must follow is IEEE 802.1AE-2018, the *IEEE Standard for Local and metropolitan area networks-Media Access Control (MAC) Security*. This standard is important to the implementation of KeepSafe because the Raspberry Pi will have a MAC address so that it can connect with the dedicated servers to store the information of the user's package. Also, this standard defines the security protocols that are used to protect the confidentiality of the user which in this case is the Raspberry Pi in the KeepSafe. The cipher suites defined in the standard allow more than 2^{32} frames to be protected with a single Security Association Key (SAK), which will provide added security to the data transferred. This secure data transfer will provide added security to KeepSafe and as well make each individual unit recognizable if long distance tech support is needed after the customer has already purchased the product.

9.3 ISO/IEC 10918-1:1994

The third standard KeepSafe must follow is ISO/IEC 10918-1:1994, *Information technology — Digital compression and coding of continuous-tone still images: Requirements and guidelines*. This standard is important for KeepSafe because it defines how the image that the camera takes will be stored and transferred which will be a JPEG file. This JPEG file will determine how the image will be viewed and stored within in the server. The user will then receive this data type through the app to view. This is important to define because for the app development the code needs to know what type of file it is looking for to display to the user and in the case of photo taken from the camera on KeepSafe it would be a JPEG file which is defined in this standard.

9.4 ASTM f1577-05(2019)

The fourth standard KeepSafe must follow is ASTM f1577-05(2019), *Standard Test Methods for Detention Locks for Swinging Doors.* While detention locks may seem extreme, they are still very much applicable to the level of security KeepSafe hopes to achieve. Detention locks are designed to withstand abuse from criminals, and so too is KeepSafe. KeepSafe's design will also incorporate a swinging door, making this standard even more applicable to our product. While at this current time the exact locking mechanism for KeepSafe has not been designed, it is important during the design phase to consider these standards for testing to help eliminate

problems that may occur later in the engineering design process. This document lays out multiple test case scenarios for these types of locks, including impact tests and remote operations tests. The maximum impact outlined in this document 271.2J, and the test is considered successful if the lock remains engaged and the door panel remains closed. (For reference, a human punch ranges from 100-450J). There is also remote operation test, which "evaluates the capabilities of remote operation locks to continuously function under normal operating cycles." This test outlined that a remote lock must complete 1,000,000 cycles before it is considered to have passed the test.

9.5 IEC 63180-06: 2020

The seventh standard KeepSafe must adhere to is IEC 63180-06: 2020, *Methods of measurement and declaration of the detection range of detectors – Passive infrared detectors for major and minor motion detection.* This standard specifically applies to the PIR sensor that is being used to detect any and all motion in front of KeepSafe, and outlines test procedures to verify the detection performance of the device.

9.6 GPS Standard Positioning Service Performance Standard 5th Edition

The eighth standard applicable to KeepSafe is the Global Positioning System Standard Positioning Service Performance Standard 5th Edition. The team was unable to find a numeric title for this standard, so it will be referenced as the GPS-SPS PS. Because KeepSafe is equipped with a GPS module, it is important to follow these standards set by the US government. It outlines that devices that have a horizontal accuracy of 3m or less and a vertical accuracy of 5m or less, both 95% of the time are considered well designed and fully functional. While the team expects KeepSafe's vertical value to remain constant, the team is holding KeepSafe' horizontal accuracy to be even more accurate than outlined in the GPS-SPS PS at 2m or better. The GPS-SPS PS also outlines the frequencies that must be used for all civilian GPS modules, so it is important to tamper with the device and potentially change its operating frequencies.

9.7 IEEE Std 2700™-2014

The ninth standard KeepSafe must adhere to is IEEE Std 2700™-2014, *IEEE Standard for Sensor Performance Parameter Definitions.* This standard applies to the gyroscopic sensor used in KeepSafe that activates the GPS module. This standard outlines operating conditions for the gyroscopic sensor and a variety of other sensors as well. This standard aims to outline what conditions must be met for a gyroscopic sensor to be considered fully operational, including but not limited to: dynamic range, sensitivity, noise, current consumption, cross-axis sensitivity, linear acceleration sensitivity, and many more guidelines. KeepSafe relies on and must adhere to this standard, to ensure its overall safety if tampered with.

10. Summary and Conclusion

Throughout the year and a half capstone sequence, the KeepSafe team has achieved our goal of having a working product. For the electrical subsystem, we completed a prototype with all the components connected to the Raspberry Pi, integrated into the physical box design, and compatible with the companion app. The components that are connected are the load cells, the HX711 amplifier, the camera, the door switch, the servo, the interior light, the 4x1 keypad, the

PIR motion sensor, the interior lights, the gyroscope module, and the GPS module. All the connected components were tested in a logic flow program written in python. The electrical subsystem is split into two smaller subsystems: the package detection subsystem and the security subsystem. The package detection subsystem utilizes the PIR sensor, the camera, the door switch, the interior lights, the servo, and the HX711 amplifier connected with the load cells. The package detection development and integration went smoothly and was a process the team had sorted out back in the fall of 2021. The security subsystem utilized the PIR motion sensor, the camera, GPS module, the gyroscope module, and the servo. The security subsystem was all new for the spring of 2022 but provided some issue to the team in the initial steps. There was an issue with interpreting the output data of the GPS module so that the python code could read it, the issue was resolved when a different function was used to read the byte data. Other than the GPS issue the rest of the security system worked as intended and was easy to implement into the overall design. One final subsystem of the electrical subsystem the team initially had but could not implement was the power consumption and management subsystem. The team planned to utilize a sleepy pi to manage power consumption through a connected battery so the battery life could last multiple days. Due worldwide supply shortages, the team could not obtain a sleepy pi in time to integrate into the first prototype of KeepSafe. As a result, the team resorted to connecting the raspberry pi directly to a power outlet. Since it was not the team's first choice to have the KeepSafe directly plugged in, the second iteration of KeepSafe is planning to have its own battery and power management system.

For the mechanical subsystem, the overall prototype design ended up as a 1' x 1' x 1' compartment with the false floor and a small compartment on top to hold the electronics. The components of the locking mechanism were 3D printed and mounted onto the box. Once the box was constructed, the team was able to integrate all the components into the box and laser etch the KeepSafe logo onto the door. The final prototype ended up being a much smaller box than initially expected because the initial design was too large to be a practical prototype that could be carried and displayed around. The team was fortunate that the process went smoothly and were able to create an enclosure that functions as a package box but also show off the electrical components.

For the application subsystem, the team created the Apache server, the html code that the server reads, the Python code that the communicates with the server, the web-based app with package status updates and images from the Raspberry Pi camera, and communication with the Raspberry Pi. The application was able to be successfully completed and control the function of KeepSafe. The team was able to connect all the functions outlined in the electrical subsystem to the application and control all major components of KeepSafe. Some of the issues with application came with integrating the code developed for the electrical subsystem and there was an issue with some of the logic. These issues were resolved when the team switched some of the action functions to Boolean values. At the end of the day, the KeepSafe team worked tirelessly through the day and night to complete the now working prototype. Due to the team's work and effort on KeepSafe, the world is now safer from package theft, one delivery at a time.

11. References

1. Fox, H. (2021, February 10). *2020 Package Theft Statistics | Swiftlane Access Control*. Www.swiftlane.com; Swiftlane. https://www.swiftlane.com/blog/delivery-management-and-package-theft/
2. *2020 Package Theft Statistics Report*. (2020). C+R Research. https://www.crresearch.com/blog/2020-package-theft-statistics-repor
3. https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/
4. https://www.raspberrypistarterkits.com/guide/raspberry-pi-accelerometer-gyroscope/
5. https://www.amazon.com/HiLetgo-GY-NEO6MV2-Controller-Ceramic-Antenna/dp/B01D1D0F5M/ref=sr_1_3?dchild=1&keywords=Neo+6M+GPS+Module+u-blox+B01D1D0F5M&qid=1635621951&sr=8-3
6. https://spellfoundry.com/product/sleepy-pi-2-usb-c/
7. Membrane 1x4 Keypad + Extras : ID 1332 : $2.95 : Adafruit Industries, Unique & fun DIY electronics and kits
8. https://www.raspberrypi.com/products/camera-module-v2/
9. https://rcracing.ro/servomecanisme/2519-high-torque-330-blue-servo.html
10. https://makezine.com/2015/02/02/microsoft-windows-support-raspberry-pi-2/
11. Hardware Engineer Salary in Washington, District of Columbia | Salary.com
12. 50kg Loadcell Bracket versionF by patrick3345 - Thingiverse
13. White LED Backlight Module - Medium 23mm x 75mm : ID 1622 : $2.50 : Adafruit Industries, Unique & fun DIY electronics and kits

