Department of Software Engineering

Lakehead University

ESOF 4969 - Degree Project

2021-22

Final Report - Software

Requirements

Group #2

Contributing Members:

Angel Martinez, Christopher Silver and Shane Davey

Date Submitted: 04/25/2022

Approval

This document has been read and approved by the following team members responsible for its implementation:

| **PRINT NAME: Angel Martinez** |
| --- |
| Signature: |
| **PRINT NAME: Christopher Silver** |
| Signature: |
| **PRINT NAME: Shane Davey** |
| Signature: |
| **COMMENTS**: |
| All of us collaborated on all aspects of editing the design document but each of us |
| had our focus areas. |
| Angel focused on Chapter 1.1, Chapter 1.2, Chapter 2.1, Chapter 2.5.1, Chapter 4.1.1, |
| Chapter 4.2, Chapter 4.2.1, Chapter 4.3, Chapter 6.1.2, and Chapter 6.2.2 |
| Chris focused on Chapter 1.1, Chapter 2.4, Chapter 2.5.2, Chapter 3, Chapter |
| 4.1.2, Chapter 6.1.3, 6.1.4, 6.2.3. |
| Shane focused on Chapter 1.2, Chapter 2.2, Chapter 2, Chapter 5.1.1, Chapter 5.1.2, |
| Chapter 5.2, Chapter 6.1.1, Chapter 6.2.1. |
|  |

# Table of Contents

[**Table of Contents**](#_o43ni8v5zvyt) **2**

[**Chapter 1**](#_w4des1a9jujo) **4**

[1.1. Introduction / Abstract](#_iohy87i3907q) 4

[1.2. Background](#_rrrscinsg9rx) 4

[1.3. Literature review](#_la6piztk76wl) 6

[1.3.1. Video-based fall detection by spatio-temporal join-point model](#_tupdb9afdnua) 6

[1.3.2. Non-invasive low cost fever detection system](#_lk0dzj1gwxos) 7

[1.4. System Requirements and Specification](#_464luwc4og8v) 7

[1.5. Requirement Engineering Tools](#_hol3jqbghyjb) 10

[**Chapter 2**](#_tqqiciarf06y) **11**

[2.1. System Design](#_cqrl99m4wuzc) 11

[2.2. Theoretical Model](#_p2vfpyqg46ed) 12

[2.3. Formal Design Methodology](#_5jaoasro6t2m) 12

[2.4. Design structure](#_jm8ndprw4egi) 14

[2.5. Analysis of Algorithm Complexities Time and Storage](#_8lvd7n2s976z) 18

[2.5.1. Fall Detection Algorithm Complexity](#_2w2il8mzc5d4) 18

[2.5.2. Fever Detection Algorithm Complexity](#_ytdfammrukuo) 18

[**Chapter 3**](#_iltg3g1sim8g) **21**

[3.1. System Implementation / Simulation](#_8viqdzsd297v) 21

[3.1.1. Algorithm Design for the Detection of Fevers:](#_3g0fvrecfeua) 21

[3.1.1.1. Gathering Information on ArraySoft V2 Output](#_eahxv3sw70w8) 21

[3.1.1.2. Parsing the Text File](#_jgdnykw3x39o) 22

[3.1.1.3. Statistical Analysis vs. Image Analysis](#_cf780x9hlhkj) 23

[3.1.1.4. Thermopile Sensor Hyperparameters](#_wks3gld3a8ba) 27

[3.1.2. Fully Implemented Features vs. Simulated Features](#_ppe3sj3arnay) 28

[3.1.3. Full Physical Prototype](#_xtzkqngg0gw) 32

[**Chapter 4**](#_eocasfibco6p) **33**

[4.1. Testing and Evaluation](#_8sp1wb8qkq5b) 33

[4.1.1. Fall Detection Test Cases](#_vp8q7dtgjn1d) 33

[4.1.2. Fever Detection Test Cases](#_9bqpoo4qfwwy) 35

[4.2. Software Quality Assurance Standard](#_lkaernxdpu63) 39

[4.2.1. Risks](#_bvw0bfnybe9y) 41

[4.3. Existing Models Comparison](#_r9ozwo1yhtcn) 41

[**Chapter 5**](#_9sarpdq4vpxc) **43**

[5.1. Design Impact](#_jnk7lkhwl7j) 43

[5.1.1. Privacy and Security Impacts](#_ekinzrzvbbn) 43

[5.1.2. Societal and Economic Impacts](#_f4jivkgdhe1n) 43

[5.2. Project Cost](#_g9v071lx8yf7) 44

[**Chapter 6**](#_luuz5rnybkrh) **45**

[6.1. Problems Encountered](#_xlsjovy8oaa2) 45

[6.1.1. Web Application](#_37zab1ai0nbg) 45

[6.1.2. Fall Detection](#_9uxdb5cnq1p2) 46

[6.1.3. Fever Detection](#_qq1j6eybv3dz) 46

[6.1.4. Hardware Issues](#_zeyc5spcd8qk) 47

[6.2. Future Work](#_k3iksnnspxci) 52

[6.2.1. Web Application](#_5jkg5wgqdr6b) 52

[6.2.2 Fall Detection](#_vzswkafgqvg9) 53

[6.2.3 Fever Detection](#_46gelp6hhfb4) 53

[**References**](#_2xh1qi9nv155) **55**

# 

# 

# 

# **Chapter 1**

## **1.1. Introduction / Abstract**

Falling is already a danger to any age group, but for age groups like senior citizens, it could be life-threatening. For this issue, we propose a motion detection and tracking solution. In this project we implement a thermopile that will be enhanced to detect infrared energy to capture a senior’s temperature at any time and send health threats such as a fever or a detected fall to a healthcare worker associated with that patient. The objective of this project is to enhance the quality of life for senior citizens. With this technology implemented, they will feel more safe in their own home and be reassured that someone will help them in case their health deteriorates. If the citizens are living in long-term care homes, these homes can either hire less PSWs or take on more patients as our system is a safeguard of sorts. In personal homes, patients can maintain their independence in living alone whilst also alleviating worry from their families as they know the system will notify the associated health care professional if something adverse like a fall or fever occurs. Upon completing our prototype, we have achieved a fall detection accuracy of 78%. This occurs in real time. We have also achieved a fever detection accuracy of 88%. This does not happen in real time, rather in batches. Moving forward, we would like to get an enhanced thermopile sensor. Our current sensor has a resolution of only 80x64 and is unable to detect humans. Because of this, we needed to implement an RGB camera for the fall detection, which uses OpenCV for facial detection. This caused us to have a low accuracy of only 78%. Any fall tests where the human was not facing the camera were not correctly detected. Also, the sensor has a temperature drop off at range, causing 12% of tests to incorrectly classify because of faults in temperature out of our control.

## **1.2. Background**

In Canada there is an aging population and each year the proportion of elderly people (65 and up) grows larger and larger as the Baby Boom generation ages. On top of this there is also a lack of personal support workers that can give care to the elderly people that are in need of it. Below in Figure 1 we can see the population distribution in Canada as of 2005, then below that in Figure 2 we can see the population distribution in Canada as of 2020. Between these two figures we can visually see that there is a large increase in elderly people especially between the ages of 55 - 75. Analyzing the statistics, there was an increase of the elderly population from 13.1% to 18% from 2005 to 2020, which is a staggering increase of 4.9% in only 15 years. People in this age range are at a far greater risk of serious injury when they fall. With the growing elderly population and the lack of PSWs, the problem of lack of care will only continue to worsen. To solve this issue we cannot just make more PSWs but we can make their job easier and enable them to care for more clients then they would be able to otherwise.

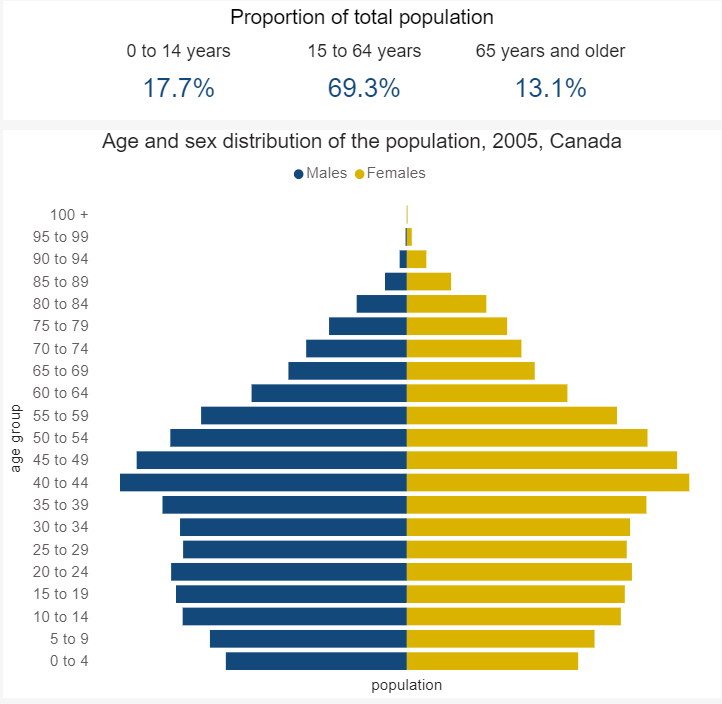


Figure. 1: Canadian Population Distribution in 2005 [15].

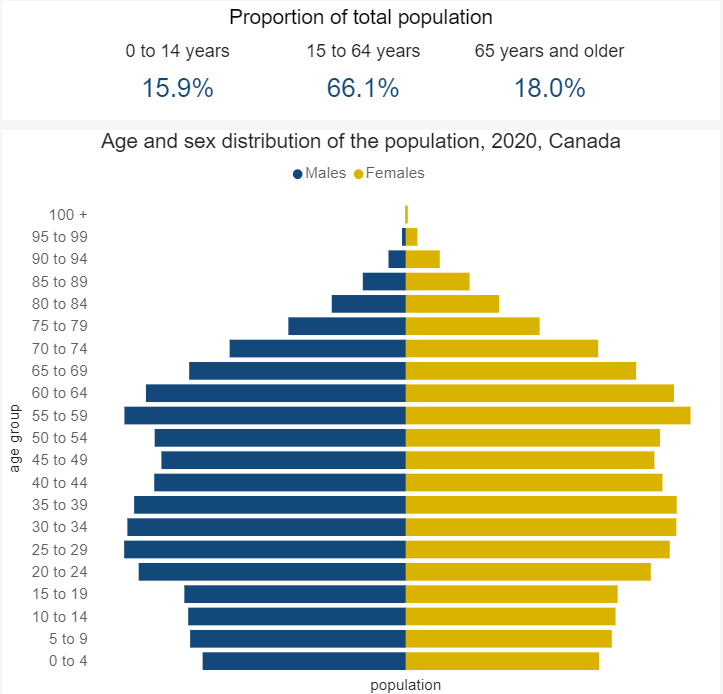


Figure. 2: Canadian Population Distribution in 2005 [15].

## **1.3. Literature review**

The literature review section focuses on papers that have been published for fall detection and fever detection. We do this in order to have a baseline of what type of work has already been completed and to what degree of success. This allows us to later do a comparison of the accuracy of related works and our proposed model.

### **1.3.1. Video-based fall detection by spatio-temporal join-point model**

This paper proposes a Long Short-Term Memory (LSTM) model to perform fall detection on a video fall [11]. The way this paper goes about implementing this solution is by first using a pose extractor which turns the frames of the video into joint-point features of the human falling down. After doing this, the paper uses a geometric joint-point filter to determine different characteristics of the person falling; one is the direction the human is falling whether it be rightward, leftward, upward, or downward. The other characteristics include the aspect ratio, lower and upper body ratio, and coordinate relationship. Once at this step the paper processes the video fall by constructing a Spatio-temporal join-point model which the LSTM uses. In the end, this paper obtained a 96.28% accuracy on the Multicam [12] dataset and a 99.00% on the URFD dataset [13].

### **1.3.2. Non-invasive low cost fever detection system**

This paper covers fever detection using a low cost system which includes a Lepton 3.5 thermal camera to capture thermal images [14]. Once the thermal image is captured it is analyzed using the YOLOv5 model to detect human eyes in the thermal images. Once the human eye is detected these pixels of the thermal image are then converted into temperature readings. The threshold used in this paper for a fever was any temperature higher than 100 degrees fahrenheit which is 37.78 degrees celsius. This paper obtained an accuracy of 99.7. This was a very cost effective solution but for our application this solution is ineffective since they elderly people will not be looking directly at the thermal sensor.

## **1.4. System Requirements and Specification**

The requirements were captured by looking at use cases, user stories, and sequence diagrams. When it came to use cases, we created use cases for each action the system takes such as when a fall or fever is detected. From this, a requirement was captured that states what the system must accomplish when this event occurs. For user stories, we wrote down what the user needs to be able to do, for example, “As a PSW I want to be able to know when a fever has been detected by email.” This user story was then turned into a requirement since it is crucial for the system. The last method we used to capture the requirements of the system was by looking at the sequence diagrams. Figure 3shows the sequence diagram of a user logging into the web application, this was then turned into a requirement. We repeated these steps until we found all the requirements of the system.

To analyze the requirements captured we looked at it from a business perspective and determined that some requirements were out of scope and instead we would focus on the important aspects of the system which were fall and fever detection and the web application. This helped narrow down our requirements and keep the scope of the project at a size that would be able to be completed in the time frame.

Next, we needed a method to validate the requirement with our final design, this means we must find a way to show that our final product meets the user's needs. To accomplish this we used a method called design validation which is seen in Figure 4. The process is as follows, first we take the system and prepare it for testing. The test cases used encompass the whole system since each system part is interconnected, the input for these test cases is based on the requirements of the system. We then get an output for each test case, the output will be validated against the requirements if it is not validated successfully we repeat the process. In the end, we end up with a fully validated design.

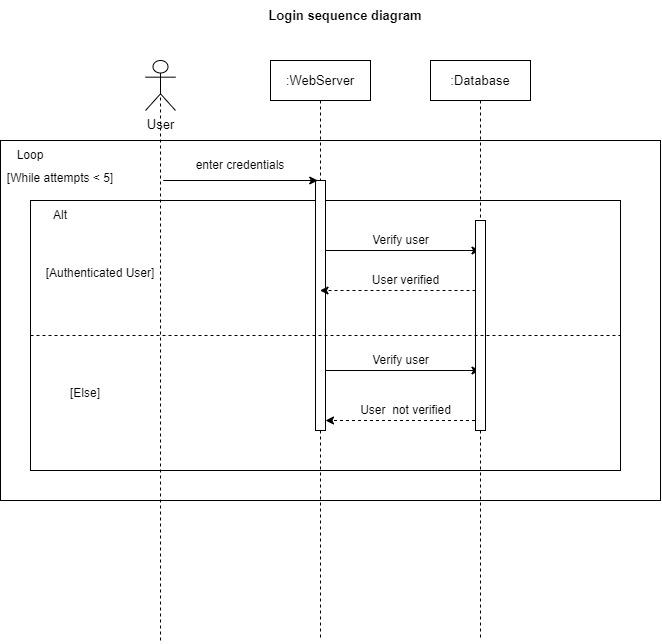


Figure. 3:User Interacting with Web Application.

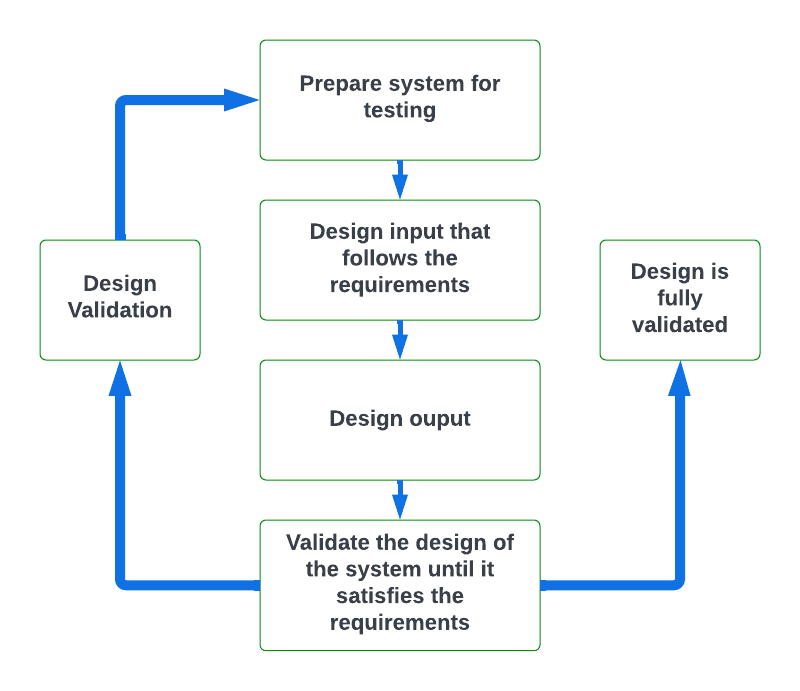
****

Figure. 4:Validating System Design.

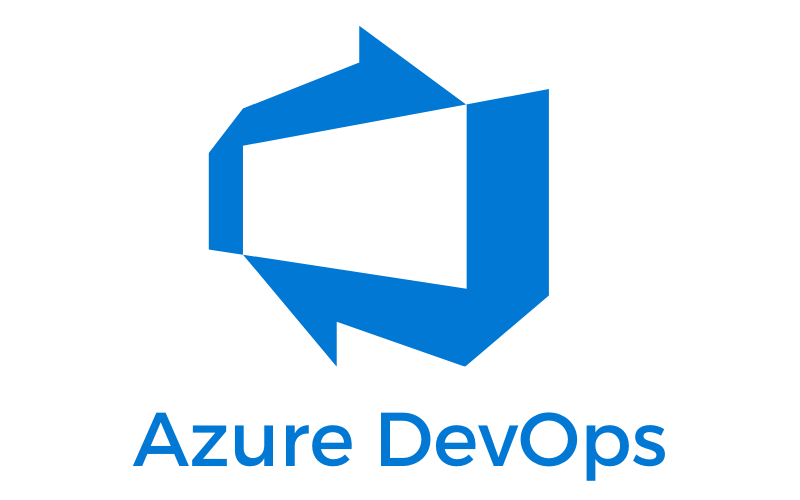
## 

## 

## **1.5. Requirement Engineering Tools**

As a group, we used two main tools to keep track of the requirements of the project, the first being GitHub and the second being Azure DevOps. At the beginning of the project we began using GitHub to share code and upload the most recent updates to the code, this allowed us to do requirement-based test cases on the current version of the project. This was still not enough to do status tracking and tracing of requirements but using features available on GitHub we were able to perform these tasks. The process of doing this was explained in the paper “Requirements management in GitHub with a lean approach” [6]. Where you build a hierarchy of tasks where each task is a requirement and then you use the issue tracker in GitHub to write the title of the requirement and a description of what is expected. This method of requirement management was very inefficient thus we started looking into alternatives and found a tool called Azure DevOps.

Azure DevOps worked well for us since it allows you to connect your GitHub account to this software thus working as a version control tool and offering the tools to keep track of requirements. The way we kept track of requirements using Azure was by creating a backlog that included all the features that needed to be implemented, from here each backlog could be assigned to a member of the group. After writing all the requirements of the system on the backlog page we then had a group meeting to decide what backlogs were to be done in what sprint. With this, any member of the group could open the software tool and check what requirements have not been done or what has been done and match it with our specification documents.



# **Chapter 2**

## **2.1. System Design**

In this section, we show the design of the system that relates to each requirement and specification found in the specification document of this project.

1. The size of both fever and fall detection algorithms is less than 32GB, in terms of computation power fall detection uses a max of 5% of the CPU and. Thus ensuring that the complexity of the algorithms is enough to run on a raspberry pi 4.

2. The sensor being used for fever detection is a thermophile sensor. For fall detection after completing tests it was concluded that fall detection with a thermophile sensor was not resulting in accurate results therefore an RGB camera will replace the sensor for fall detection.

3. The completed product is set up in a wooden box that can be mounted to the corner of a room.

4. The system can save information in the database after a fall or fever is detected, this information is then sent to the web application and is displayed for PSWs to see. The communication between the web application and database works both ways meaning any changes made on the web application will be reflected in the database.

5. The system can detect falls in real-time using OpenCV and a haar cascade file.

6. Implementation of the system on the raspberry pi has not been completed but once completed the raspberry pi contains an ethernet port that will enable the raspberry pi to access the internet as long as the electricity is on.

7. A python script was written to automate sending an email to the corresponding PSW indicating whether there was a fall or fever. The patient's information is also sent through email since the python script communicates with the database.

8. The web server is being run locally for testing purposes until the product is fully functional and integrated into the raspberry pi, from here the web server would be hosted on google cloud.

9. The database is fully connected to the locally hosted web server allowing us to perform queries which will check for user login credentials as well as add users to the database.

## **2.2. Theoretical Model**

No theoretical models were used or developed during the process of developing this project.

## **2.3. Formal Design Methodology**

The design methodology that we used in order to develop this project was the iterative and incremental development lifecycle model. We chose this one so that we would be able to mitigate risks early on as well as develop a working prototype early on in the development. This way we would be able to get a better understanding of the changes and improvements that we would have to make in order to create a quality product. The prototype would also give us a good idea of the final scale of the system and having it early on would help to give us a good idea of just how much time it would take to implement everything we would need, this in turn helped us develop a structured timeline. This methodology was used throughout the entire development process from the requirements to the final testing as we worked in workflows instead of phases and although we used multiple workflows at the same time one of them was always the main focus during a given period of time. An example of the workflows in parallel can be seen in Figure 5 below**.**

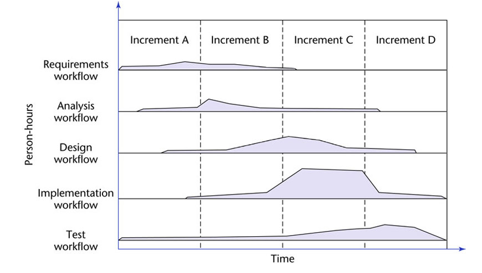


Figure. 5:Iterative and Incremental Workflows.

## 

## 

## 

## 

## **2.4. Design structure**

The design follows a modular approach and can be represented from the following block diagrams. We created the design in this manner so we could build each component separately, test separately, integrate, and keep the code organized.

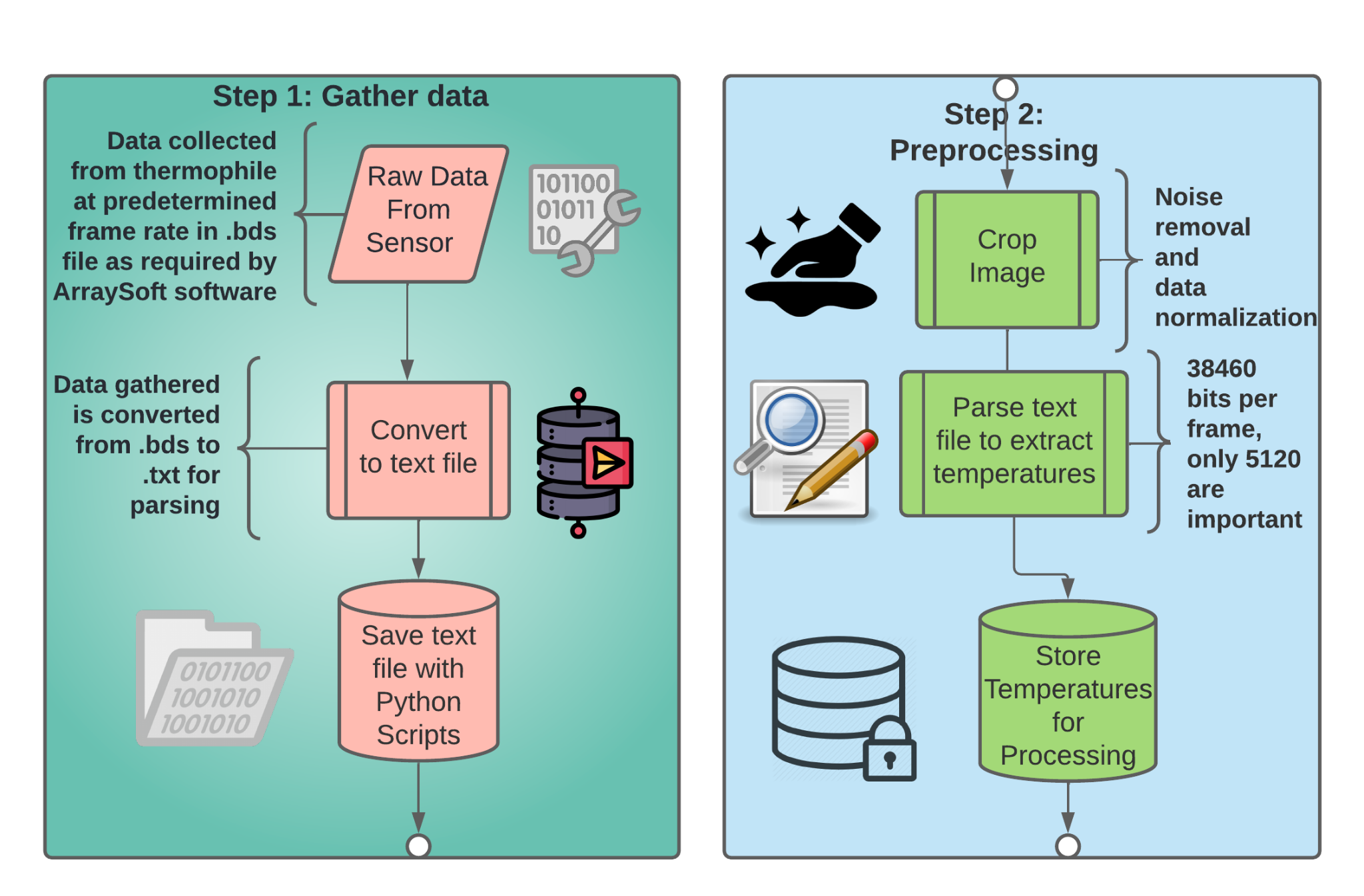


Figure. 6:First Two Steps of our High-Level Design.

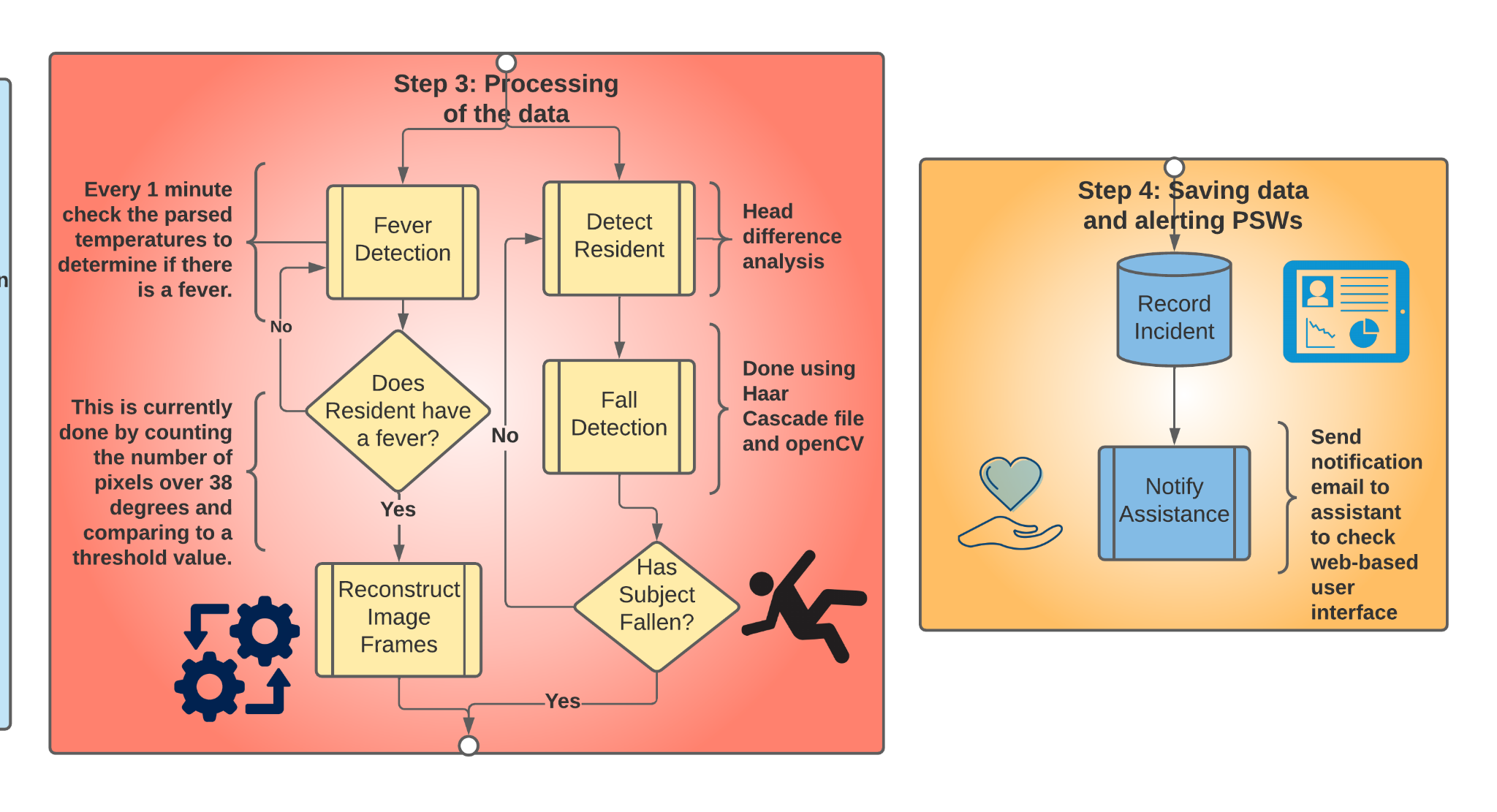


Figure. 7: Last Two Steps of our High-Level Design.