12. Appendices

Appendix 1. Timeline Estimation and Milestones

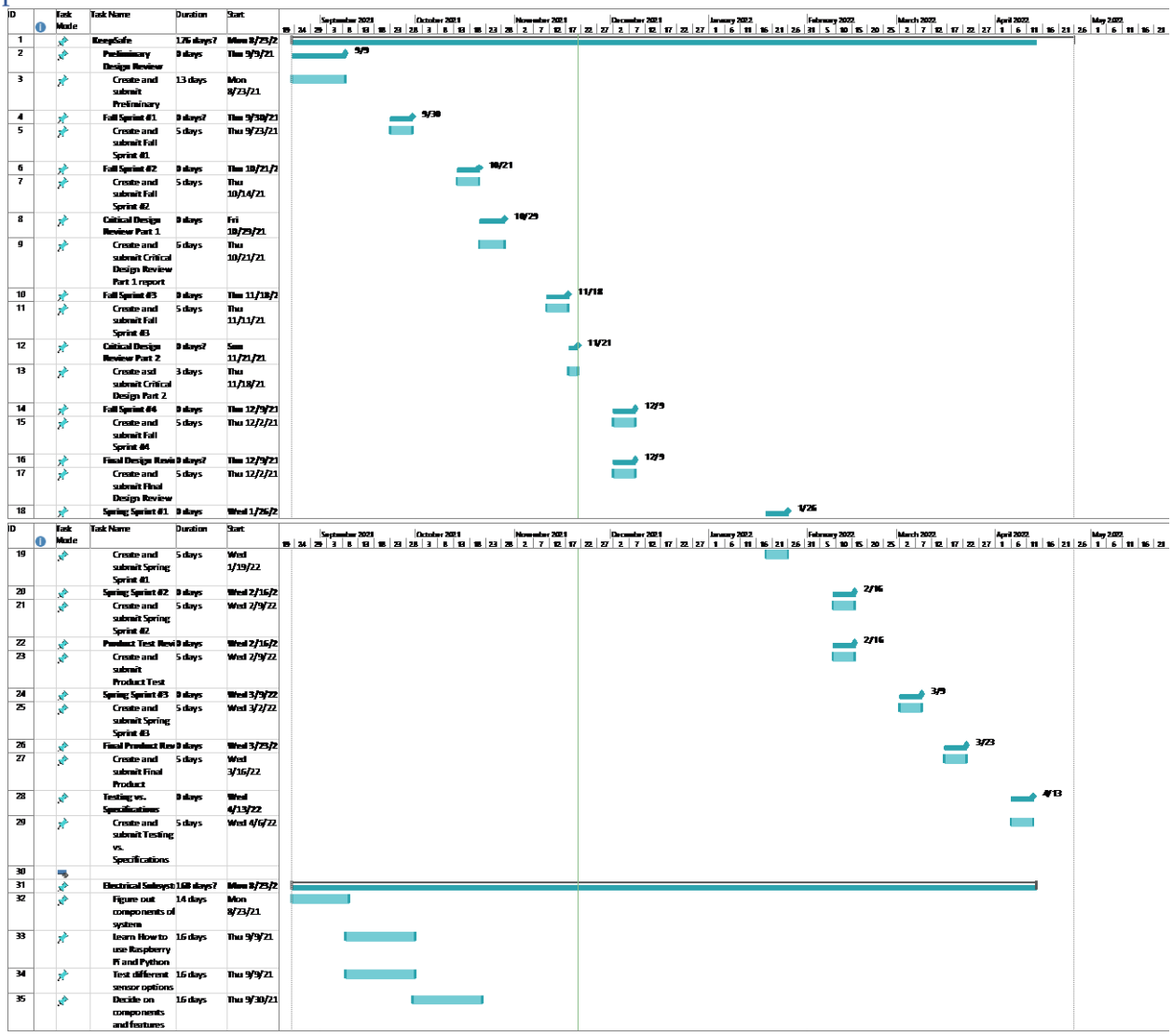


Figure 32: Overall KeepSafe Timeline

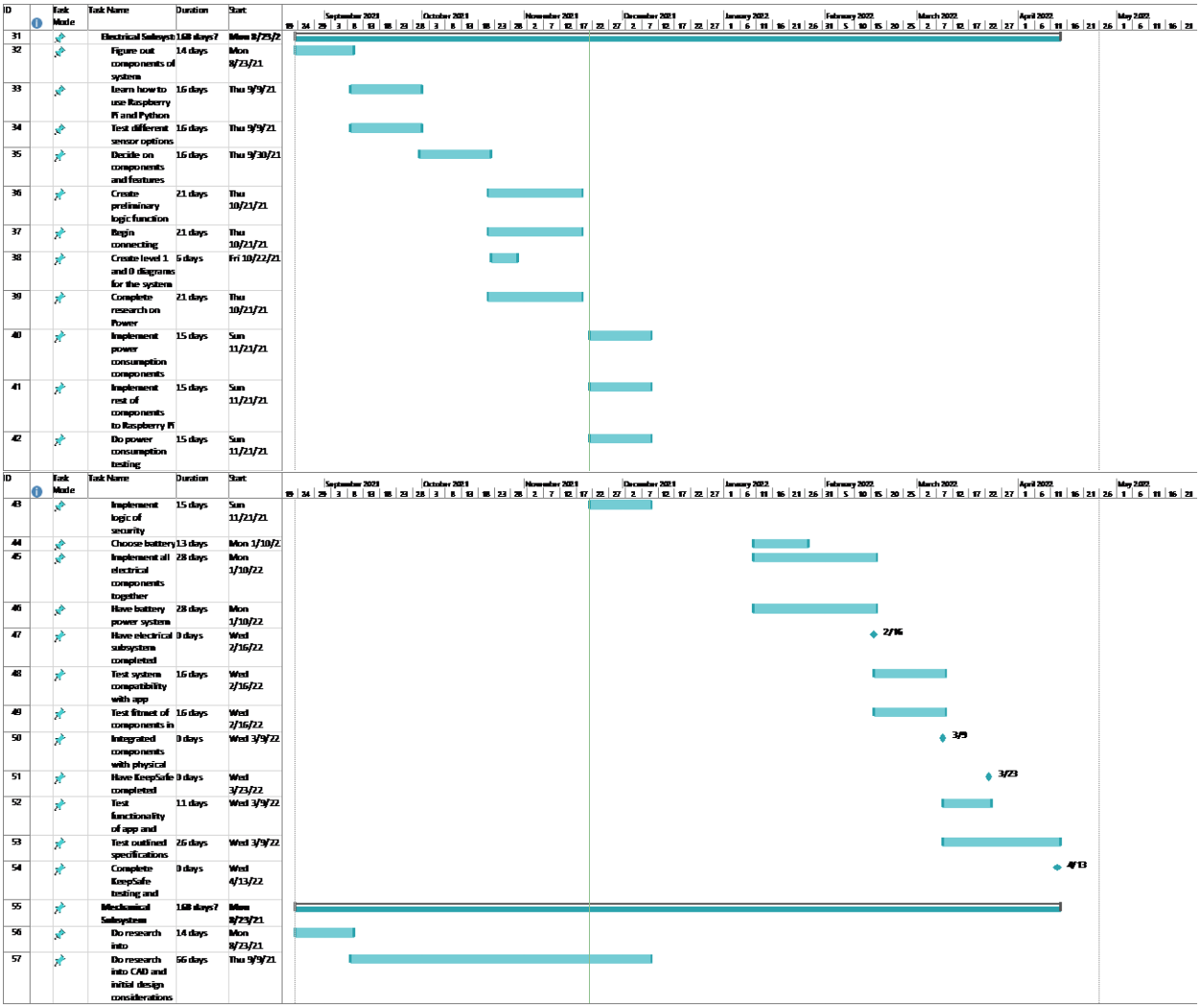


Figure 33: Electrical Subsystem Timeline

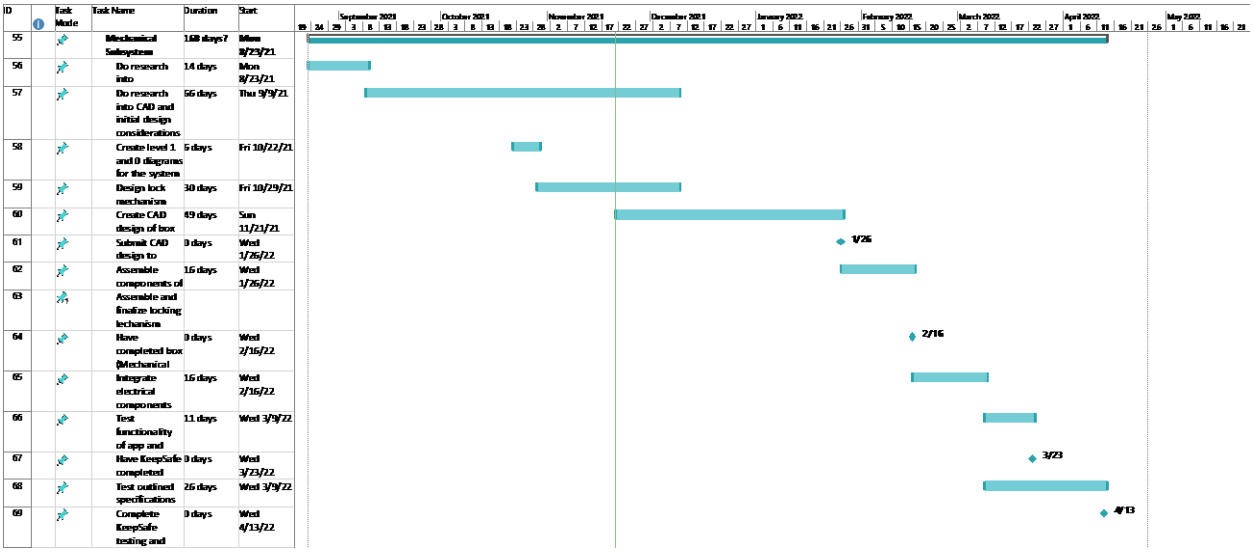


Figure 34: Mechanical Subsystem Timeline

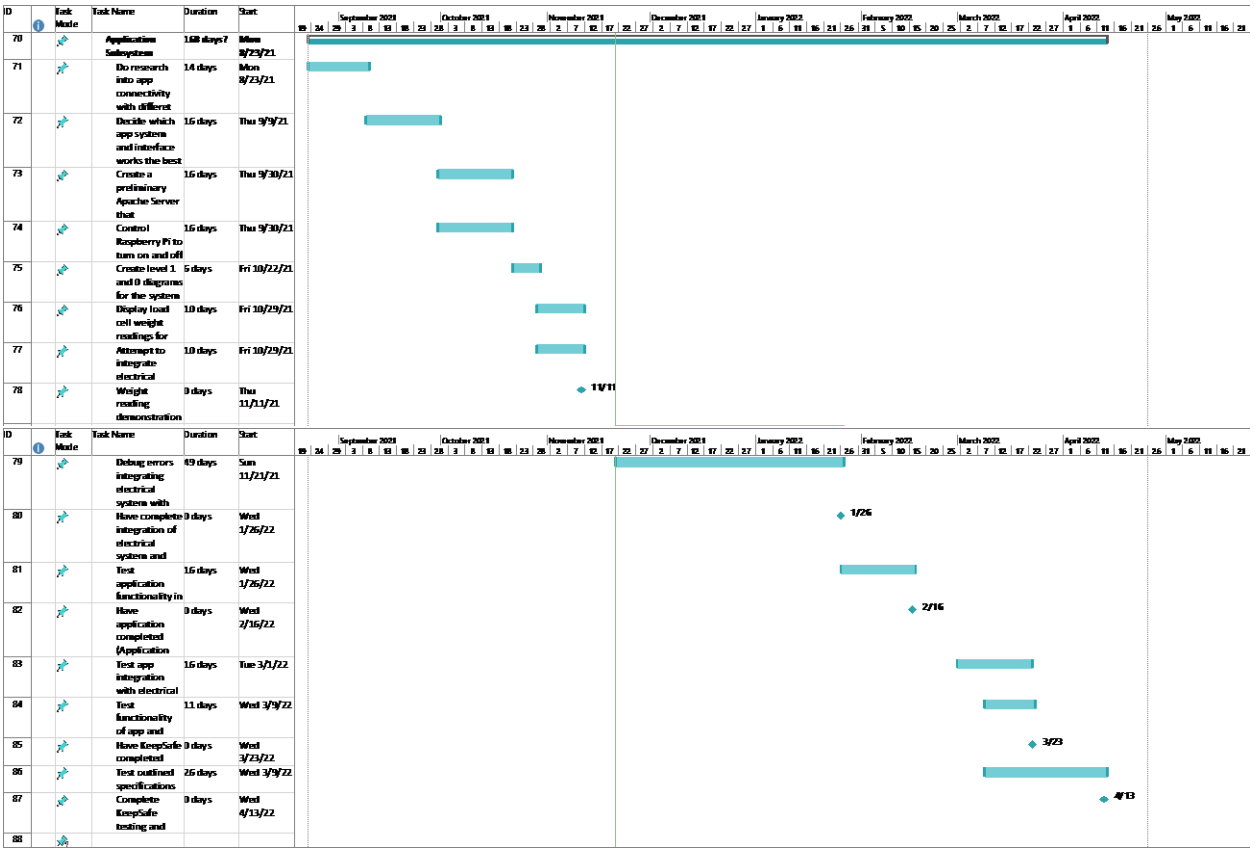


Figure 35: Application Subsystem Timeline

Appendix 2. Economic Analysis

Engineering / Development Operation	Personnel	# of personnel	Hourly Rate	Estimated # of working hours	Salary
	Project Manager	1	$66.00	80	$5,280
	Design Engineer	2	$57.00	160	$18,240
	Hardware Engineer	2	$48.00	160	$15,360
	Software Engineer	2	$40.00	160	$12,800
	Test Engineer	1	$36.00	53	$1,920
	Technical Writer	1	$30.00	27	$800
			Total Labor	640.00	$54,400
		(typically 2-3.3)			
	Salary to contract multiplier	2.5		Cost to Contract	$136,000
	Cost of all parts and external services needed to produce the final, pre-manufacturing prototype				
		Parts total	$321	(from BOM page below)	
		Machine shop	$1,000	Estimate	
		PCB fabrication and population	$1,000	Estimate	
		Total parts & external services	$2,321		
	Pass-through fee	5%	$116		
	Total Prototype Cost	$138,437			

Figure 36: Economic Analysis 1

- Engineer salaries for the Washington, DC region were used [11]
- The total development time was estimated at 2 weeks
    - The current KeepSafe team was working part time over multiple semesters while a full team of engineers working full 40-hour weeks will be able to do this development quicker.
- Each role had a multiplier used for the approximated time they would be needed for the overall 2-week project sequence
    - Engineer multiplier (design, hardware, software): 75%
    - Test Engineer multiplier: 66%
    - Technical Writer multiplier: 33%

Production Cost - Labor	Manufacturing process development & verification		Manufacturing Time (weeks)
	# of hours required	120	3
	Hourly rate	$20.00	
	Total mfg verification cost	$2,400	
	Software testing		Software Testing Time (weeks)
	# of hours required	120	3
	Hourly rate	$15.00	
	Total sotware testing cost	$1,800	
	Multiplier	2	
	Total Production "Labor" Cost	$3,600	

Production Cost - Non Labor	Parts cost		Percent Reduction from Prototype Parts Total: 40%
	# of production units	1000	Estimated manufacturing price per unit: $ 192.62
	Parts cost per unit	$192.62	
	Total production's part cost	$192,618	
	Documentation & Packaging Cost		Production units: 1000
	# of production units	1000	
	Printing & packaging cost per unit	$30.00	
	Total printing & packaging cost	$30,000	
	Total Production "Non Labor" Cost	$222,618	

Figure 37: Economic Analysis 2

- The manufacturing and software testing time for the product was also estimated to be around 3 weeks due to the number of units being produced

- The total price from the BOM per unit was reduced by 40% due to savings from buying in bulk, utilizing product specific PCBs, and streamlined manufacturing process
- The number of production units for these calculations was set at 1,000

	Total Production Cost	**$226,218**
	Overhead/Administrative Cost Rate	**40%**
	Profit (fee) Margin	**20%**
	Total Production Cost with Overhead & Profit (fee) Margin	**$380,046**
	Total Production Cost per unit	$380.05
	Total Cost of the Project (Prototype + Production cost)	**$518,483**
	Total Project Cost per unit	**$518.48**
Estimation of Wholesale & Retail	Wholesale Multiplier	**20%**
	Wholesale Price (per unit)	**$622.18**
	Retail Multiplier	**50%**
	Retail Price (per unit)	**$777.72**

Figure 38: Economic Analysis 3

- A retail price of around $778 is a little more on the pricier side but considering the number of features and smart capabilities KeepSafe possesses it would be worth it for the consumer
- Also, further analysis can be done to find cheaper materials and deduct costs in other sectors to eventually lower the price
- Compared to other products on the market, this product offers much more value

Appendix 3. Part List

Revised BOM from FDR

Table 8: Hardware Bill of Material

Part Name	Description	Manufacturer	Mfg. Part #	Supplier	Supplier Catalog #	Cost ($)	Qty	Datasheet
Tower Pro MG995R High Torque Servo	High Torque Micro Servo to control door lock	Tower Pro	MG995R	Amazon	B09BZTRSDP	$10.99	1	link
RPi computer	Raspberry Pi 4B/2GB	Raspberry Pi	Raspberry Pi 4B/2GB	Digikey	1690-RaspberryPi4B/2GB-ND	$35.00	1	link
Load Sensor	Load Sensor Network for Package Detection	Mantech Electronics LTD	ALX-50KG	Amazon	B097T3SX6W	$10.99 total	4	link
HX711 Amplifier	Amplifier for load sensor network	AVIA Semiconductor	HX711				1	link
Membrane 1x4 Keypad	1x4 matrix Keypad for manual unlock of box	Adafruit	100535	Amazon	B00OKCRZ70	$6.48	1	link
White LED Backlight Module	LEDs to light up the inside of the compartment (23mm x75 mm)	LuckyLight	KWB-R7323W/1W	Amazon	B00R5CDHNI	$3.49	2	link
Door switch	Switch to recognize when the door is opened	Panasonic Industrial Devices	ASQ10237	Mouser	769-ASQ10237	$7.58	1	link
Neo 6M GPS Module	GPS to give location of box	HiLetgo	3-01-1087-UK-1PC	Amazon	B01D1D0F5M	$8.59	1	link
HC-SR501 PIR Sensor	Motion Sensor to start code	HiLetgo	3-01-0120	Amazon	B07KZW86YR	$7.29	1	link

Raspberry Pi Camera Module 2	Camera to take picture of Box user	Raspberry Pi	RPI-CAM-V2	Amazon	B01ER2SKFS	$37.44	1	link
GY-521 MPU-6050 MPU6050	3 Axis Accelerometer Gyroscope Module	HiLetgo	3-01-0122-AB	Amazon	B01DK83ZYQ	$5.99	1	link
Stainless Steel L Bracket (pack of 20)	0.78 x 0.78 inch, 20 x 20 mm	Iziusy		Amazon	B07QG19QMQ	$7.99	1	n/a
Magnetic Door Catch	Hold Door in place (Pack of 2)	Jiayi		Amazon	B09331WDKH	$7.99	1	n/a
Plexi Glass Sheet	Plexi Glass Sheet for top of box (12" x 12" x 3mm)	Lesnlok	30265802	Amazon	B09P74K7BR	$16.68	1	n/a
Gripper Feet	Non-slip pads for bottom of box(1 in. round, pack of 16)	SlipStick	CB147	Amazon	B08C373VBW	$6.99	1	n/a
RPi GPIO Breakout Expansion Board + Ribbon Cable	Expansion Board for easy connection to Raspberry Pi	Kuman	SC05-New	Amazon	B0761NYF6Y	$10.99	1	n/a
Marine-Grade Plywood Sheet	Panels of box (12x12x1/4)			McMaster-Carr	1125T411	$8.88	1	n/a
Cdx-Grade Plywood Sheet	Fake Floor for Load Sensors (12x12x3/8)			McMaster-Carr	1125T511	$6.01	7	n/a
Galvanized Steel Corner Bracket	Main corner bracket supports			McMaster-Carr	1556A44	$2.45	16	n/a

	(2.5x2.5x1/2)							
Phillips Screw (No. 6 size)	Screws for larger brackets (pack of 100, length: 0.375)			McMaster-Carr	90294A146	$7.93	1	n/a
Surface Mount Hinges	Door Hinges (holes: 1x3/8)			McMaster-Carr	1603A3	$1.85	4	n/a
Hinge Screws (No. 2 size)	Hinge screws (pack of 100, length: 0.25)			McMaster-Carr	92114A077	$7.47	1	n/a
Washers	0.156/0.312			McMaster-Carr	92141A008	$1.20	1	n/a
Small Corner Brackets	7/8x7/8x5/8			McMaster-Carr	1556A64	$2.33	2	n/a
Nylon Pan Head Phillips Screws	Screws to mount PIR and Camera (pack of 100, M4 x 0.70 mm)			McMaster-Carr	92492A725	$11.76	1	n/a

Visual Studio Code

- Subsystem: Server/App & Electrical
- Version: 1.61
- Source: https://code.visualstudio.com/?wt.mc_id=DX_841432
- Cost: Open Source

Python

- Subsystem: Sever/App & Electrical
- Version: 3.9
- Source: https://www.python.org/download/other/
- Cost: Open Source

HTML

- Subsystem: Networking
- Version: 5.3
- Source: https://www.w3.org/TR/2021/NOTE-html53-20210128/
- Cost: Open Source

HX711 Library

- Subsystem: Electrical
- Version: n/a
- Source: https://github.com/tatobari/hx711py
- Cost: Open Source

Flask Library

- Subsystem: Server/App
- Version: 2.0.2
- Source: https://pypi.org/project/Flask/#files
- Cost: Open Source

Picamera Library

- Subsystem: Server/App & Electrical
- Version: 1.13
- Source: https://picamera.readthedocs.io/en/release-1.13/
- Cost: Open Source

Jinja2 Library

- Subsystem: Server/App
- Version: 3.1.1
- Source: https://pypi.org/project/Jinja2/

- Cost: Open Source

MarkupSafe Library

- Subsystem: Server/App
- Version: 2.1.1
- Source: https://pypi.org/project/MarkupSafe/
- Cost: Open Source

Flask-googlemaps Library

- Subsystem: Server/App
- Version: 0.4.1
- Source: https://pypi.org/project/flask-googlemaps/
- Cost: Open Source

Dynaconf Library

- Subsystem: Server/App
- Version: 3.1.7
- Source: https://pypi.org/project/dynaconf/
- Cost: Open Source

SolidWorks

- Subsystem: Mechanical
- Version: 2022
- Source: GWU license
- Cost: N/A

Google Maps API Key

- Subsystem: Server/App
- Version: NA
- Source: https://developers.google.com/maps/documentation/javascript/get-api-key
- Cost: Free

Appendix 4. Qualification of Key Personal

Sean Letavish's qualifications for this project include all prior coursework, including ECE 1120 (C Programming) ECE 2110 (Circuit Theory), ECE 1125 (Data Structures), ECE 2115 (Engineering Electronics), ECE 2140 (Logic Systems), ECE 2210 (Circuits, Signals, and Systems), ECE 3125 (Analog Electronics Design), ECE 3135 (Digital Design with FPGAs), ECE 3220 (Digital Signal Processing), and ECE 3410 (Communications Engineering). He also has much experience with C# and has a general understanding of C++ due to its similarities to C and C#. Sean has also done prior work with 3D modeling, in both Blender and AutoDesk Inventor. He also has prior carpentry skills, which will prove useful in finalizing the overall design of KeepSafe.