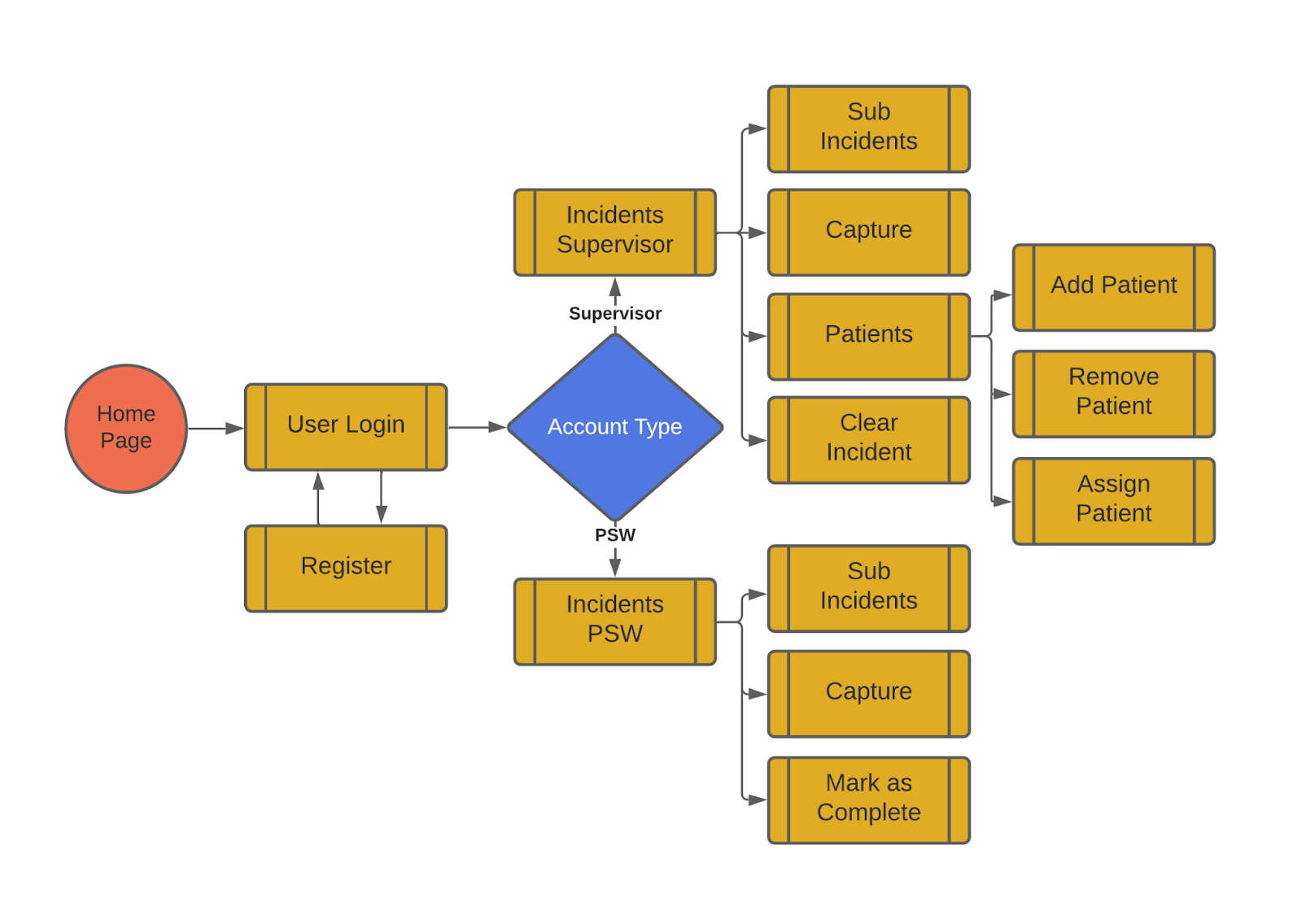
****

Figure. 8: Design of the Web Application.

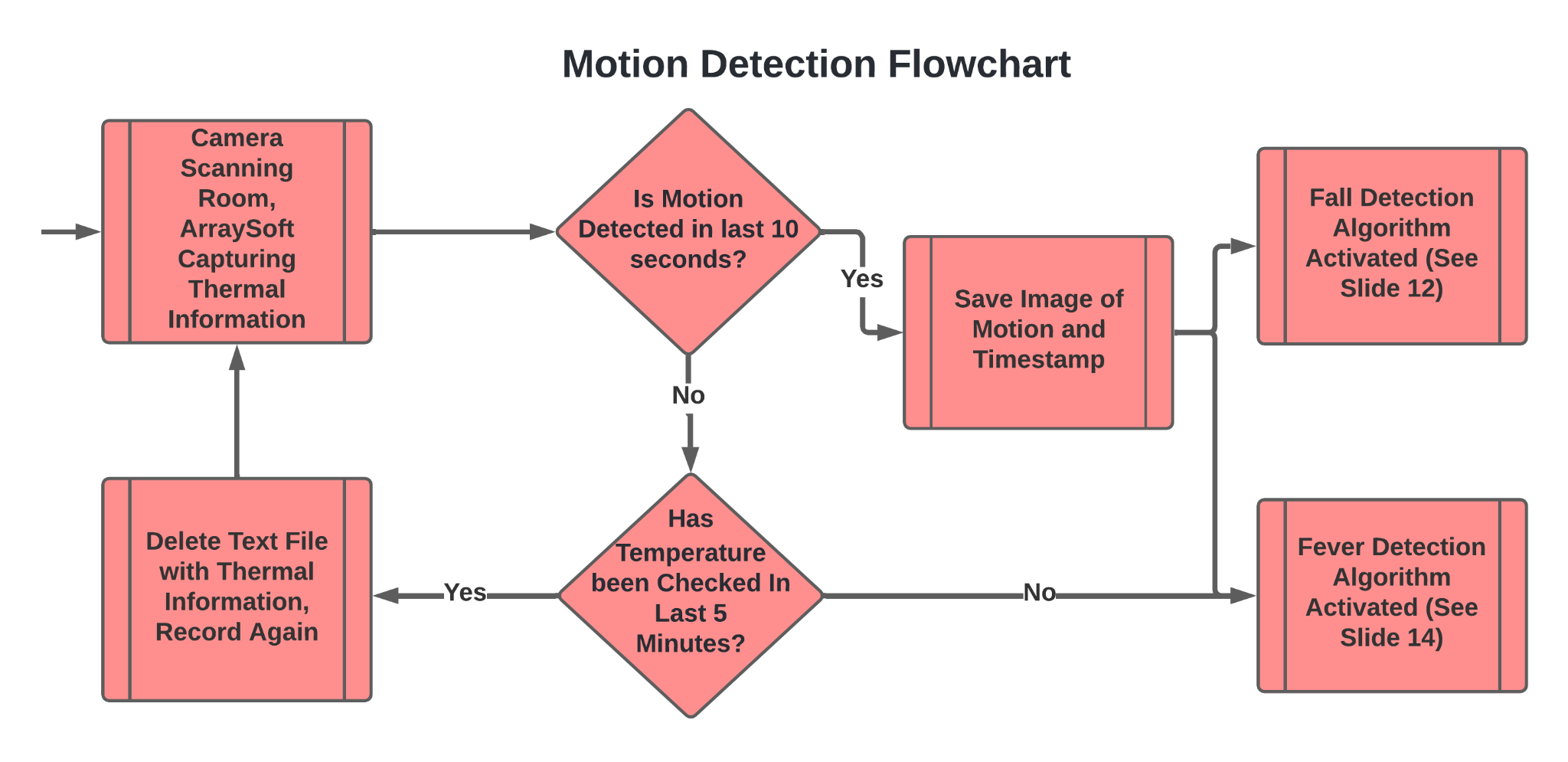
****

Figure. 9: Design of Motion Detection Algorithm.

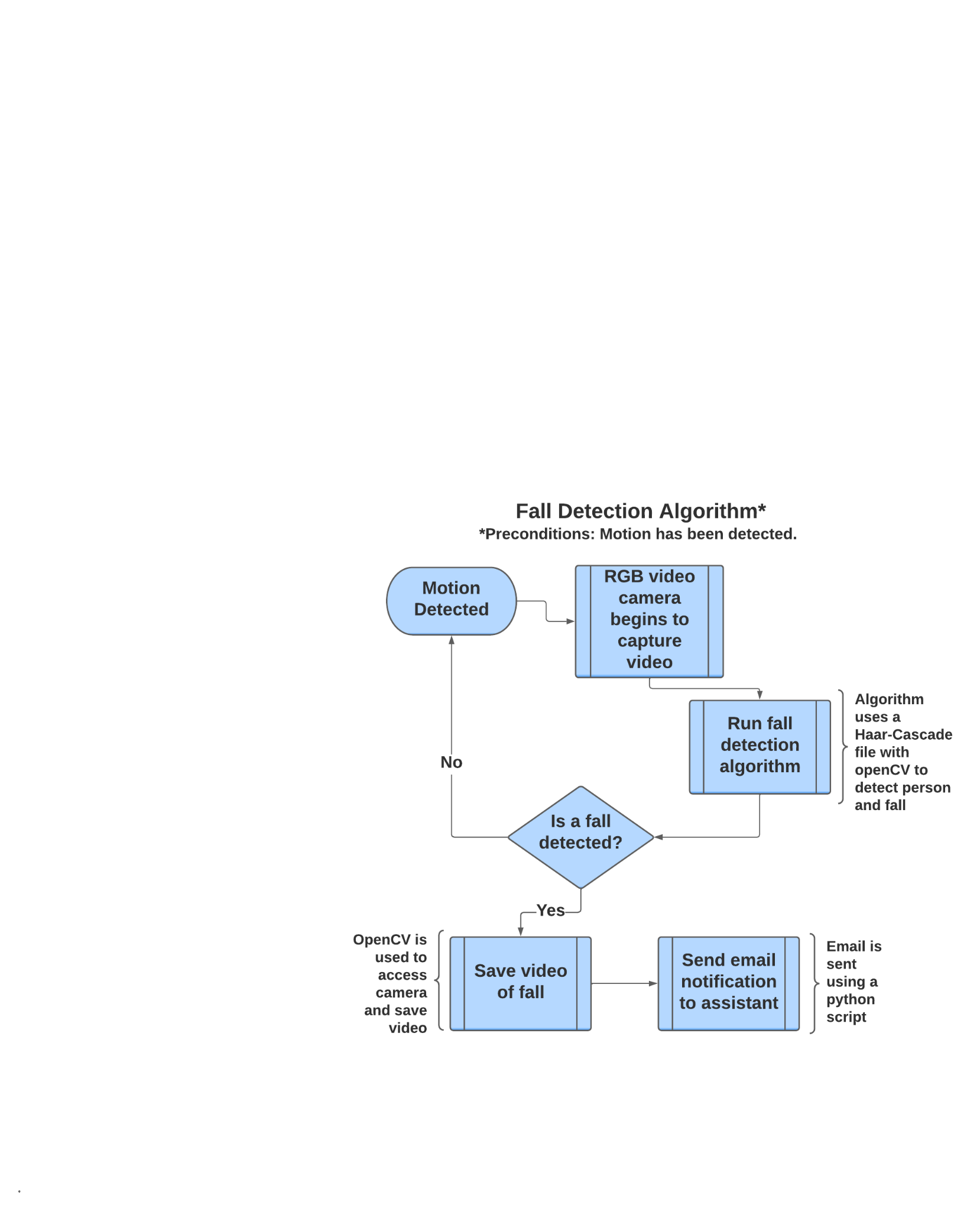
****

Figure. 10: Design of Fall Detection Algorithm.

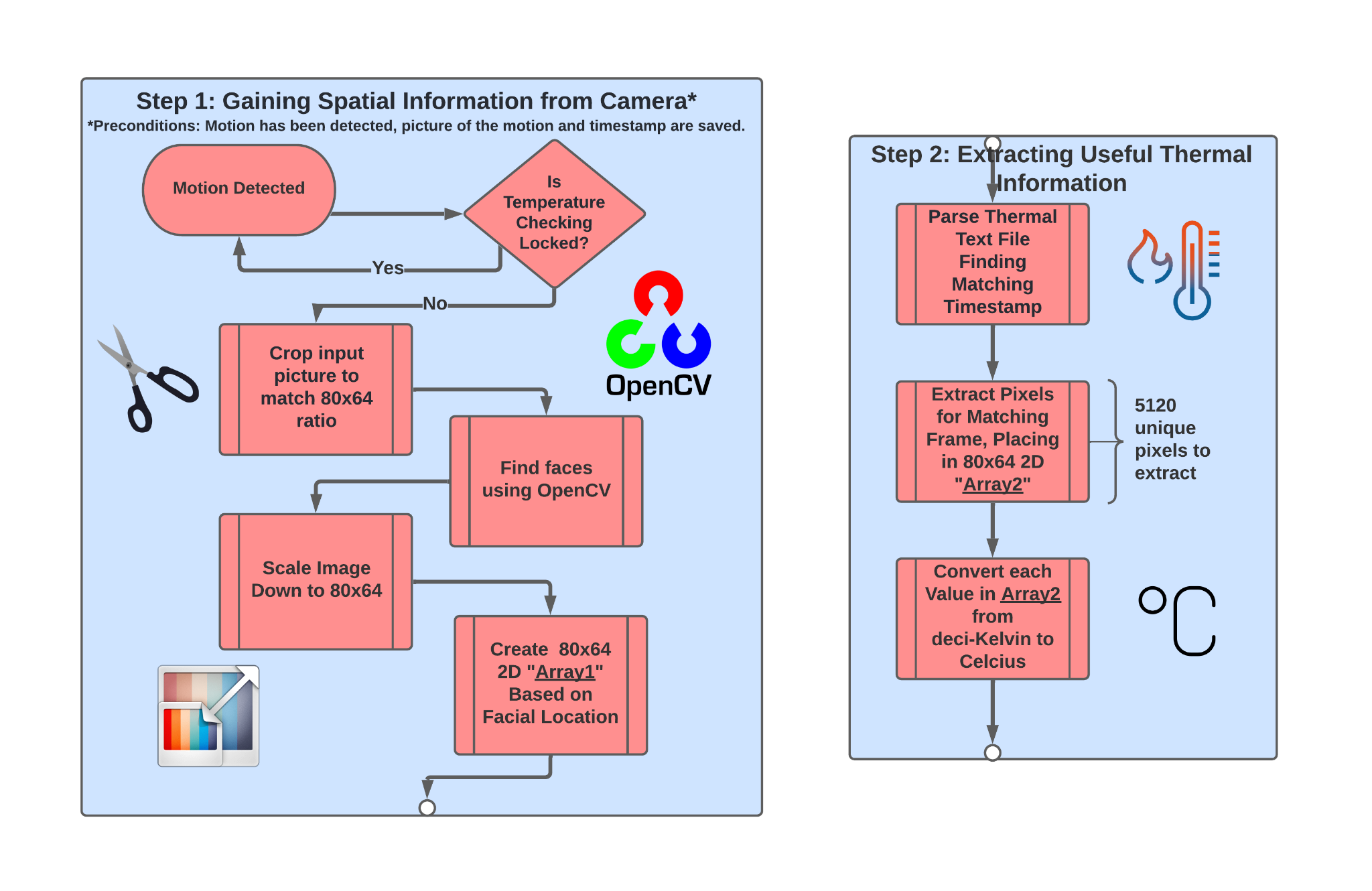
****

Figure. 11:First Two Steps of our Fever-Detection Design.