Jaret Williams's qualifications for this project include all prior coursework, employment, and personal experience. Some relevant electrical engineering coursework includes: ECE 2110 (Circuit Theory), ECE 2115 (Engineering Electronics), ECE 2140 (Logic Systems), ECE 2210 (Circuits, Signals, and Systems), ECE 3125 (Analog Electronics Design), ECE 3135 (Digital Design with FPGAs), ECE 3220 (Digital Signal Processing), and ECE 3410 (Communications Engineering). He has extensive experience with circuits, sensors, and microcontrollers from many electronics projects. His first in 2013 when he made the first prototype of KeepSafe and won the grand award in the regional science fair. From there, he expanded his engineering knowledge in the STEM Academy growing up and then the many relevant courses here at GW. This past summer, he was employed as an RF design engineer intern where he had the opportunity to design complex PCB layouts for software defined radios.

Nathan Pen's qualifications for this project include prior coursework, employment, and personal projects. Some relevant electrical engineering coursework includes: ECE 2110 (Circuit Theory), ECE 2115 (Engineering Electronics), ECE 2140 (Logic Systems), ECE 2210 (Circuits, Signals, and Systems), ECE 3125 (Analog Electronics Design), ECE 3135 (Digital Design with FPGAs), ECE 3220 (Digital Signal Processing), and ECE 3410 (Communications Engineering). In his previous internship he created multiple wiring harnesses of 100+ wires, as well as programming microcontrollers for a final internship project. He also has done many hands-on projects outside of school that include working on cars, restoring cars, and a few carpentry projects like building a boat and random projects around the house back home. Many of these projects require working with a team and adapting to and quickly reacting to unexpected changes in the projects.

Appendix 5. Intellectual Contributions

1. We will purchase the microcontroller and related sensors or shields.
2. We will design, test, and implement circuit components to operate as desired.
3. We will optimize power consumption to allow for extended operation between charges.
4. We will purchase all the materials for physical design.
5. We will design the overall schematic.
6. We will write, test, and optimize any scripts used by the RPi.
7. We will design, write, test, and optimize any code and UI necessary for the app.

8. We will create the physical KeepSafe locking mechanism design in a CAD program and submit it for 3D printing.
9. We will construct the physical body of KeepSafe at the machine shop using provided tools.

Appendix 6. Teaming Arrangements

KeepSafe is very much a team-oriented project and all members, although specialized, are expected to help on all modules. The general projects that each member is managing is as follows:

- Jaret will be the lead in the circuitry of KeepSafe. This includes connecting the components to the microcontroller such as the servos, the load cells, camera, motion sensors, GPS, door switch, LED lights, and keypad. Jaret will also be responsible for overall box construction and cable management. He will also lead the mechanical side of the project such as the physical box and locking mechanism. Jaret will also assist with general tasks, team organization/planning, and the physical design of KeepSafe.
- Nathan will be the lead on programming for the microcontroller and its controls. Such as writing the logic flow in Python, creating the sensor test setups, writing the logic for most of the sensors, and combing them all together. He will also be working on the app's development as well and will oversee the app's communication with the microcontroller.
- Sean will be the lead on the app's development and the modeling of KeepSafe. This includes designing KeepSafe in Inventor and creating the assembly drawings. Once the design is finalized, work will begin on the app's UI and functionality, along with its connectivity with the microcontroller. He will write the HTML code that serves as the front end of the app and the python code that controls the backend. As well as help with the integrating the components to the electrical system.

Appendix 7. User Manuel

KeepSafe is entirely controlled by the app, with different print statements laid out within the code to give updates to the server output. In order to initialize the app, a few tasks must be done. First step is to boot up the Raspberry Pi. If the login credentials are needed, the username is pi and the password is KeepSafe1. Once logged in, open a new terminal. The first line to be written is as follows: “cd /var/www/piapp”. This will open the directory and the terminal will show the following screenshot.

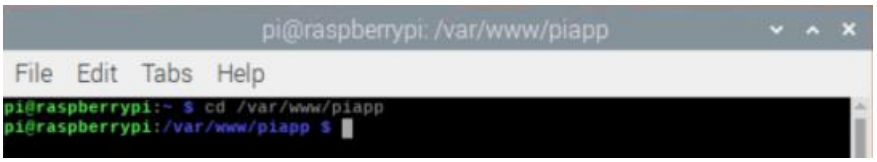


Figure 39: KeepSafe Setup 1

Once in this directory, write the following line: “export FLASK_APP=piapp2.py”. This will tell the pi which file should be run on the server, which is the piapp2 python file. Once the pi knows which file to run, the last line needed is: “flask run –host=0.0.0.0”. This will start the server and run it on the local IP of 0.0.0.0:5000. Note that in order to access the app on the phone or other device, the public IP will be needed which will be discussed later. The server is running when the following screen is shown

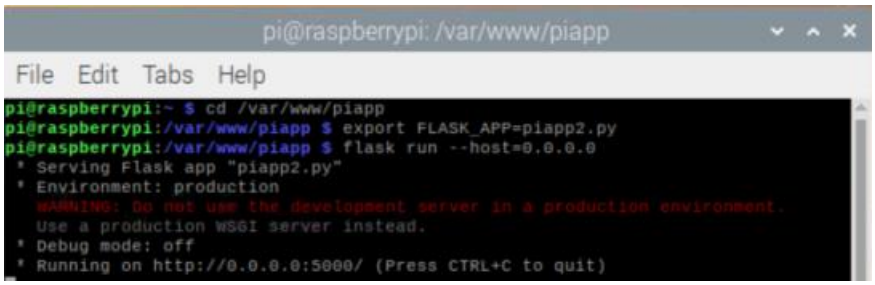


Figure 40: KeepSafe Setup 2

Now that the server is running, the user can connect using their phone. There is more initialization to do even after the server is running. First, locate the public IP address. The easiest way to determine this is by opening VNC viewer on the pi. The IP address listed under “Connectivity” is the address that will be needed to access the app on the phone. It is important to note that the phone and the pi must be on the same network in order to connect. For this example, the IP address will be 10.199.8.179. To access the app’s initialization page with this IP address, the user will type the following into a browser on their phone: 10.199.8.179:5000/initialization. The general form would be *User’s IP*:5000/initialization. When this page is displayed, KeepSafe will unlock itself and begin sensor scanning. KeepSafe is now fully initialized!

Once KeepSafe is up and running, in the search bar, remove “initialization” from the search. Following the earlier example, this would be 10.199.8.179:5000/, with the general form being *User’s IP*:5000/. This is KeepSafe’s main page and central operating point. This page

will refresh itself every five seconds to continually scan sensor inputs. The homepage will look like the following image.

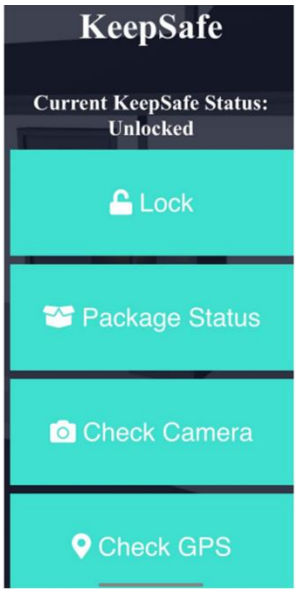


Figure 41: KeepSafe User Interface

Shown in this image are four options, a manual lock button, a package status button, a check camera button, and a GPS locater. While the lock and unlock buttons keep the user on the main page, the other three will bring the user to a new page. These new pages each will have a back button that will bring the user back to the main screen.

With the app fully running, it is important to understand the many capabilities included in the product. First, KeepSafe includes a PIR sensor, which continually scans for motion. If motion is detected, the sensor will send a message to the server, which can be seen through the print statements. KeepSafe is also equipped with a camera, which can display a live feed using the check camera button. A GPS is included in this product, and the location of KeepSafe can be found using the Check GPS button. KeepSafe also utilizes a gyroscope system, which can detect if an intruder is trying to break into the product. If the gyroscope is activated, the auto-locking system will initiate. KeepSafe's main functionality comes from load sensors, which are installed at the base of the locker. These sensors are not only able to detect when a package is placed inside KeepSafe, but can also measure its weight, which can be seen using the Package Status button. The load cells are also tied to the locking mechanism, so when KeepSafe detects a package and the door closure, the auto-locking system will initiate. To detect this door closure, a door trigger is attached at the top. To lock the door, a servo is used to drive a rack and bolt lock system, pushing the bolt into the side panel so the door cannot be opened. While many of the sensors can affect the lock's position, the user can manually control the servo using the lock and unlock buttons on the app. KeepSafe can also be unlocked using the keypad found on the right side of the device, with the code being 2143.

In order to turn off the device, the user must return to the terminal and the server. To turn KeepSafe off in the terminal, the user must press ctrl+C and then Enter. This will turn off

KeepSafe and the app. It is important to note that if using the same terminal page, only the "flask run –host=0.0.0.0" command is necessary to restart the device. If the terminal is closed, the entire initialization process is required. Each time is app is connected to on the phone, the initialization page must be opened as well.

Appendix 8. Board Fabrication Details (BFD)

Not applicable for our project.