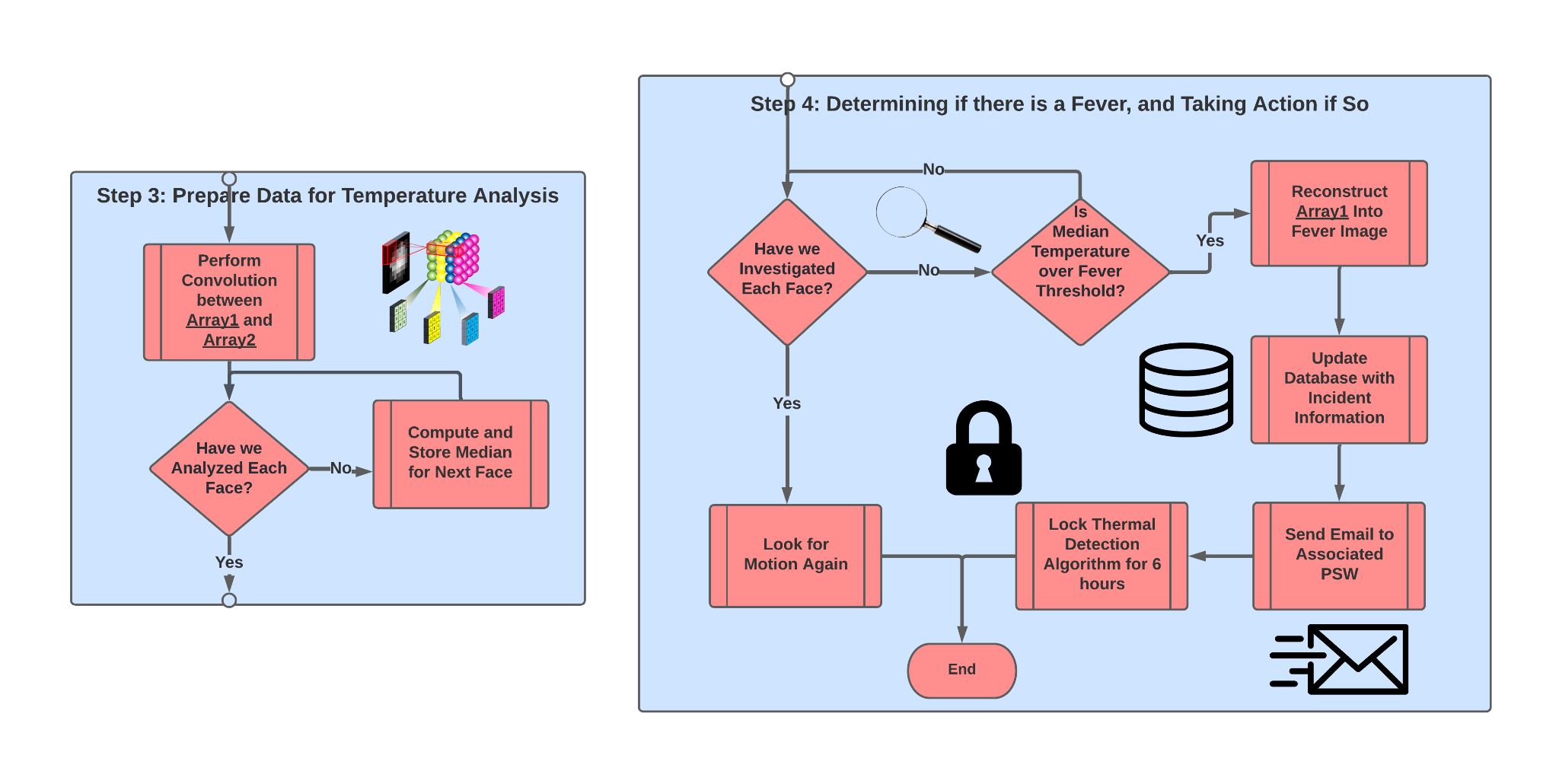
****

Figure. 12:Last Two Steps of our Fever-Detection Design.

As can be seen in the above Figure 6 to Figure 12we had a top down approach. We first created our initial high-level design and then split the remaining design into the web application, motion detection, fall detection, and fever detection. From here the system was modularized and we were able to implement each individual component before integrating them into and realizing our high-level design.

## **2.5.** [**Analysis of Algorithm Complexities Time and Storage**](https://docs.google.com/document/d/15k3QWW89QD-aol1_h5VATa4tAzlPlJ-bXxykaZB5GXE/edit#heading=h.idp5v7o0pfnn)

### **2.5.1. Fall Detection Algorithm Complexity**

The fall detection algorithm has little complexity in terms of time and storage thus working well for real-time-based fall detection. Table 1 shows the CPU usage and memory usage of a computer running the algorithm. We see that the CPU usage ranged from 1.7% to 5% this was due to other processes running in the background causing lag in the CPU making the algorithm demand more CPU power to keep things running smoothly. In terms of memory usage, it remained constant at 81.6 MB. These two results show that the algorithm consumes little resources making it optimal to use in a raspberry pi that will be running both fall and fever detection algorithms.

Table. 1: CPU and Memory Consumption of Fall Detection Algorithm.

|  | CPU Usage | Memory Usage |
| --- | --- | --- |
| Fall detection algorithm | 1.7% to 5% | 81.6 MB |

### **2.5.2. Fever Detection Algorithm Complexity**

As previously mentioned, fever detection happens in batches because of a hardware limitation out of our control. We are able to control the number of frames in the batches however, and were able to gather the results below in Table 2 which compare the number of frames to the time in seconds it takes to parse.

Table. 2: Number of Frames Vs Time in Seconds for Parsing.

| Number of Frames | 2 | 21 | 70 | 136 | 1000 |
| --- | --- | --- | --- | --- | --- |
| Total Time(s) | 6.7 | 8.1 | 11.1 | 17.0 | 81.2 |

As can be seen, the time for the system to complete all steps increases as the number of frames increases. This seemed to be a linear pattern and we extrapolated and interpolated to create Equation 1 where Y is the total time of execution (in seconds) and X is the number of input frames. It should be noted that these times were calculated as the fall detection algorithm and other processes were open, and that these were tested on a personal computer which has more resources than our raspberry pi. Also, this is execution time from the very start when the text file is opened to parse for temperatures, to the very end when the website is updated with a new incident.

Equation. 1: Linear Equation for Time to Execute (Y) with Number of Frames Input (X).

A linear equation is a good sign and shows that the system will not get out of control and consistently behind in parsing. If we employ 5 minutes between testing our batches of temperature information, and if we gather 1 frame every 5 seconds, we will have 11.12 seconds of processing every 5 minutes. This means there will be a maximum of 5 minutes as 12 seconds between a fever occurring and a PSW being notified. This is excellent as we are confident the system will not consistently try to catch up processing previous batches of temperatures. The only potential issue is that this was not calculated on a Raspberry Pi, but rather a personal computer. If the Raspberry Pi has 25 times less resources than the personal computer, the processing would still be completed in 278 seconds, therefore having 22 seconds to spare before the next batch starts to be processed. We know the raspberry pi doesn’t have 25 times less resources, so we are happy with these results.

In terms of space complexity, old text files and pictures are deleted from the local memory of the system after the batch is completely parsed. These files and pictures only comprise a few megabytes anyways. The resulting incidents are stored on a web database, so space is not needed on the local machine for these incidents. Space is not really a concern until a database would completely fill, but we are doing our best to avoid this by locking the fever detection algorithm for a few hours after a fever is detected so the database does not get flooded with incidents that are all the same fever being reported multiple times.

# **Chapter 3**

## **3.1. System Implementation / Simulation**

### 3.1.1. Algorithm Design for the Detection of Fevers:

#### 3.1.1.1. Gathering Information on ArraySoft V2 Output

We first connected the sensor to our testing computer using a LAN cable. We could also connect wirelessly using a personal router, but we ran into issues with the Lakehead WiFi on residence so we remained with the LAN cable for the project.

Once we were able to connect to the sensor we tried to figure out the format of the output files. When the thermopile array records, it saves the information as a binary data stream (.bds) file. This .bds file is unreadable, with seemingly random characters everywhere. Within the ArraySoft V2 software, there is an option available to convert this binary data stream into a text file (.txt) that has readable information. We converted the .bds stream to a .txt file. We opened the resulting text file and were bombarded with a massive amount of 5 digit integers.

We consulted with the manual and found the following information about the text file: each line in the text file corresponded to one frame captured by the sensor. Each frame starts with a temperature value for each pixel, and then a voltage value for each block of 4 pixels. Next there is 1 number for the VDD, 1 number for the calculated ambient temperature, 6 numbers for PTAT, and then 1 time value (notably 10 bits) at the end of each line.

We found the length of each frame in Python, which turned out to be 38,470 bits long. The table 3 below describes which bits in each frame correspond with what information. Note that the 80 and 64 represent the resolution of the sensor, and each number in the text file is 5 digits, and then 1 character of whitespace, so each 6 characters in the text file represents one integer value.

Table. 3: Description of text file generated from video stream in ArraySoft V2 software.

| **Type of Information** | **Information Needed?** | **Bit Values for this Type of Information** | **Equation to get type of information** | **Total Number of Bits in Current Frame** |
| --- | --- | --- | --- | --- |
| Temperature of each pixel | Yes | 0 - 30,719 |  | 30,720 |
| Electrical offset values of each block of 6 pixels | No | 30,720 - 38,399 |  | 38,400 |
| VDD | No | 38,400 - 38,405 |  | 38,406 |
| Ambient temperature | Yes | 38,406 - 38,411 |  | 38,412 |
| PTAT values | No | 38,412 - 38,459 |  | 38,460 |
| Current time | No | 38,460 - 38,469 |  | 38,470 |

Now we knew we needed to look between bit 0 and 30,719 for the temperature of each pixel, and look at bits 38,406 to 38,411 for the ambient temperature pre-calculated. Next we needed to parse the text file to gather this information.

#### 3.1.1.2. Parsing the Text File

We opened the file and looped from the first line (first frame) until the final line (final frame). Within each frame, we looped 6,410 times () because there are 6,410 different integers in each frame representing different information as can be seen in Table 3 above.

We had two variables, called “xStart” and “xEnd”, which represented the current position within the text file. We then said as long as the xStart variable was less than 30,720 then this is a temperature value corresponding to a pixel in that specific frame. We gather the current temperature of that pixel using Equation 2 because we needed to convert the temperatures from deci-Kelvin to degrees celsius:

Equation 2: Converting Temperature from deci-Kelvin to Celcius.

Once the temperature was in degrees celsius, we simply compared the temperature of this frame to a threshold temperature, and if that pixel’s temperature was higher than our threshold temperature, we recorded where on the image this pixel was found. To simply test this, after parsing all integers in that frame, we compared the number of hot pixels found in that frame to our threshold number. If there were too many hot pixels, we inserted a record into the “Incidents” table in our local database, with the necessary information such as patient information, date and time, current temperature, etc.

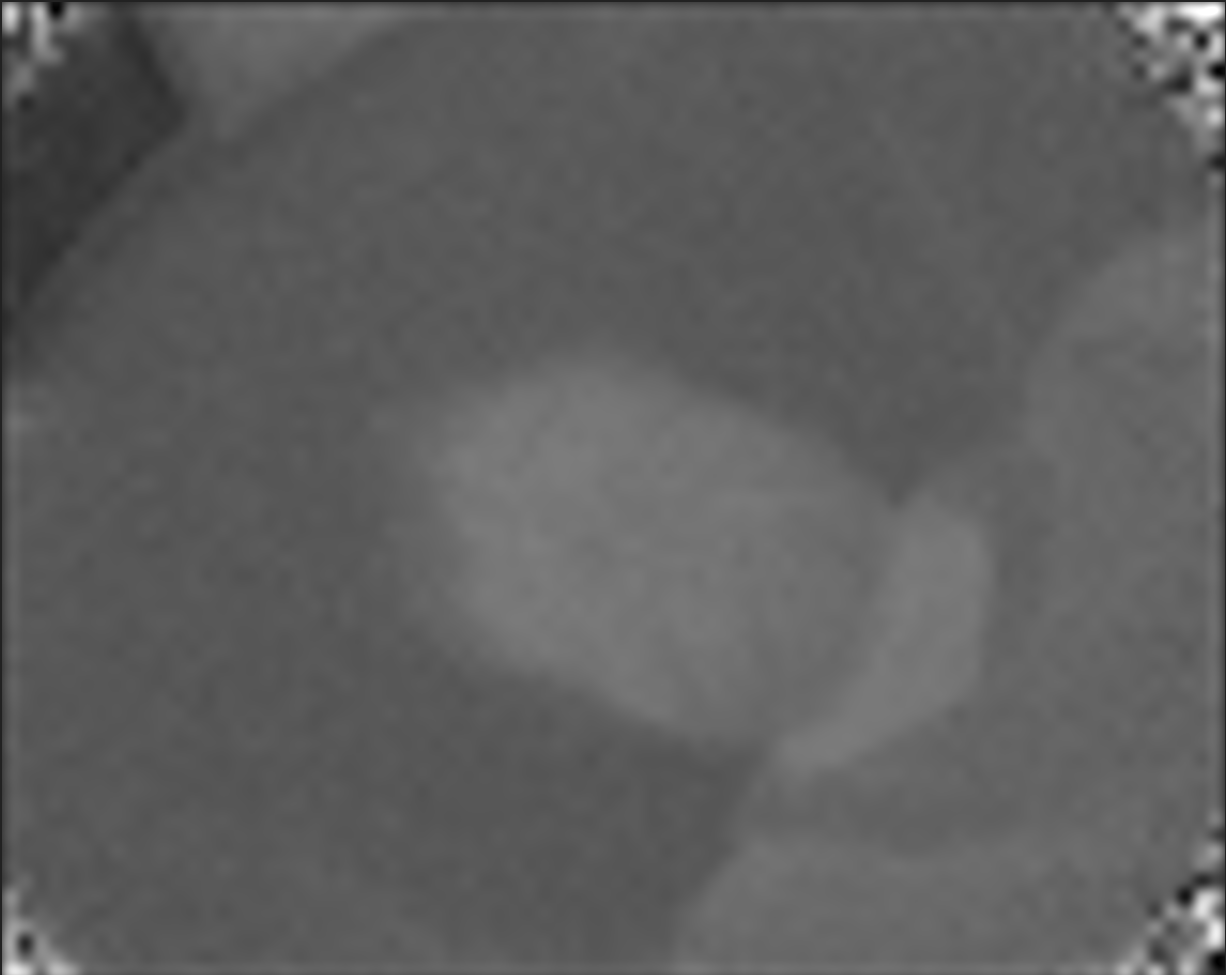
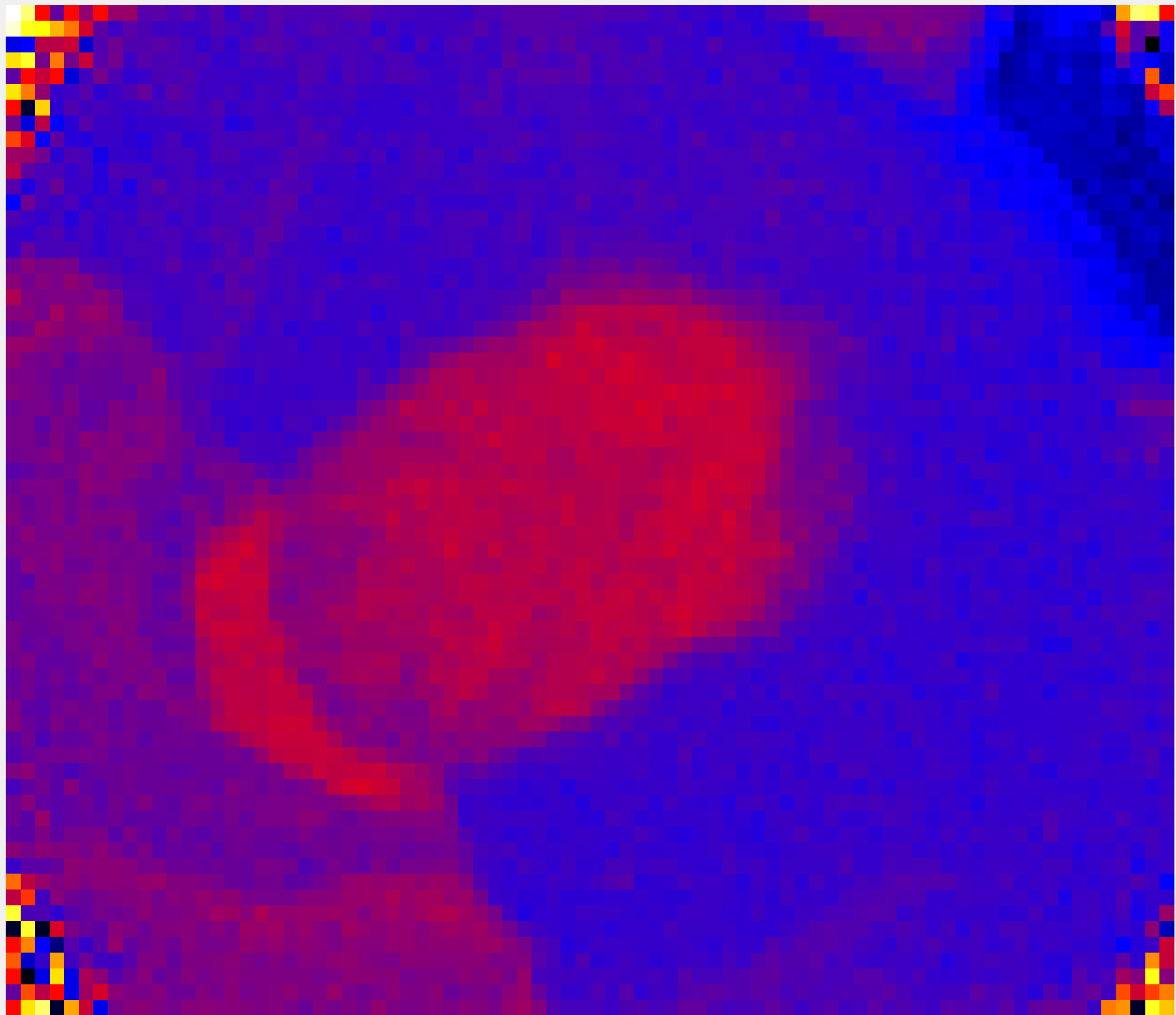
From here we had two approaches. We knew that just comparing the total number of “hot” pixels in a frame to a threshold value will not result in a high accuracy of predicting fevers. If there are hot items in the room such as a hot bowl of soup, warm technology, etc. then our system would determine this is a fever and our system would quickly lose credibility and annoy PSWs. Our two approaches were a statistical approach and an image analysis approach.

#### 3.1.1.3. Statistical Analysis vs. Image Analysis

First, considering statistical analysis, we have the location in each frame of the hot pixels, and we have many frames in the same text file. We considered comparing the hot pixels in one frame to another. If an item moves, it is more likely to be a human, and not something stationary like a hot computer. This is also not a perfect approach, as our system would classify someone walking into the room holding hot soup as a fever. We then considered only looking for the head of someone when taking the temperature. It is extremely unlikely that an item like hot soup will be held at eye level, thus if we try to find a head and measure the temperature of it, we would be able to be more accurate. Without using machine learning, this is difficult. Our initial thoughts were to just examine the top half of each frame, since that would capture the person’s head. If they were on the ground, they would have likely fallen, and our fall detection system would have found this. This again wasn’t the best solution because it wouldn’t work if the person were to lay down.

Finally we also considered calculating the distance between each “hot” pixel using a method like theEuclidean Distance Algorithm. We could determine whether the hot pixels are likely a human, because a human head would be the warmest part of the body, and there would need to be a large number of “hot” pixels all within a short distance of each other to be a human. This was the most plausible statistical solution but this would produce results that would cause a lot of misclassifications so we decided to move away from statistical analysis.

Next we considered image analysis. This is much more dependable but we figured it would be hard to get high quality reconstructed images. The problem with the output files from the ArraySoft V2 software is that they are .bds or .txt and not image based so we are not able to feed an input image into a machine learning model and classify the image as a fever or not. This is the most reliable way to implement our ideas, so we tried to reconstruct the images based on the text file. We were able to take each deci-Kelvin value from the text file and put it into a numpy array. We were then able to reshape the array to fit our 80x64 resolution found in the camera. The issue here, however, is that the deci-Kelvin values are very large in number. For a frame in a normal room temperature room, values ranged from 2,694 to 3,471. Because of this, we scaled each value between 0 and 255, and then used the Pillow (PIL) library in Python to reconstruct the image with these scaled values. Because we were only given single number values indicating the temperature of each pixel we were not able to reconstruct a coloured / RGB image. Upon reconstructing the image, we found that the majority of the reconstructed frames were clearly what we recorded (a human head), but about one third of the samples are unrecognizable. Regardless, we tried to feed these reconstructed images into a OpenCV facial detection algorithm, however the algorithm was unable to detect the faces in even the most accurate images. Figures 13 and 14 below show an image from the ArraySoftV2 software (on the left), and the image on the right is the reconstructed image but mirrored. A lot of the information is retained, but this is one of the better reconstructed images and it is hard to tell that there is a face.



Figures. 13 and 14: An original and mirrored reconstructed frame

Because of this, we needed an adjusted approach. Ideally, we would have a thermal sensor that would have a high enough resolution that it would be able to detect human faces. Since we did not have this sensor, we needed to have a workaround. We realized that since the faces were unable to be detected, we would also be unable to perform fall detection on these images as the resolution was simply not high enough. We needed a working prototype so we used an RGB camera for fall detection. Since we were using an RGB camera, this camera had a high enough resolution to detect human faces accurately. Because of this, we thought of a solution for our fever detection algorithm.

The camera scans the room until it detects motion, at which point an OpenCV algorithm looks for a human face. If a human face is detected, there is a screenshot taken. This screenshot can be compared with the thermal image information. We needed to get the frame from the thermal text file that corresponds to the same image frame of the screenshotted RGB image. We then are able to find the coordinates of the face from the RGB image, so we would know where to look on the thermal information to look for a fever.

This is a great workaround and will be able to solve the issue of classifying non-human objects (food, technology) as a fever. However, we had another issue. The thermal text file is processed in batches, and the timestamp associated with each frame is relative to when the thermopile started recording, not to an actual time. Therefore, we are unable to get the matching frame in the thermal file when a face is located on the RGB camera. Of course, in a real-world application we would want to remove the RGB camera, and just have the thermopile. Since we needed a working demo, we simulated a further portion. We recorded a person from the same distance in both a RGB and the thermal sensor, and we manually captured the same frame from both sources. Now we could compare the RGB frame and the thermal frame.

Now that we had the two frames to compare, we needed to resize the RGB image to match the same resolution as the thermal resolution. The thermal camera has a resolution of 80x64, so this can be scaled up to 800 x 640. The width and height of the RGB image is put into a variable in the python file, and then the difference between the width of the images is found, divided by two, and this much is cropped from the left and right of the image. The same process happens for the top and the bottom when finding the difference in height.

Now that the images are the same sizes, the images can be compared. The location of the faces in the RGB image are saved, and then we create an 80x64 array, and this array is filled with the value of 1 when it is within the location of the face in the RGB image, and the array is filled with 0 otherwise (there is no face in this pixel). This array is then multiplied by the array of temperatures, and where there is a face, the temperature will be a non-zero (the temperature) value. Otherwise, the temperature will be zero. We were then able to calculate the mean and standard deviation of the temperature of each face. We wanted to use a gaussian distribution and if the temperature within 1 standard deviation of the mean (so about 68% of the data) was above the fever threshold of 38 degrees celsius, there is a fever.

However for ease of creating a working prototype we got the median temperature instead and compared this median temperature to the threshold of 38 degrees. If the median temperature is over the threshold, there is a fever. Otherwise, there is not.

#### 3.1.1.4. Thermopile Sensor Hyperparameters

Table. 4: Experiment in ArraySoftV2 with Various Hyperparameters.

| Type of Room | IIR | FIR | Number of Pixels Detected | Num of Frames | Pixels over Threshold per Frame | Actual Number of Pixels Over Threshold |
| --- | --- | --- | --- | --- | --- | --- |
| Dark | True | True | 28731 | 46 | 624.6 | 642 |
| Dark | True | False | 30935 | 45 | 687.4 | 642 |
| Dark | False | True | 31434 | 46 | 683.3 | 642 |
| Dark | False | False | 28328 | 45 | 629.5 | 642 |
| Bright | True | True | 33305 | 46 | 724.0 | 720 |
| Bright | True | False | 36352 | 46 | 790.3 | 720 |
| Bright | False | True | 37114 | 45 | 824.8 | 720 |
| Bright | False | False | 36905 | 46 | 802.3 | 720 |

Table 4 above shows some experiments we ran to determine whether or not to use IIR or FIR in dark or bright rooms. In a bright room, having the IIR and FIR on produces the best result (as the pixels over threshold per frame is closest to the actual number of pixels over the threshold for bright rooms), and in a dark room the IIR and FIR should be turned off for best results (as the pixels over threshold per frame is closest to the actual number of pixels over the threshold for dark rooms).

There are various other hyperparameters we could have tuned to improve the accuracy of our sensors, but these would have diminishing returns. These are explained in the “Future Work” section of this report. It should be noted that we manually set the scaling to a maximum temperature of 40 degrees celsius and a minimum of 15 degrees celsius when working on the project. This has no bearing on the textual results, rather it changes the way the feed is displayed on screen for us to test and interpret results.

### 3.1.2. Fully Implemented Features vs. Simulated Features

There is a mix of fully implemented features and simulated features. Our initial goal when starting the project was to be able to run the system on a Raspberry Pi. It would be able to parse all information files, detect falls, host the web server, etc. Upon doing work on the project, we realized that the software (ArraySoftV2) that came with the sensor was only compatible with Windows systems. Raspberry Pi’s operating system is “Raspberry Pi OS” formally known as “Raspbian.” The ArraySoftV2 software was therefore unable to run on the Raspberry Pi unless we were able to emulate the Windows operating system on the Raspberry Pi. We did some work into this and had planned to implement it, but when we started running short on time we had to prioritize our tasks. We figured having a working system on a PC was more important than having a semi-working system on a Raspberry Pi.

Our final implementation is close to what we had envisioned. We have one computer hosting the database and web server while simultaneously looking for falls and fevers.

There are a variety of sub-tasks when considering monitoring for a fever.

Table. 5: Sub-Tasks when Monitoring for a Fever.

| **Task** | **Pre Condition** | **Post Condition** | **Fully Automated?** |
| --- | --- | --- | --- |
| Gather Thermal Information from Thermopile | ArraySoftV2 Working with Sensor Connected and Turned On | Text File Containing Thermal Information Saved | No |
| Parse File Gathering Only Temperatures | Text File Containing Thermal Information Opened | Array of Temperatures Corrected Filled with Temperatures | Almost |
| Convert Pixels to Celsius | Array of Temperatures Accessed | Array of Temperatures Converted from deci-Kelvin to Celcius | Yes |
| Crop Corners of Image | Array of Temperatures Filled | New Array Created with Cropped Image | Yes |
| Convolution Between Normal Image and Thermal Image | RGB Image and Thermal Image of Same Timestamp Saved | Thermal Array is “0” Where Faces are Not Found in RGB Image | Almost |
| Gather Median of Each Face | Thermal Array is “0” Where Faces are Not Found in RGB Image | Median Temperature of Each Face Calculated and Saved | Yes |
| Determine if Each Face is a Fever | Information about Each Face Location Accessed | Each Face Found in OpenCV is Classified as Fever or No Fever | Yes |
| Reconstruct Thermal Image | Original Array of Temperature Information Available | Array Converted Back Into Image to Send to Website | Almost (file name) |
| Send Email to PSW if Fever is Detected | Fever is Detected in a Face | Email is Sent to Corresponding PSW | Almost |
| Update Database and Website if there is a Fever | Fever is Detected in a Face | Incident Table and Website Updated with new Incident | Almost |

Below is the explanation of the tasks from Table 5 that are not fully automated.