Appendix 9. Mechanical Drawing

CAD

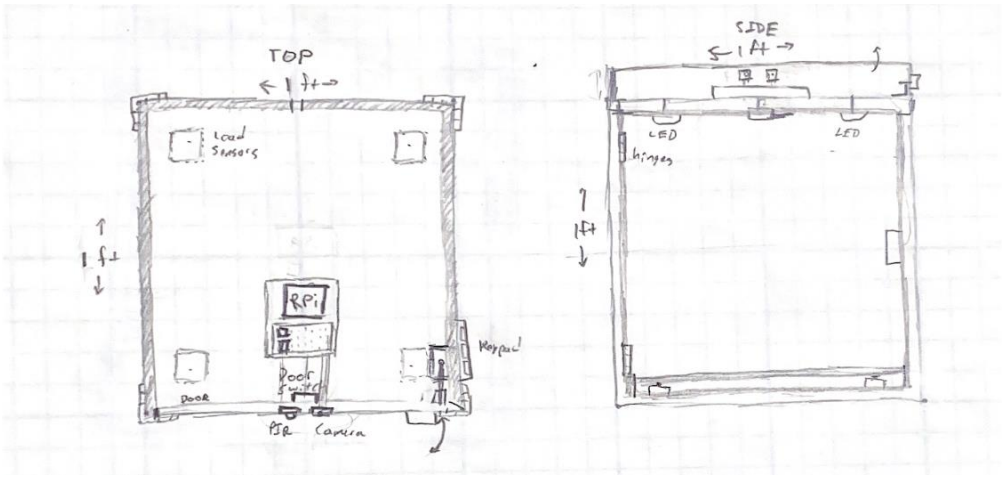


Figure 42: Mechanical Drawing

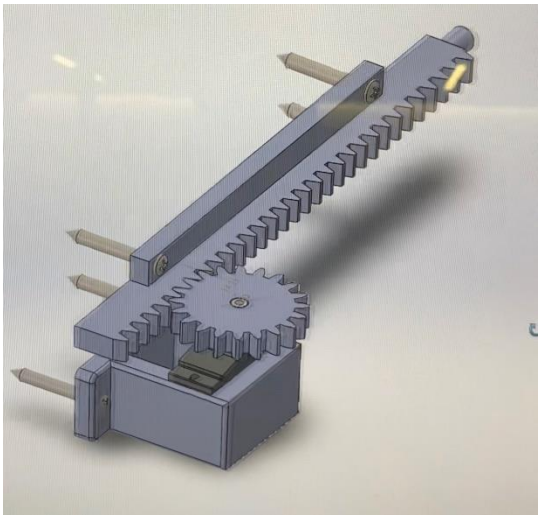


Figure 43: Locking Mechanism CAD

Appendix 10.