“Gather Thermal Information from Thermopile” is not fully automated and needs to be simulated. We realized that the thermal sensor software did not report temperatures in real time. Rather, they are collected and can be processed in batches. We had an idea to automate the system, by using an Arduino Centipede to automatically enter keyboard and mouse inputs to gather the thermal information. Even if we did this, however, we would still have the issue of comparing these temperatures to the RGB image to find the faces using our convolution approach. This lack of automation is a limitation by the sensor manufacturer and is out of our control.

“Parse File Gathering Only Temperatures” is almost fully automated. The only non-automated section is that the script needs the name of the text file to be parsed at the top of the script. This file name currently needs to be manually entered, this issue would be gone using a new sensor that doesn’t communicate through text files.

“Convolution Between Normal Image and Thermal Image” is almost fully automated. It has been coded in a way where the normal image could have any resolution and it will be scaled to 80x64 before the convolution is performed between the RGB and the thermal arrays. However, the section that is not automated is the filename of the pictures that are being convoluted. This is not a long-term issue as we hope to remove the convolution feature and only use a thermal image once we can get a thermopile with a better resolution that would allow us to detect faces without the need of an RGB camera. Also, for the simulation we needed to manually capture an image and the same thermal image. This is because the fall detection (using RGB camera) is in real time and the fever detection is done in batches. We needed to perform the convolution between the images in the same timestamp so the face was in the same location in both images. This was done manually but this again won’t be a problem moving forward with a better thermopile sensor.

“Reconstruct Thermal Image” is almost fully automated. The part that is not fully automated is the output name of the image. This is manually entered into the script. This could be fixed using the concept of primary key. The file name could be the name of the citizen and a number directly after. This number could be the total number of images saved for this user, and everytime a fever or falls occurs and an image is saved, this value could be incremented in the database and the next image could take the next available number.

“Send Email to PSW if Fever is Detected” and “Update Database and Website if there is a Fever” are almost fully automated. They have the same issue. We currently check the database to find the PSW associated with the citizen, however the citizen’s ID must be manually entered at the top of the script in a variable in order for this to happen. There is not a work-around for this. Each time the system is installed in someone’s room or house, there must be a variable set for citizen ID that is unique to them represented in the database.

There are a variety of sub-tasks when considering monitoring for a fall.

Table. 6: Sub-Tasks when Monitoring for a Fall

| Task | Pre Condition | Post Condition | Fully Automated? |
| --- | --- | --- | --- |
| Fall Detection Using Video File | Pre-Recorded Video of a Fall or No Fall | Video Classified as Fall or No Fall | Yes |
| Fall Detection Using RGB Camera as Input | Working RGB Camera in Use | Video Classified as Fall or No Fall | Yes |

All tasks in Table 6 are fully automated so no explanation is needed as to why these features aren’t fully automated.

This was all implemented with a modular frame of mind. There is the main program that executes each of the tasks above, but actual tasks are completed in their own functions. Some of these functions have further functions that perform different operations or gather different information. Doing our implementation this way allowed us to be able to create our main program from the beginning, and then we just needed to implement each function, similar to a framework with hooks to implement. Completing the project in this way allowed us to not overlook some features. We could see how fast we were implementing, we could test each module separately, we could localize bugs and defaults easier, and we could easily add new modules in by adding a new function and function call. It also enhanced the readability and understandability of the code so anyone with domain knowledge or a background in software development can easily understand how it works.

### 3.1.3. Full Physical Prototype

Figure 15 below shows our finished prototype. The sensor is mounted on the right side of the wooden box, strapped in by adjustable velcro to change the angle of the sensor. On the left there is the Raspberry Pi. The box is enclosed with plexiglass so if the box falls or tilts, the hardware should stay within the box. This box is about 5 pounds total and can be held up to the wall with 12 pound command strips which do not leave residue when properly removed from the wall.



Figure. 15 : Image of our prototype to be mounted to the wall.

# **Chapter 4**

## **4.1. Testing and Evaluation**

This project uses multiple machine learning models in order to run the numerous scripts that we need for the features such as the fever and the fall detection. Since we are using machine learning we had to complete an amount of testing that may not be necessary for other non-machine learning systems. The testing strategy we used was batch testing throughout the development process, after one of the models had been trained it needed to be tested before it could be improved. The model would then be tested with many test cases for each of the possible outcomes. Once we had run all the test data through the model we could then measure how accurately the model performed and then using that knowledge improve the model before testing it on another batch of test cases.

### **4.1.1. Fall Detection Test Cases**

In the first iteration of our fall detection model, we were using the thermal sensor to detect a fall thus we needed a machine learning algorithm that worked with thermal images, for this we used a MobileNetV2 model. Once the model was trained we performed a sanity test to make sure the model was working correctly and then went on to test the model with 10 falls and 10 non falls to obtain the accuracy of the model. This model type was changed due to the thermal sensor quality being low thus impacting the accuracy of the model. Another reason the model was changed was due to the mechanics of how the thermophile sensor worked. For us to process the thermal video we needed to manually get the video file as the output of the sensor through the user interface. Doing so would result in the fall detection not being real-time since we would need to stop the system to get the video file and then pass this to the fall detection algorithm. For this reason, we switched to an RGB camera to detect falls which was our final iteration.

In the final iteration of the fall detection model, we decided to use an RGB camera instead of the thermal sensor thus we were able to use OpenCV which is a lightweight way to process images which is important when looking to import this algorithm into the raspberry pi. Still, we needed a model that would detect the human and figure out if a fall occurred or not and this would need to happen in real-time. For this, we used a haar cascade file which is a file containing the weights of a trained machine learning algorithm; this removed the need to train a model. Once the model was ready we performed the following test cases.

Test Case 1 (Running the fall detection algorithm with video camera as input)

**Item to test:** Can the algorithm run using a video camera’s feed as a continuous input?

**How to test:** Run the python file containing the main function with the built-in video camera as the input to the OpenCV method *cv2.VideoCapture()*.

**Results:** All passed.

Test Case 2 (Running the fall detection algorithm with a video as the input)

**Item to test:** Does the algorithm work by passing a video as the input?

**How to test:** Run the python file containing the main function but pass a video to the OpenCV method *cv2.VideoCapture()*.

**Results:** All passed.

Test Cases 3 to 103 (Find performance of the fall detection algorithm)

**Item to test:** Test the accuracy of the fall detection algorithm.

**How to test:** Perform 50 falls and 50 non falls in real-time and note down the results of each test.

**Results:** 78 tests passed, 22 tests failed resulting in a score of 78/100 giving us an accuracy of 78%.. For test cases, 3 to 103 a confusion matrix was made to visualize the results of the tests that were completed, this can be seen in Table 5. The confusion matrix shows the number of correct and incorrect predictions made by the fall detection model. Our model was able to correctly classify.

Table. 7: Confusion matrix for fall detection test cases.

|  |  | **Predicted Condition** | |
| --- | --- | --- | --- |
| **Actual Condition** | **Total test cases 100** | **Fall** | **Non Fall** |
| **Fall** | **36** | **14** |
| **Non Fall** | **8** | **42** |

### **4.1.2. Fever Detection Test Cases**

Test Case 1-5 (Gathering Thermal Information from Thermopile)

**Item to test**: Does the text file contain the correct number of frames?

**How to test**: Record for a set amount of time at a known frame rate, compare number of frames in text file to expected number of frames.

**Result**: All passed.

Test Case 6-10 (Parse File Gathering Only Temperatures)

**Item to test:** Are “saved temperatures” the actual temperatures? Do voltages get captured as temperatures? Is first line of text file (which is a string) incorrectly gathered?

**How to test**: Compare first and last temperature in text file to first and last temperature in saved temperature array. If they are matching, all tests pass.

**Result**: All passed.

Test Case 11-15 (Convert Pixels to Celcius)

**Item to test:** Do the pixels change to correct floating point value?

**How to test:** Manually calculate the celsius temperature and compare it to the computed value.

**Result:** All passed.

Test Case 16-20 (Crop Corners of Image)

**Item to test:** Are the corners of the image dropped and the rest of the image the same?

**How to test**: Compare thermal reconstructed full images with cropped thermal images to see if the cropped version is simply a subset of the original image.

**Result:** 4 tests passed,1 test failed. The test that failed was because we cropped too much of the image and when we scaled the image back to a 0-255 color range it misconstrued the image. This is not a problem as we do not crop this much of the image anymore.

Test Case 21-25 (Convolution Between Normal Image and Thermal Image)

**Item to test**: Does the convolution correctly identify in the thermal array where the faces are?

**How to test:** Test with different numbers of faces, ensure there is a “1” in the array where a pixel is in the camera image. To perform this, scale the camera image down to 80x64 and perform the convolution, then compare where the non-zero values are

**Result:** 4 tests passed, 1 test failed. The first test failed where we had 2 faces and the system only performed the convolution for one face. A for loop was added to fix this for future tests and they passed.

Test Case 26-30 (Gather Median of Each Face)

**Item to test**: Is the calculated median temperature of face accurate?

**How to test:** Compare calculated median to individual temperatures found in ArraySoft

**Result:** All passed.

Test Case 31-35 (Determine if Each Face is a Fever)

**Item to test**: Which faces in frame are fevers?

**How to test:** Multiple people in frame, some over fever threshold, some not, compare to output of algorithm.

**Result:** 4 tests passed, 1 test failed. The test that failed is because of the temperature drop off of the sensor. When a person over the fever threshold was far from the sensor, they were not correctly classified as a fever. This is a defect out of control.

Test Case 36-40 (Reconstruct Thermal Image)

**Item to test:** Does a reconstructed thermal image look like the original thermal image?

**How to test:** Save thermal information as both an .avi file and a .txt file. Reconstruct a frame, and compare it to the same frame from the video.

**Result:** All passed.

Test Case 41-45 (Send Email to PSW if Fever is Detected)

**Item to test**: Does the correct PSW get notified if a fever is detected?

**How to test:** Try incidents with different patients, check the database to see their associated PSW’s email, check their email for an incident email.

**Result:** All passed.

Test Case 46-50 (Update Database and Website if there is a Fever)

**Item to test**: Is database and website updated correctly when fever is detected?

**How to test**: Check if the database is updated with correct patient information, ensure the website isn’t spammed with incidents if it keeps detecting the same fever, and ensure the correct PSW can view their patient’s incidents and those incidents only.

**Result:** All passed.

Test Cases 51 to 151 (Find performance of the fever detection algorithm)

**Item to test:** Test the accuracy of the fever detection algorithm.

**How to test:** Perform 50 falls and 50 non fevers in real-time and note down the results of each test.

**Results:** 88 tests passed, 12 tests failed. For test cases 51 to 151 a confusion matrix was made to visualize the results of the tests that were completed, this can be seen in Table 6. The confusion matrix shows the number of correct and incorrect predictions made by the fall detection model. Our model was able to correctly classify 88/100 of the test cases giving us an accuracy of 88%. The 2 non-fevers that were classified as a fever but were not is because we put a hot object with the person’s face (laptop, food) for 5 of the tests. 2 of these failed because this hot object took up over half of the bounded box calculated by OpenCV. This was expected, but it failed as it predicted the wrong outcome. This would be fixed with a thermal sensor that can detect a face rather than a convolution with an image. The 10 fevers that were incorrectly classified as non-fevers were incorrectly classified because of the temperature drop off at range. We noticed the sensor has inaccurate readings starting about 5 to 6 feet away from the sensor, and the temperature reported by the sensor gets lower the further the person stands from the sensor. This is an error caused by the sensor and is nothing we can control, although it is good to know so we can aim to purchase another sensor in the future that won’t have this drop-off issue.

Table. 8: Confusion matrix for fever detection test cases.

|  |  | **Predicted Condition** | |
| --- | --- | --- | --- |
| **Actual Condition** | **Total test cases 100** | **Fever** | **Non Fever** |
| **Fever** | **40** | **10** |
| **Non Fever** | **2** | **48** |

## **4.2. Software Quality Assurance Standard**

To ensure the quality of our software was up to certain standards we followed the guidelines of the ISO 9001 [7]. The reason we went with this software quality standard is because of the well-known reputation that it holds for not only software companies but all types of different companies. The way we approached the implementation of this quality assurance standard is by dividing the creation of the project into 4 steps: planning, implementation, monitoring, and improvements. Figure16 below shows the model we are working with that includes all the important steps and leadership in the middle showing the importance of leadership for each task. When it comes to planning, the first task was to define our scope. Since the completion of the system had a set deadline, our scope needed to be a realistic representation of what could and could not be accomplished in the given time period. This ensured that the team was not burdened with completing extra features while not having time to ensure the quality of these features.

In ISO 9001 before moving to implement the system, the leadership roles need to be assigned to each member of the team ensuring that the leader of the project keeps the team on track with structuring quality activities. In our case, each group member was given 3 months to be the leader of the project, thus it was the job of whoever was the leader at the time to ensure the quality of the software was kept up to standards. After and during implementation, weekly meetings occurred to monitor the quality of the software. The quality of the software was checked based on performance, functionality, the total number of errors, user appeal, and most importantly taking into account the requirements that were discussed in the specification document. After each meeting team members had to go back and fix any issues that were discussed and present their improved version in the upcoming quality meeting.

The last step of following ISO 9001 as a quality standard was to improve the software. The way this was done was first improving the performance of the system. Since fever and fall detection are the main aspects of the system, it is required that these two aspects have a high performance for the quality to be up to standards. Thus we would provide improvements to both the systems until we achieved a performance of around 78% accuracy for fall detection and 87% accuracy for fever detection which seemed to be our plateau based on hardware limitations. Lastly, since there is a lot of future work that can be added to the system we needed to make a robust system. This ensures that any upgrades that are required due to customer needs can be satisfied quickly without having large costs.We achieved this by programming in a modular fashion that is easy to add to.

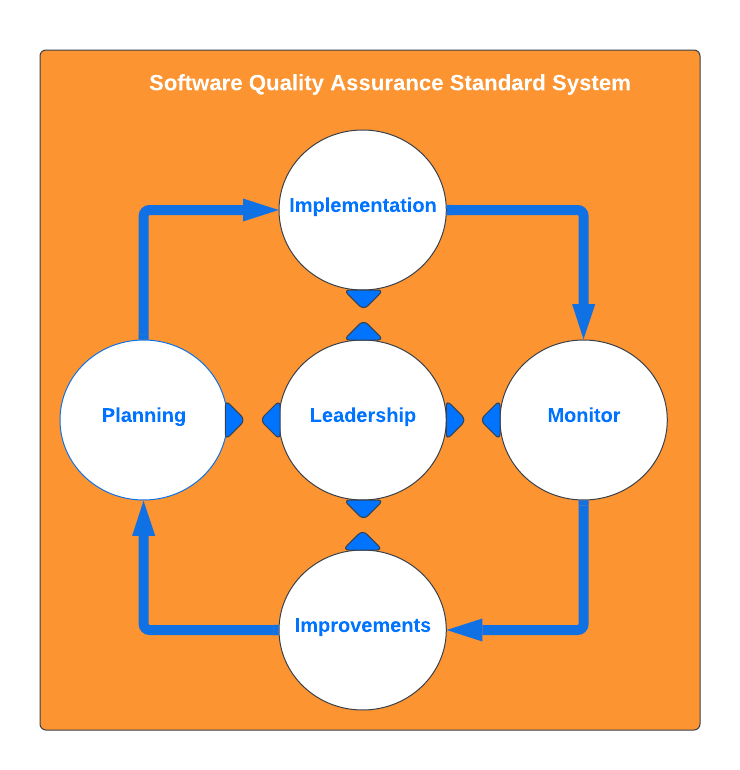


Figure. 16: SO 9001 software quality assurance model.

### **4.2.1. Risks**

Another key step in the ISO 9001 quality standard is accessing the risk, the way this is done in the ISO 9001 is by using the SWOT analysis. The SWOT analysis looks at the strengths, weaknesses, opportunities, and threats of the project as a whole. Table 9shows the SWOT analysis of this project.

Table. 9: SWOT analysis for this project.

| **STRENGTHS** | **WEAKNESSES** |
| --- | --- |
| Team members are responsible and have good programming practices. | Little knowledge using thermal sensors. |
| **OPPORTUNITIES** | **THREATS** |
| Our product can be added to any room in any home. | Timeline of the project can affect the scope of the requirements. |

## **4.3. Existing Models Comparison**

To evaluate our fall detection model we needed to compare it with existing solutions to do this we looked at 5 different papers which provided different solutions to fall detection. Each model type and the accuracy can be seen in Table 10. The first model is a random decision forest that uses depth images that are preprocessed by performing foreground segmentation [1]. This model obtained an accuracy of 96%. The next model being compared is created using SVM and a threshold-based accelerometer to verify a potential fall [2]. This model was trained using depth images and achieved an accuracy of 94%. The third model that we compare our fall detection model to is a convolution neural network where the inputs of this model are optical flow images [3], this model was able to get an accuracy of 95%.

The next two models use a 2D pose estimation to gather information on the human bone data, making the fall detection more accurate since more information about the pose of the human falling is being obtained. One model uses long short-term memory (LSTM) which is a recurrent neural network (RNN) alongside 2D pose estimation to predict falls and has an accuracy of 97% [4]. The last model instead uses a graph convolution network (GCN) with 2D pose estimation and was able to get an accuracy of 100% [5] making it the best model.

In the case of our model, it used OpenCV with a haar cascade file to perform fall detection. We were able to achieve an accuracy of 78%, which makes the model the worst out of the models we are comparing it to. Our model was the worst for a multitude of reasons. The other models spared no computational power since two models used 2D pose estimation and other methods of preprocessing which take a lot of computational power. In our case, we could not afford to do this since our system is to be deployed on a raspberry pi that has limited computation power and we needed to consider the computation power it would take to run the fever detection in parallel. Another reason the other models succeed in getting very accurate results is that they were able to train the models on large datasets by using GPU acceleration. By doing so the time of training the model was greatly reduced, this was also not an option for us since we did not have access to a GPU that would allow for acceleration of the training process. We also had to detect a face before doing fall detection, so any tests where the person was not facing the camera was not detected. This occurred because the thermopile had only a resolution of 80x64 which was not good enough to detect humans, so we needed to use the RGB cameras as a workaround.

Table. 10: Comparing performance of existing fall detection model with our model.

| **Fall Detection Model** | **Accuracy** |
| --- | --- |
| Random Decision Forest using depth images[1] | 96% |
| SVM using depth images[2] | 94% |
| CNN and Optical Flow[3] | 95% |
| LSTM using 2D skeleton[4] | 97% |
| GCN using 2D skeleton[5] | 100% |
| OpenCV using Haar Cascade File (This paper’s model) | 78% |

# **Chapter 5**

## **5.1. Design Impact**

### **5.1.1. Privacy and Security Impacts**

Privacy was a large concern for us throughout the development of this project as it can seem to be quite intrusive having a camera installed inside of your house that other people can access. In order to make sure that the clients are happy, safe, and comfortable we had to come up with a few ways of maintaining their privacy. First off there is no data saved from the devices unless an incident has actually occurred, until this happens none of the footage is saved or streamed anywhere. Next we implemented some access control so that in the case that there was an incident and an image or a small video of them was recorded, not everyone in the system would be able to see it, the only person who would have access is the PSW that is assigned to that client, and the PSW’s supervisor. Finally we wanted to use the thermopile sensor for both of our detection algorithms this way the image is in infrared and there would be no way to tell the person's identity from the image, however the quality of the sensor was not good enough for the facial detection so this is something that would be implemented later on with better hardware.

### **5.1.2. Societal and Economic Impacts**

This project has the potential to make a lot of people's lives significantly safer. Elderly people are very prone to adverse medical incidents and at that age even just a small fall could be devastating. With this system they can have much more peace of mind knowing that if something were to happen to them they would get the help that they need immediately. This will also help the family feel more comfortable putting their loved ones into a long term care home as they too can be assured that they are being monitored 24 hours a day even when the staff may be busy dealing with other things. If this is installed in someone’s private home, the family can have peace of mind knowing that if their loved one has an incident, someone will be notified right away. This is important as some of the elderly population want to maintain their independence and stay in their own homes. Finally it will also save care homes money and save the PSW’s some time as they won't have to spend as much time running around making sure that everyone is doing fine and they can spend that time helping the people in the home that currently need it.

## **5.2. Project Cost**

In order to develop the first unit that we made it turned out to be expensive, most of which went towards the hardware that was used. In total to cover the cost of all the parts: the raspberry pi, thermopile sensor and camera came to $660.41 CAD. This is the normal full price and if we were to go into actual production of the units we would get some kind of a wholesale price for the parts of the system. In terms of development costs based on the size and complexity of the system we estimated that the development costs from start to finished working product would be around $16,992 CAD as found in our SPMP document. On the upside we live in a location with a high elderly population and high demand for a product like this, through some research we found that there are around 1116 long term care beds in thunder bay alone. This is a lot of potential customers. If we sold each of the units at a cost of $749.95 CAD we would only need to sell 190 units to break even including the development costs. This would also not be the final price as it would most likely end up being significantly lower based on the wholesale prices that we would be able to get when working with a manufacturer for the hardware.

Figure 17 below shows the distribution of revenue for sale of one unit. Selling a unit for $749.95 means we need to pay $660.41 CAD in hardware costs, which is 88.1% of the overall cost. The remaining 11.9% of revenue ($89.54 CAD) goes towards offsetting the software development costs. Dividing 16,992 by 89.54 gets 189.77, rounded to 190 units sold to break even. There would be other costs that need to be considered such as overhead costs for starting a business, taxes, legal fees, etc. however there is lots of room to grow when trying to enter the market of people’s personal homes.

Distribution of Revenue for Sale of One Unit.

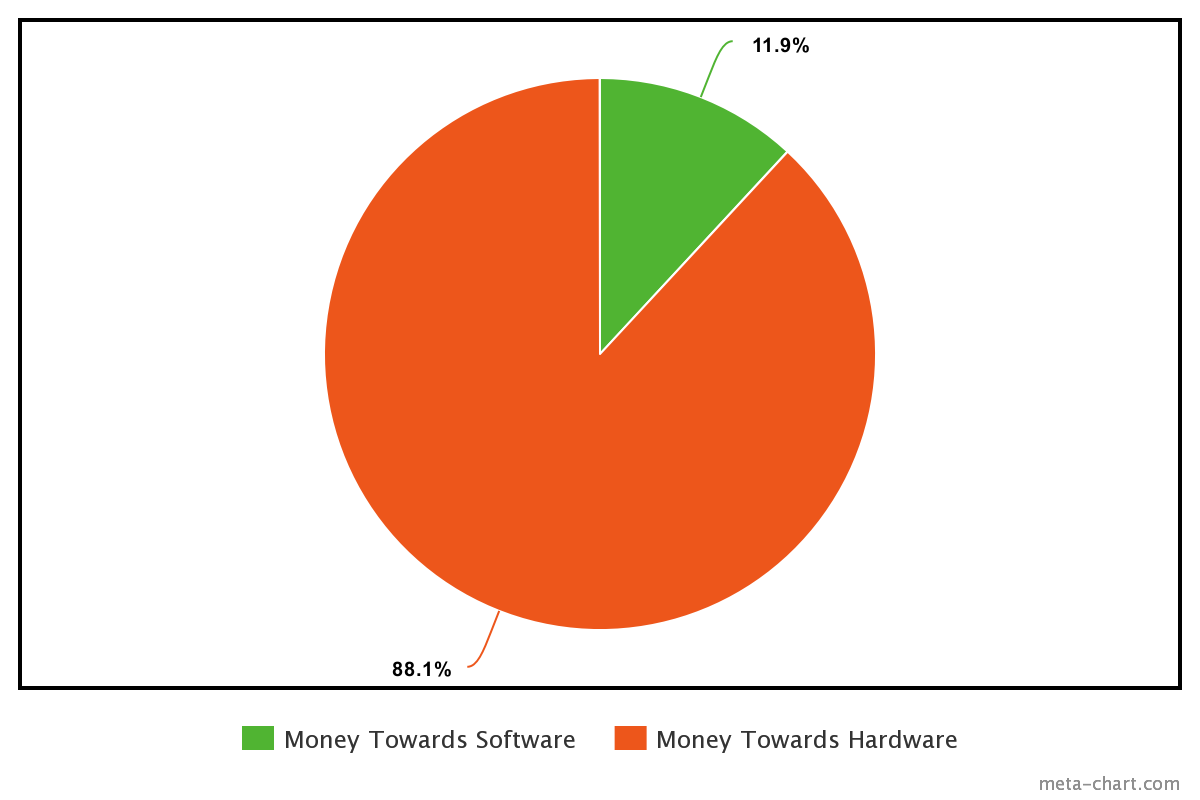


Figure. 17: Distribution of Revenue for Sale of One Unit.

# **Chapter 6**

## **6.1. Problems Encountered**

### 6.1.1. Web Application

Considering the web application, the main problems that were encountered were all mostly due to just a lack of experience working with these languages. Prior to this project we did not have very much experience at all working in html, php and css and although it is a great opportunity to learn and gain more experience, it was not without its difficulties. When it came to creating the interface it was usually not too difficult to get the page 90% of the way to how it is supposed to look but the last 10% of small details and getting it just right was the real struggle. Another issue was working with the tables. In many of the pages the system needed to read the data entries from the database and display all of the data nicely in tables in the web application. There were some issues with the formatting of the tables such as the table moving out of the window instead of scrolling, some of the text in the data entries being cut off, and putting in more data would sometimes change the size of the cells ruining the formatting of the table.

### 6.1.2. Fall Detection

For fall detection the issues encountered revolved around the type of machine learning algorithm to use since we needed an algorithm that accepted sequential data. Another issue with the type of algorithm was that for our first iteration we planned to use the thermal sensor to detect falls thus we needed an algorithm that worked well with thermal images but because of the accuracy being low we finally decided to change to using a normal video camera in order to increase accuracy. The other big issue was that we needed a lightweight algorithm because our final plan is to implement this in the raspberry pi which lacks computational power and storage. There was also the issue of making sure this system worked in real-time, thus we could not use very complex preprocessing methods because this would cause the system to be delayed especially when we had low computational power due to sharing the resources with the fever detection algorithm.

### 6.1.3. Fever Detection

The main issue for fever detection was being able to determine where in the image to check for a fever. We had an original idea of getting the overall number of pixels over the threshold and comparing this to a number, but this cannot be relied on as hot food or technology often has higher temperature readings than people. We then wanted to use OpenCV to detect faces and then only look within the boundaries of the faces, but because the sensor had the poor quality of 80x64, facial recognition algorithms were unable to detect faces using the thermal images. Because of this we needed to create workarounds using RGB cameras and image convolutions to determine where in the temperature information to look for a fever.

Another large issue was the fact that the ArraySoft software saved the thermal information in a .bds file that needs to be manually converted to text files in batches. This makes it quite difficult to create a real-time system. We could have purchased an Arduino Centipede and programmed it to do keyboard and mouse inputs, but this is just a workaround for a sensor and software that is not good enough to bring to market, so we didn’t waste time and resources on this.

### 6.1.4. Hardware Issues

There were various problems encountered while working on this project, however one of the most significant issues was the fact that we were given the incorrect software and incorrect instructions.

Figure 18 below shows the instruction page that came with the sensor. There is a link provided so we followed that link, which brought us to the website on Figure 19.