Figure 44: Final KeepSafe Picture 1

Figure 45: Final KeepSafe Picture 2

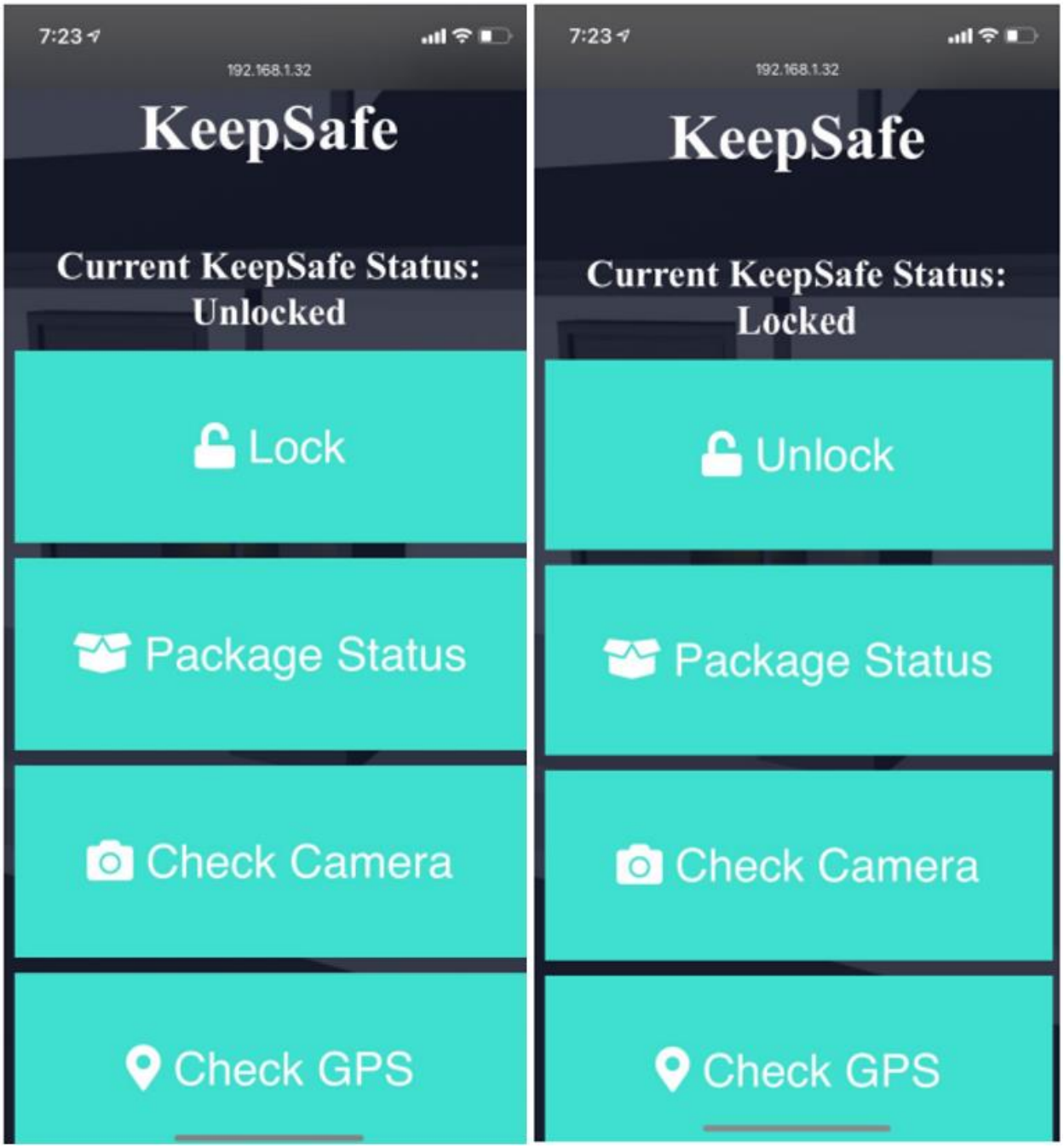


Figure 46: App Home Screen Unlocked & Locked

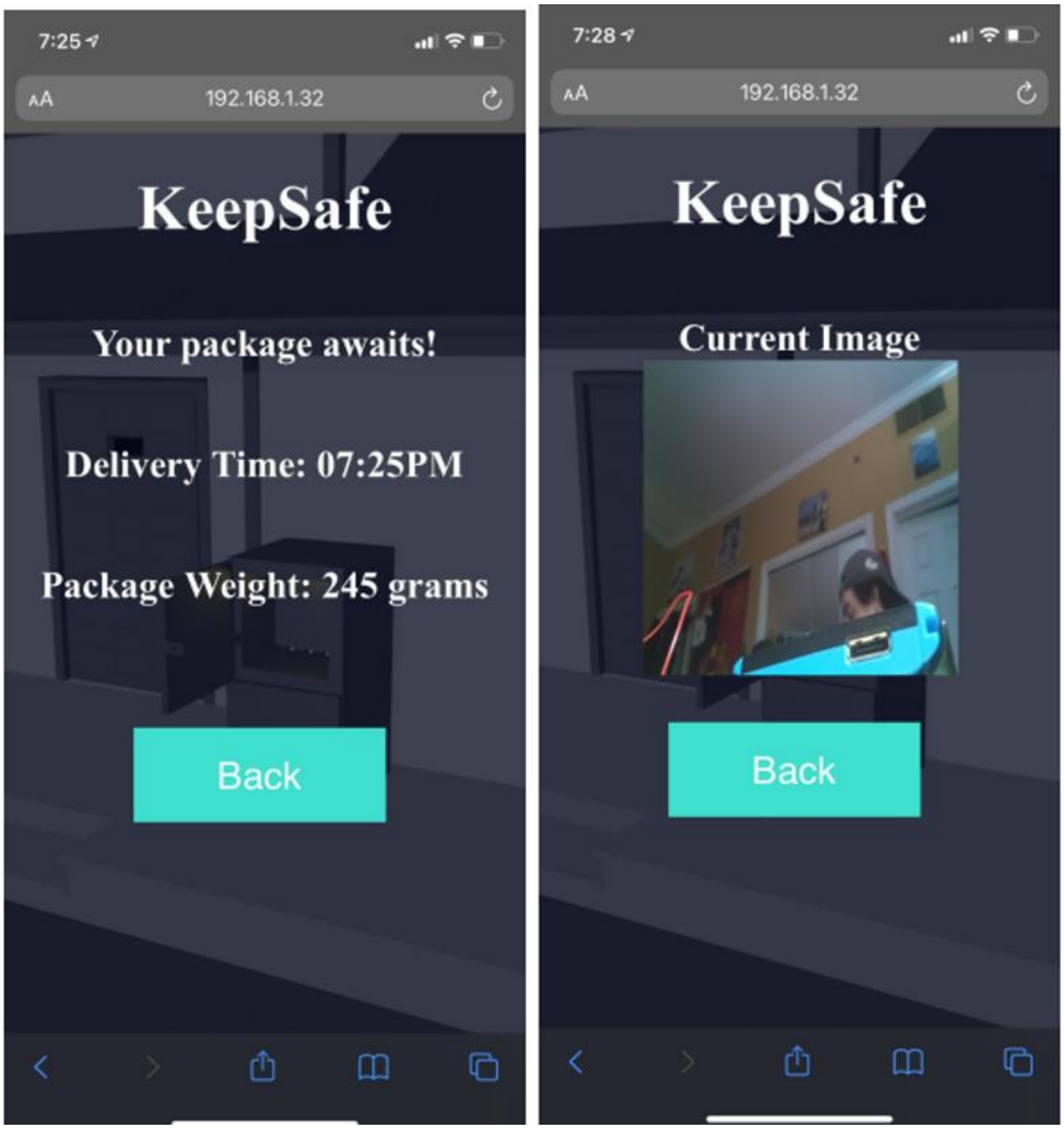


Figure 47: App Package Status and Check Camera Pages

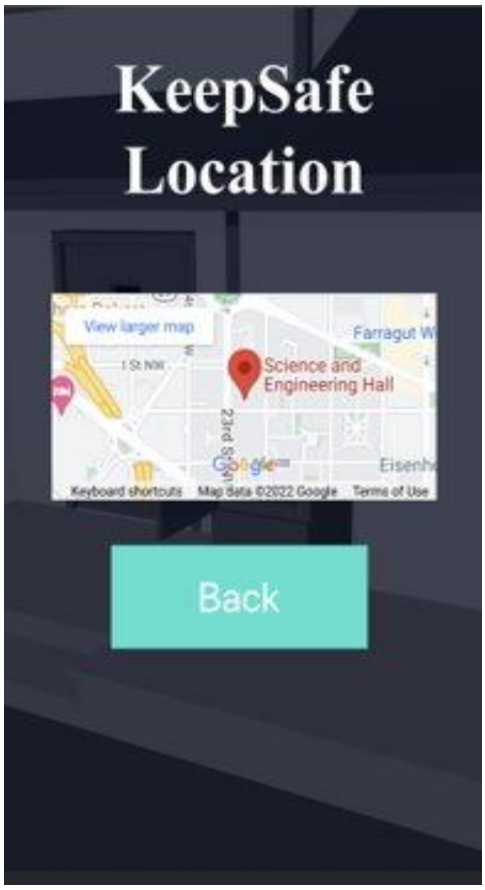


Figure 48: App Check GPS Page

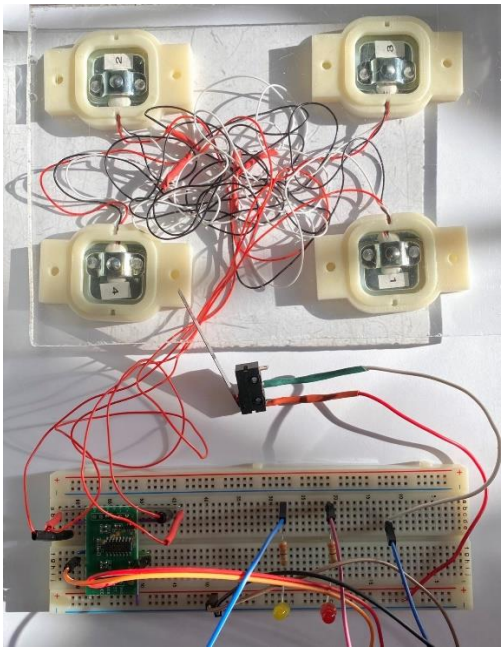


Figure 49: Package Detection Sensor Development

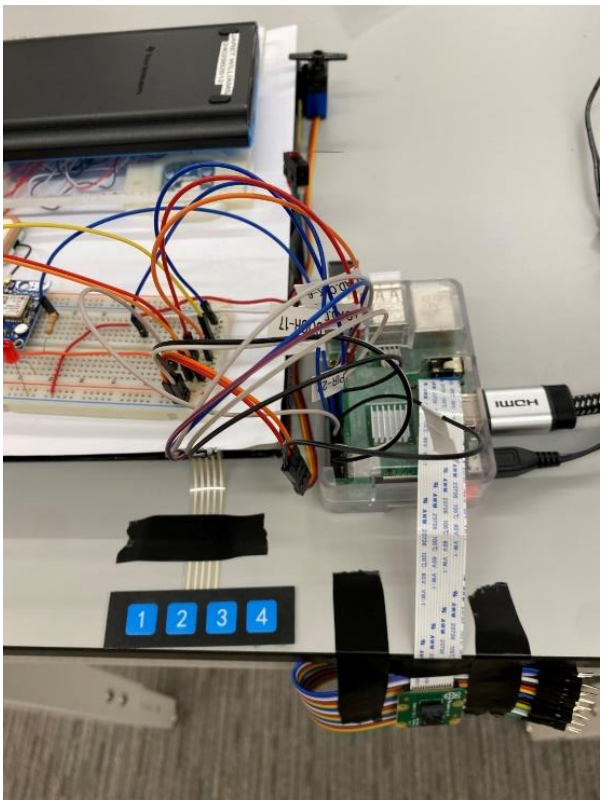


Figure 50: Camera, Keypad, Servo, and Switch Integration

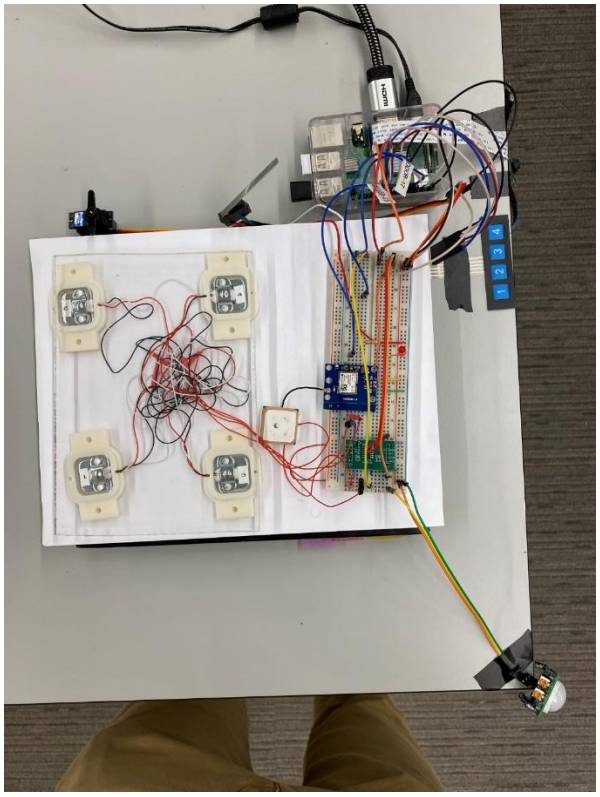


Figure 51: Full Sensor Integration Testing

Figure 52: KeepSafe Rendering 1

Figure 53: KeepSafe Rendering 2

Appendix 11.

```
import readline
from flask import Flask, render_template, Response, stream_with_context, request
import RPi.GPIO as GPIO
from picamera import PiCamera
from hx711 import HX711
from time import strftime
from gpiozero import Button
import time
import sys
import numpy
import threading
import cv2
import serial
from gpiozero import MotionSensor
import smbus2
from jinja2 import Environment, FileSystemLoader
from markupsafe import Markup
from flask_googlemaps import GoogleMaps, Map, icons
from dynaconf import FlaskDynaconf

# Imports ------------------------------------------------------------------------------------------

app = Flask(__name__) #Initializing Flask
app.config['GOOGLEMAPS_KEY'] = "AIzaSyBOl7vXaftpPpJzowQLEWB28VLtPxYqIaE"
GoogleMaps(app, key="AIzaSyBOl7vXaftpPpJzowQLEWB28VLtPxYqIaE") #Declaring API Key provided by Google
FlaskDynaconf(app) #GoogleMaps Initialization

servopin = 13
hx=HX711(5,6) #Declaring Load Cell pins


button = Button(17) #GPIO pin of Door Switch
pir = MotionSensor(22) #GPIO pin of PIR sensor

GPIO_LOCK=27
GPIO_LIGHT1=23
GPIO_LIGHT2=24 #Declaring Lock and Light Pins

GPIO.setup(GPIO_LOCK, GPIO.OUT)
GPIO.setup(GPIO_LIGHT1, GPIO.OUT)
GPIO.setup(GPIO_LIGHT2, GPIO.OUT) #Setting lock and light pins as outputs
```

Figure 54: Flask Code Part 1

```
42
43  pwm = GPIO.PWM(GPIO_LOCK,50) #Declaring servo pin
44  pwm.start(0)
45
46
47  video = cv2.VideoCapture(0) #Initializing Camera
48
49
50  referenceUnit= 22.03478058 #Reference unit for load cell
51  hx.set_reading_format("MSB", "MSB") #Initializing load cell
52  hx.set_reference_unit(referenceUnit)
53  hx.reset()
54  hx.tare()
55
56
57  numCount = 6 #Integer so that package weight can be averaged
58  TotalWeight = 0 #Initializng the weight variable
59
60  doorlocked = False #Booleans for door and lock status
61  doorclosed = True
62
63  setinitial= 0 #Boolean for initializing entire KeepSafe operation. Turns to 1 when complete
64
65  kp1 = False #Keypad booleans for each button and total status
66  kp2 = False
67  kp3 = False
68  kp4 = False
69
70  kpfinal = False
71
72  kt = 0
73
74
75  R1 = 11 #Pins for each keypad button
76  R2 = 8
77  R3 = 7
78  R4 = 1
79  C1 = 0
```

Figure 55: Flask Code Part 2

```
80
81  GPIO.setup(R1, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #Declaring each keypad button as a button and that they will produce an output
82  GPIO.setup(R2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
83  GPIO.setup(R3, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
84  GPIO.setup(R4, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
85  GPIO.setup(C1, GPIO.OUT)
86
87  GxLimit = 0.5 #Limits for gyroscope threshold
88  GyLimit = 0.5
89  GzLimit = 0.5
90
91  Gx = 0 #X Y and Z axis for gyroscope
92  Gy = 0
93  Gz = 0
94
95  #Values for Gyroscope
96  PWR_MGT_1 = 0x6B
97  CONFIG=0x1A
98  SAMPLE_RATE = 0x19
99  GYRO_CONFIG = 0x1B
100 ACCEL_CONFIG = 0x1C
101 ACCEL_X_HIGH = 0x3B
102 ACCEL_Y_HIGH = 0x3D
103 ACCEL_Z_HIGH = 0x3F
104 GYRO_X_HIGH = 0x43
105 GYRO_Y_HIGH = 0x45
106 GYRO_Z_HIGH = 0x47
107 TEMP_OUT_HIGH = 0x41
108
109 #initializes Gyro functionality
110 bus = smbus2.SMBus(1)
111 Device_Address = 0x68
112
113 ser = serial.Serial("/dev/serial0",baudrate = 9600,timeout=0.5) #Initializes serial port for GPS
114
115 GoogleLat = 0 #Lattitude and Longitude for GPS and google maps
116 GoogleLong = 0
117
118 #Setup ----------------------------------------------------------------
119
```

Figure 56: Flask Code Part 3

```
120 def CheckDoorStatus(): #Function to determine door status
121     if(button.is_active == True):
122         print ("doorclosed")
123         GPIO.output(GPIO_LIGHT1,False)
124         GPIO.output(GPIO_LIGHT2,False)
125         doorclosed = True
126
127     if(button.is_active == False): #If the door is open, start scanning for package
128         print ("dooropen")
129         doorclosed = False
130         GPIO.output(GPIO_LIGHT1,True)
131         GPIO.output(GPIO_LIGHT2,True)
132         WaitForPackage()
133
134     return doorclosed
135
136 def WaitForPackage(): #This function turns on the lights, then takes 5 readings and averages them to determine package weight
137     GPIO.output(GPIO_LIGHT1,True)
138     GPIO.output(GPIO_LIGHT2,True)
139     TotalWeight = 0
140     while(button.is_active==False):
141         time.sleep(5)
142         for x in range(numCount):
143
144             # Prints the weight.
145             val = hx.get_weight(5)
146
147             TotalWeight += val
148
149             hx.power_down()
150             hx.power_up()
151             time.sleep(0.1)
152
153     AvgWeight = TotalWeight/numCount
154
155
156
```

Figure 57: Flask Code Part 4

```
157     if(button.is_active==True): #When to door recloses Lock
158         DoorLock()
159 
160     return None
161 
162 def MeasureWeight(): #This function can be called at anytime, and is not tied to the door functionality
163     TotalWeight = 0
164     for x in range(numCount):
165 
166         # Prints the weight.
167         val = hx.get_weight(5)
168 
169         TotalWeight += val
170 
171         hx.power_down()
172         hx.power_up()
173         time.sleep(0.1)
174 
175     AvgWeight = TotalWeight/numCount
176     return AvgWeight
177 
178 def DoorLock(): #Turns of lights, and turns servo to lock
179 
180     GPIO.output(GPIO_LIGHT1,False)
181     GPIO.output(GPIO_LIGHT2,False)
182     pwm.ChangeDutyCycle(6.5)
183     print("DoorLocked")
184     global doorlocked
185     doorlocked= True
186     GetLockTime()
187 
188     return doorlocked
189 
```

Figure 58: Flask Code Part 5

```
190 def DoorUnlock(): #Unlocks the door and resets keypad controls
191     global setinitial
192     global kt
193     kt=0
194     if(setinitial==0):
195         pwm.ChangeDutyCycle(3)
196         pwm.stop()
197         fixInitial()
198 
199 
200     pwm.ChangeDutyCycle(3)
201     print("DoorUnlocked")
202     global doorlocked
203     doorlocked=False
204     return doorlocked
205 
206 def fixInitial(): #Sets the initialize boolean to 1
207     global setinitial
208     setinitial = 1
209     return setinitial
210 
211 def InitialUnlock(): #KeepSafe begins operation unlocked
212     if(setinitial==0):
213         pwm.ChangeDutyCycle(3)
214         print("DoorUnlockedInitial")
215         fixInitial()
216 
217     return None
218 
219 def GetLockTime(): #This function returns the time when the door was last locked
220     global locktime
221     locktime=strftime("%I:%M%p")
222     return locktime
223 
```

Figure 59: Flask Code Part 6

```
224 def readLine(line, chars): #Keypad function. Code is 2143
225     print ("readline")
226     global kt
227 
228     global kp1
229     global kp2
230     global kp3
231     global kp4
232 
233     global kpfinal
234     GPIO.output(line, GPIO.HIGH)
235     if(GPIO.input(R1)==1):
236         kp3 = True
237         time.sleep(0.2)
238     if(GPIO.input(R2)==1):
239         kp2 = True
240         time.sleep(0.2)
241     if(GPIO.input(R3)==1):
242         kp1=True
243         time.sleep(0.2)
244     if(GPIO.input(R4)==1):
245         kp4=True
246         time.sleep(0.2)
247 
```

Figure 60: Flask Code Part 7

```
249     if(kp2 == True and kp1 == False and kp3 == False and kp4 ==False and kt==0):
250     |   kt = 1
251     
252     if(kp2 == True and kp1 == True and kp3 == False and kp4 ==False and kt==1):
253     |   kt = 2
254     
255     if(kp2 == True and kp1 == True and kp3 == False and kp4 ==True and kt==2):
256     |   kt = 3
257     
258     if(kp2 == True and kp1 == True and kp3 == True and kp4 ==True and kt==4):
259     |   kt = 4
260     
261     if(kp2==True and kt>0 and (kp3==True or kp4 ==True)):
262     |    kp1==False
263     |    kp2==False
264     |    kp3==False
265     |    kp4==False
266     |    kt=0
267     if(kp2==True and kp1==True and kt>0 and kp3==True):
268     |    kp1==False
269     |    kp2==False
270     |    kp3==False
271     |    kp4==False
272     |    kt=0
273     if ((kp1==True or kp3 ==True or kp4==True) and kp2==False):
274     |    kp1==False
275     |    kp2==False
276     |    kp3==False
277     |    kp4==False
278     |    kt=0
279     
280     if(kt==4): #If the correct code is input, the door will unlock
281     |    DoorUnlock()
282     |    print("Code Accepted")
283     
284     GPIO.output(line, GPIO.LOW)
285     
286     return kt
```

Figure 61: Flask Code Part 8

```
288  def video_stream(): #Function to get live camera feed
289      while True:
290          ret, frame = video.read()
291          if not ret:
292              break
293          else:
294              ret, buffer = cv2.imencode('.jpeg', frame)
295              frame = buffer.tobytes()
296              yield (b' --frame\r\n' b'Content-type: image/jpeg\r\n\r\n' + frame +b'\r\n')
297
298
299
300  def MPU_initialization(): #Initializes gyroscope settings
301      print("MPU_Initialization")
302      bus.write_byte_data(Device_Address, PWR_MGT_1, 1)
303      bus.write_byte_data(Device_Address, SAMPLE_RATE, 7)
304      bus.write_byte_data(Device_Address, CONFIG, 0)
305      bus.write_byte_data(Device_Address, GYRO_CONFIG, 24)
306
307  def Read_data(reg_add): #Activates serial bus for gyroscope
308      print("ReadData")
309      high = bus.read_byte_data(Device_Address, reg_add)
310      low = bus.read_byte_data(Device_Address,reg_add+1)
311      value = (high<<8)|low
312
313      if value>35768:
314          value = value-65536
315
316      return value
317
318
```

Figure 62: Flask Code Part 9

```
319  def GyroCheck(): #Begins reading gyroscope data. If the reading is higher than assigned threshold, the door will lock
320      reg_add=0
321      print("GyroCheck")
322      MPU_initialization()
323      Read_data(reg_add)
324      global Gx
325      global Gy
326      global Gz
327
328      GYRO_X = Read_data(GYRO_X_HIGH)
329      GYRO_Y = Read_data(GYRO_Y_HIGH)
330      GYRO_Z = Read_data(GYRO_Z_HIGH)
331
332      Gx = GYRO_X/131.0
333      Gy = GYRO_Y/131.0
334      Gz = GYRO_Z/131.0
335
336      print(Gx)
337      print(Gy)
338      print(Gz)
339
340      if((abs(Gx)>=GxLimit or abs(Gy)>=GyLimit or abs(Gz)>=GzLimit)):
341          DoorLock()
```

Figure 63: Flask Code Part 10

```python
def NMEAConverter(data,direction): #NMEA converter for GPS. Outputs coordinates in latitude and longitutde
    LatTemp = str(data)
    NMEACoord = LatTemp[2:12]
    DirTemp = str(direction)
    CoordDirection = DirTemp[2:3]
    NMEAMin = NMEACoord[-8:]
    if(len(NMEACoord) == 10):
        NMEADeg = NMEACoord[0:2]
    else:
        NMEADeg = NMEACoord[0:3]
    DecDeg = int(NMEADeg)
    DecMin = float(NMEAMin)
    DecMin = DecMin/60
    DecCoord = round(DecDeg + DecMin,5)
    Coordinates = str(DecCoord) + " " + CoordDirection
    if(CoordDirection == "S" or CoordDirection == "W"):
        GoogleCoord = DecCoord * -1
    else:
        GoogleCoord = DecCoord
    return Coordinates, GoogleCoord
```

Figure 64: Flask Code Part 11

```python
def GPS(): #Takes data from NMEA converter and outputs values to be used for google maps
    while True:
        received_data = ser.readline()
        parts = received_data.split(b',')

        if b'$GPGLL' in parts[0]:

            if b'V' in parts[6]:
                print("There is no stable GPS connection")

            else:
                Latitude,GoogleLat = NMEAConverter(parts[1],parts[2])
                #print(Latitude)

                Longitude,GoogleLong = NMEAConverter(parts[3],parts[4])
                #print(Longitude)

                Coordinates = Latitude + " " + Longitude
                GoogleCoordinates = str(GoogleLat) + ", " + str(GoogleLong)

                print("The Current Location of your KeepSafe is: " + Coordinates)
                print("The Google Maps Coordinates are: " + GoogleCoordinates)

#Funtions ----------------------------------------------------------------------------
```

Figure 65: Flask Code Part 12

```
@app.route('/initialize') #Initialize function
def off():

    InitialUnlock()
    local_doorstatus = CheckDoorStatus()

    print ("initialzed")
    if(doorlocked==False):
        return render_template('off4.html')

    if(doorlocked==True):
        readLine()
        return render_template('on4.html')

@app.route('/') #Main page. Constantly reading keypad, gyroscope, pir, and door status. Webpage returned depends on data
def mainpage():

    local_doorstatus = CheckDoorStatus()
    readLine(C1, ["3", "4", "1", "2"])
    GyroCheck()
    global kt

    print("kt = ", kt)

    if(pir.motion_detected == True):
        print("Motion detected")


    if(doorlocked==False or kt>9):
        return render_template('off4.html')

    if(doorlocked==True):
        print("Sitting on lock")
        return render_template('on4.html')
```

Figure 66: Flask Code Part 13

```
@app.route('/Unlock') #Page returned if unlock button is pressed
def unlockpage():
    DoorUnlock()
    GyroCheck()
    local_doorstatus = CheckDoorStatus()


    if(doorlocked==False):
        return render_template('off4.html')

    if(doorlocked==True):
        return render_template('on4.html')

@app.route('/Lock') #Page returned if lock button is pressed
def lockpage():
    global kt
    DoorLock()
    readLine(C1, ["3", "4", "1", "2"])
    print("kt = ", kt)
    GyroCheck()
    local_doorstatus = CheckDoorStatus()


    if(doorlocked==False):
        return render_template('off4.html')

    if(doorlocked==True):
        return render_template('on4.html')
```

Figure 67: Flask Code Part 14

```python
@app.route('/PackStat') #Page returned to check the package's status
def PackStat():
    global locktime

    weightVal=MeasureWeight()
    weightVal_round=round(weightVal)

    return render_template('PackageStatus.html', weightVal_round=weightVal_round, time=locktime)

@app.route('/pic') #Page returned to check the live camera
def pic():
    return render_template('CheckCamera.html')

@app.route('/video_feed') #Function to create the live image from the camera
def video_feed():
    return Response(video_stream(), mimetype = 'multipart/x-mixed-replace; boundary=frame')

@app.route('/gps') #PAge returned for GPS. Takes the google coordinate from the globaal variables and gives them to the html
def mapview():
    print("GPS")
    global GoogleLat
    global GoogleLong
    gmap = Map(
        identifier="gmap",
        varname="gmap",
        MapLat = GoogleLat,
        MapLong = GoogleLong,

    )

    return render_template("GPS.html", gmap=gmap)
```

Figure 68: Flask Code Part 15

```html
<!DOCTYPE html>
<html>
<head>
        <meta name ="viewport" content="width=device-width, initial-scale=1.0"> <!-- Setting up html, making sure it adjusts to screen size -->
        <meta http-equiv="refresh" content="5"> <!-- Refreshes page every 5 seconds to check sensors -->
        <title> KeepSafe Home </title> <!-- Title of page -->
        <link href = "{{url_for('static', filename = 'style.css')}}" rel="stylesheet" type="text/css"> <!-- Adding CSS style, fonts, and icons -->
        <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
        <script src="https://kit.fontawesome.com/9d4ad59c28.js" crossorigin="anonymous"></script>

        <style>

        .header{
            min-height: 100vh;
            width: 100%;
            background-position: center;
            background-size: cover;
            position: relative;
            background-image: linear-gradient(rgba(4,9,30,0.7), rgba(4,9,30,0.7)), url({{url_for('static', filename='Render2.png')}});
            }
        </style> <!-- Added style to put KeepSafe Image as background -->
</head>
<body>
    <section class="header">
        <h1> KeepSafe </h1>
        <h2> Current KeepSafe Status: Unlocked</h2> <!-- Titles  -->

    <form action="/Lock">
    <button class="button  btn--block"><i class="fas fa-unlock"></i> Lock </button></form> <!-- Each button is linked to a flask function, and an icon is added to each button

    <form action="/PackStat">
    <button class="button  btn--block"><i class="fas fa-box-open"></i> Package Status </button></form>

    <form action="/pic">
    <button class="button  btn--block"><i class="fas fa-camera"></i> Check Camera </button></form>

    <form action="/gps">
    <button class="button  btn--block"><i class="fas fa-compass"></i> Check GPS </button></form>

</section>
</body>

</html>
```

Figure 69: HTML Code, Main Page

```
<!DOCTYPE html>
<html>
<head>
        <meta name ="viewport" content="width=device-width, initial-scale=1.0"> <!-- Setting up html, making sure it adjusts to screen size -->
        <meta http-equiv="refresh" content="5"> <!-- Refreshes page every 5 seconds to check sensors -->
        <title> KeepSafe Home </title> <!-- Title of page -->
        <link href = "{{url_for('static', filename = 'style.css')}}" rel="stylesheet" type="text/css"> <!-- Adding CSS style, fonts, and icons -->
        <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
        <script src="https://kit.fontawesome.com/9d4ad59c28.js" crossorigin="anonymous"></script>


        <style>
        .header{
            min-height: 100vh;
            width: 100%;
            background-position: center;
            background-size: cover;
            position: relative;
            background-image: linear-gradient(rgba(4,9,30,0.7), rgba(4,9,30,0.7)), url({{url_for('static', filename='Render2.png')}});
            }

        </style> <!-- Added style to put KeepSafe Image as background -->

</head>
<body>
    <section class="header">
        <h1> KeepSafe </h1>
        <h2> Current KeepSafe Status: Locked</h2><!-- Titles  -->

    <form action="/Unlock">
    <button class="button  btn--block"><i class="fas fa-unlock"></i> Unlock </button></form> <!-- Each button is linked to a flask function, and an icon is added to each button

    <form action="/PackStat">
    <button class="button  btn--block"><i class="fas fa-box-open"></i> Package Status </button></form>

    <form action="/pic">
    <button class="button  btn--block"><i class="fas fa-camera"></i> Check Camera </button>

    <form action="/gps">
    <button class="button  btn--block"><i class="fas fa-compass"></i> Check GPS </button></form>

</section>
</body>

</html>
```

Figure 70: HTML Code, Main Page 2

```
<!DOCTYPE html>
<html>
<head>
        <meta name ="viewport" content="with=device-width, initial-scale=1.0"> <!-- Setting up html, making sure it adjusts to screen size -->
        <title> Check Camera </title>
        <link href = "{{url_for('static', filename = 'style.css')}}" rel="stylesheet" type="text/css"><!-- Adding CSS style-->

        <style>

          .header{
            min-height: 100vh;
            width: 100%;
            background-position: center;
            background-size: cover;
            position: relative;
            background-image: linear-gradient(rgba(4,9,30,0.7), rgba(4,9,30,0.7)), url({{url_for('static', filename='Render2.png')}});
            }

        </style><!-- Added style to put KeepSafe Image as background -->
</head>
<body>
    <section class="header">
        <h1> KeepSafe </h1>
        <h2>Current Image</h2>

        <img id = "bg" src = "{{ url_for('video_feed')}}" <!-- Pulls video feed function from Flask -->


        <form action="/">
            <button class="button2 btn--back">Back</button></form><!-- Back button to return to homepage -->

</section>


</body>

</html>
```

Figure 71: HTML Code, Live Camera

```html
<!DOCTYPE html>
<html>
<head>
    <meta name ="viewport" content="with=device-width, initial-scale=1.0"><!-- Setting up html, making sure it adjusts to screen size -->
    <title> Package Status </title>
    <link href = "{{url_for('static', filename = 'style.css')}}" rel="stylesheet" type="text/css"><!-- Adding CSS style-->

    <style>
    .header{
        min-height: 100vh;
        width: 100%;
        background-position: center;
        background-size: cover;
        position: relative;
        background-image: linear-gradient(rgba(4,9,30,0.7), rgba(4,9,30,0.7)), url({{url_for('static', filename='Render2.png')}});
        }
    </style><!-- Added style to put KeepSafe Image as background -->
</head>
<body>
    <section class="header"> <!-- Displays time and weight for package info. Time and weightVal_round variables are pulled from Flask -->
        <h1> KeepSafe </h1>
        <h2> Your package awaits!</h2>
        <h3> Delivery Time: {{time}} </h3>
        <h4> Package Weight: {{weightVal_round}} grams</h4>

        <form action="/">
        <button class="button btn--back">Back</button></form><!-- Back button to return to homepage -->

</section>
</body>

</html>
```

Figure 72: HTML Code, Package Status

```html
<!DOCTYPE html>
<html>
<head>
    <meta name ="viewport" content="with=device-width, initial-scale=1.0"><!-- Setting up html, making sure it adjusts to screen size -->
    <title> GPS</title>
    {{gmap.js}} <!-- Including google maps javascript code -->
    <link href = "{{url_for('static', filename = 'style.css')}}" rel="stylesheet" type="text/css"><!-- Adding CSS style-->

    <style>
    .header{
        min-height: 100vh;
        width: 100%;
        background-position: center;
        background-size: cover;
        position: relative;
        background-image: linear-gradient(rgba(4,9,30,0.7), rgba(4,9,30,0.7)), url({{url_for('static', filename='Render2.png')}});
        }
    </style><!-- Added style to put KeepSafe Image as background -->
</head>
<body>
    <section class="header">
        <h1> KeepSafe Location </h1>
        <!-- Embedding google maps into page -->
        <h2><iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3105.0513665474305!2d-77.05044957287505!3d38.89994065933144!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1

        <form action="/"><!-- Back button to return to homepage -->
        <button class="button btn--back">Back</button></form>

</section>
</body>

</html>
```

Figure 73: HTML Code, GPS Location

```css
*{
    margin: 0;
    padding: 0;
} /* Setting webpage to display with no border in center */

.header{
    min-height: 100vh;
    width: 100%;
    background-image: linear-gradient(rgba(4,9,30,0.7), rgba(4,9,30,0.7)),url(images/Render2.png);
    background-position: center;
    background-size: cover;
    position: relative;
}/* Centering background image*/

.CurrentImage{

    height: 450px;
    width: 400px;

    color: white;
    padding-top: 500px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 36px;
    margin: 4px 2px;


    top: 100px;
    left: 300px;
    background-image: url(images/SeanLetavish_Picture.jpg);
    background-size: cover;
    position: relative;

} /* Setting size for the live video feed */
```

Figure 74: CSS Code Part 1

```
.button {
    display: inline-flex;
    height: 250px;
    width: 200px;
    border: none;
    color: white;
    padding: 0;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 36px;
    margin: 4px 2px;
    cursor: pointer;
    background: #40e0d0;
    position: absolute;
    top: 300px;
    left: 560px;
} /* Creating buttons. Buttons are teal, somewhat large, and centered on all pages. Buttons 2, 3, and 4 follow this same design */

.button:hover {
    background: #34b7aa
} /* Button slightly darkens when hovered over by mouse cursor */

.button:active {
    background:#227a71
} /* When pressed, button darkens even more */

.button2 {···
}

.button2:hover {···
}

.button2:active {···
}

.button3 {···
}

.button3:hover {···
}

.button3:active {···
}

.button4 {···
}
```

Figure 75: CSS Code Part 2

```
138 >  .button4:hover {···
140    }
141
142 >  .button4:active {···
144    }
145
146
147
148  h1{
149      color: white;
150      text-align: center;
151      padding-top: 30px;
152      font-size: 50px;
153  } /* Setting headings size, color, and location. As headings increase, font size decreases, with paragraph being the smallest font */
154
155  h2{
156      color: white;
157      text-align: center;
158      padding-top: 60px;
159      font-size: 30px;
160  }
161
162  h3{
163      color: white;
164      text-align: center;
165      padding-top: 60px;
166      font-size: 30px;
167  }
168
169  h4{
170      color: white;
171      text-align: center;
172      padding-top: 60px;
173      font-size: 30px;
174  }
175  h5{
176      color: white;
177      padding-left: 400px;
178      padding-top: 200px;
179      font-size: 30px;
180  }
181
182  p{
183      color: white;
184      padding-left: 1000px;
185      font-size: 30px;
186  }
```

Figure 76: CSS Code Part 3