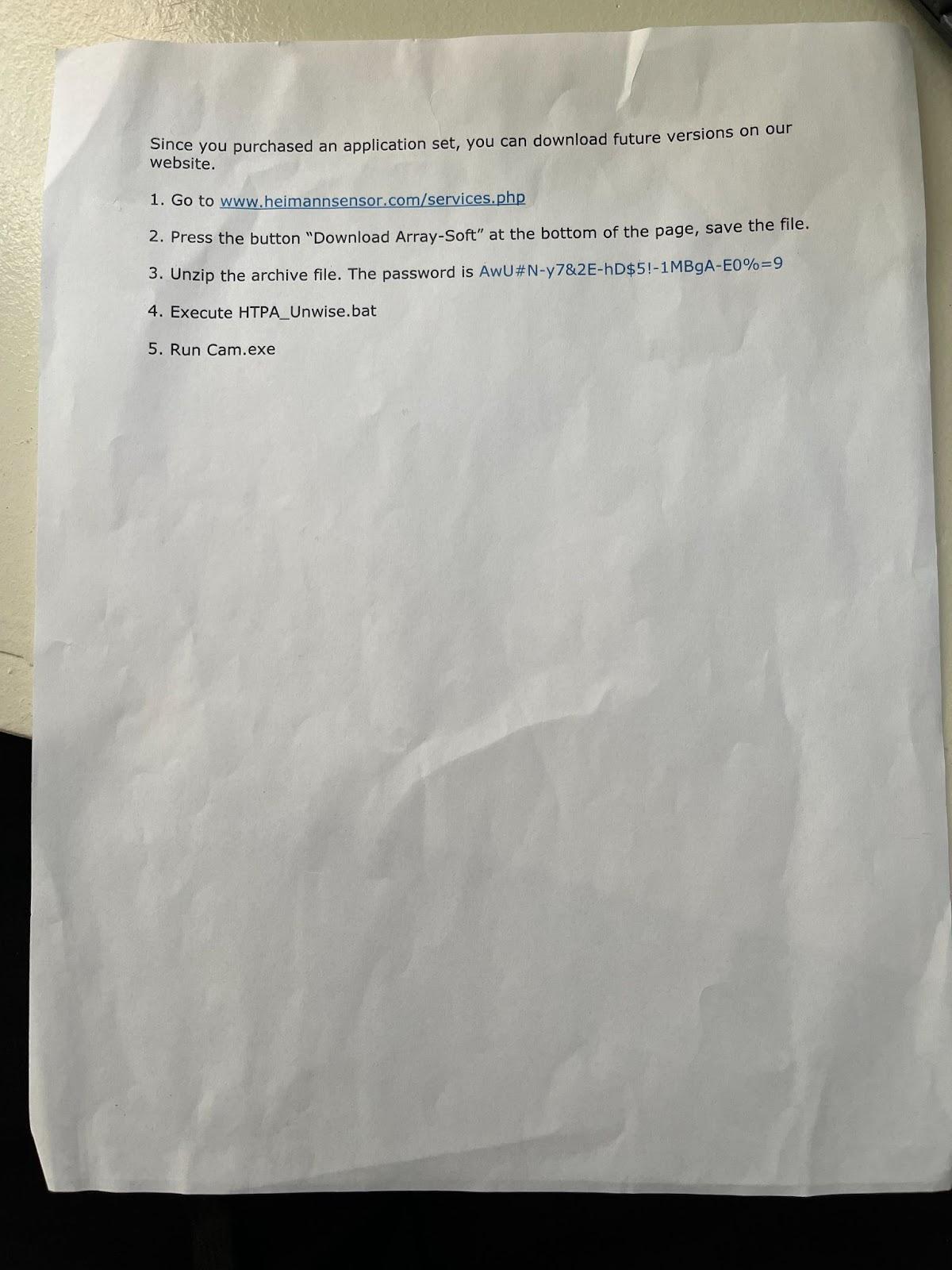


Figure. 18: Instruction Page Included with Sensor.

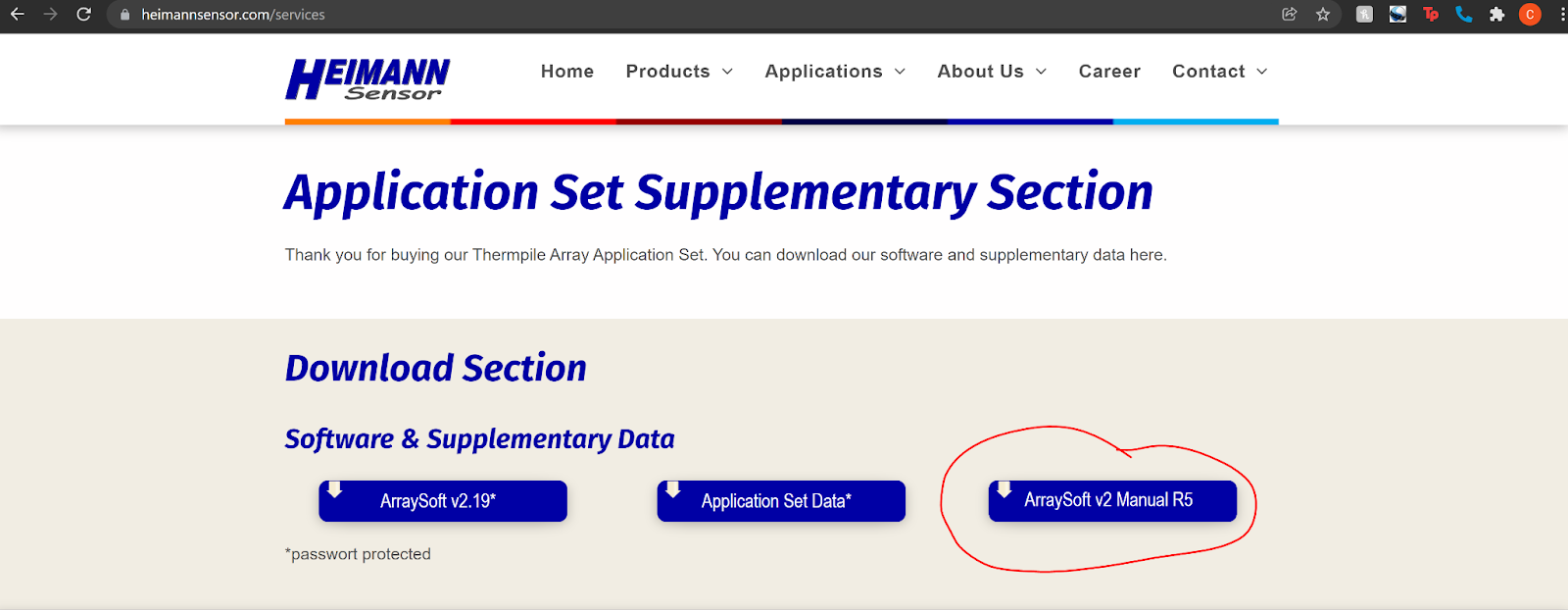


Figure. 19: Website Linked in Figure. 18 Above, Showing “Manual R5.”

The manual from the above Figure 19 matched the physical manual we received. We continued to follow the steps given in the application set for downloading the ArraySoft software, and the version of the software (1.28) can be seen below in Figure 20. We were unable to collect meaningful results from this version 1.28 of the software. We couldn’t figure out why the software wasn’t matching the instructions, and we realized that the version at the top of the window, 1.28, did not match the version listed on the physical and digital manual, which is v2 as can be seen below in Figure 21.



Figure. 20: Version of ArraySoft Downloaded from Linked Website.

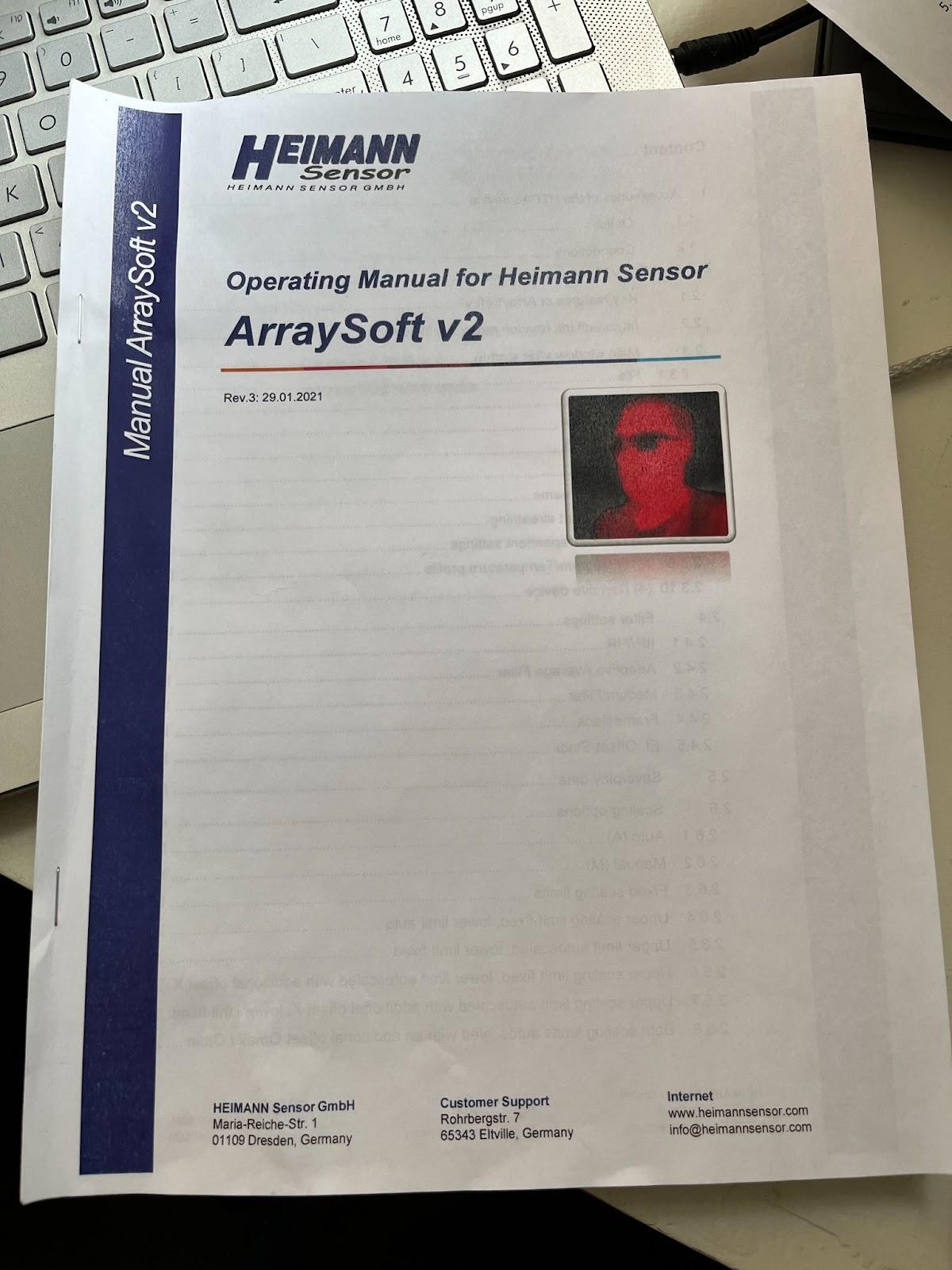


Figure. 21: Image of Operating Manual Included with Sensor.

Within the manual we found a page where we could install Arraysoft v2, so we now followed these instructions. However, the files the manual wants us to copy, “ArraySoftv2.exe” and “GUIv2\_Unwise.bat”, as shown in Figure 22 below, were not found in any of the folders we downloaded from their website.

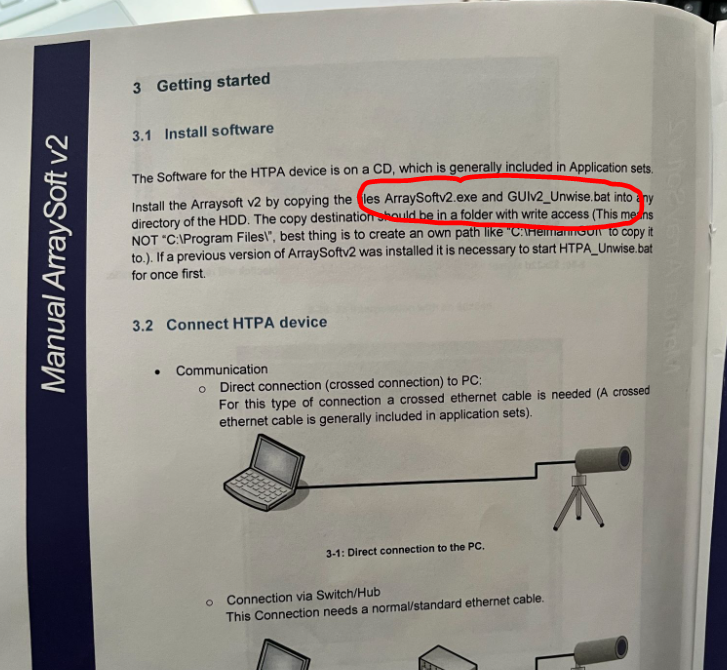


Figure. 22: Installation Instructions from Included Operating Manual.

We were quite confused as nothing was matching up, and eventually we were troubleshooting the issue online. We were lucky to notice an inconsistency across the webpages. Comparing Figure 19 from before, and Figure 23 below, we can see that the manual version is different from each website, changing from “R5” to “R6”. We opened this “R6” manual and after following these steps we were able to install and use the software. As can be seen in Figure 24 below, a different .exe file was given for installation, and this file was indeed found in the files from the website.



Figure. 23: Website Found from Online Troubleshooting, Showing “Manual R6.”

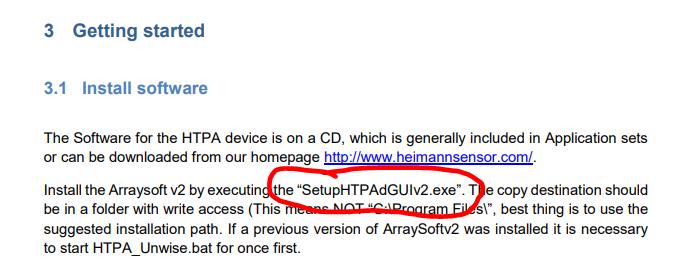


Figure. 24: Correct Installation Instructions from “Manual R6” Found from Website in Figure 23 Above.

In summation, the paper instruction sheet given told us incorrect steps to install the software, the physical copy of the manual had incorrect file names to move and run, and the digital copy found through the link on the paper instruction sheet also had the incorrect file names. The only reason we were able to figure out this issue is by stumbling across the website (which was not provided by the sensor manufacturer at all) in Figure 23 and noticing that the name of the manual was slightly different. Without finding this, we would still be lost.

We also noticed in all of the manuals that it states that we should have a CD for installation (Figure 25 below), as well as a USB HTPA application set (Figure 26 below). These would have been much easier ways to install and use the hardware, but the CD and USB application set were not included with the sensor when it was delivered. We emailed the manufacturer and the re-seller, showing proof that we indeed purchased an application set (see Figure 18) and we asked for them to send us the missing hardware. We never got the hardware and they avoided this when replying to the email.

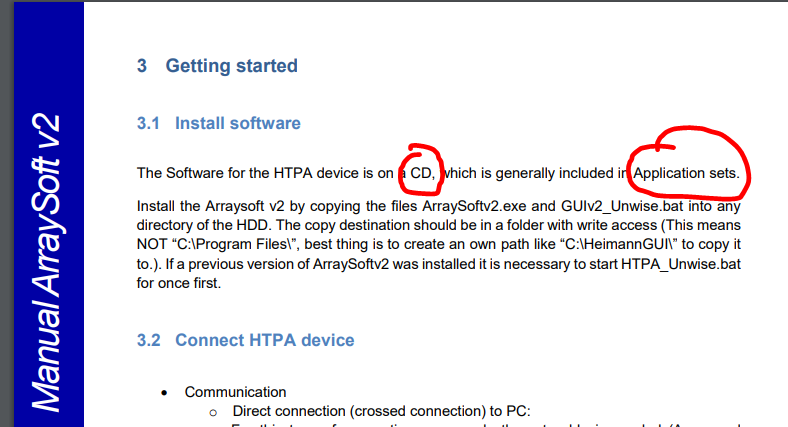


Figure. 25: Showing that we should have Received a CD.

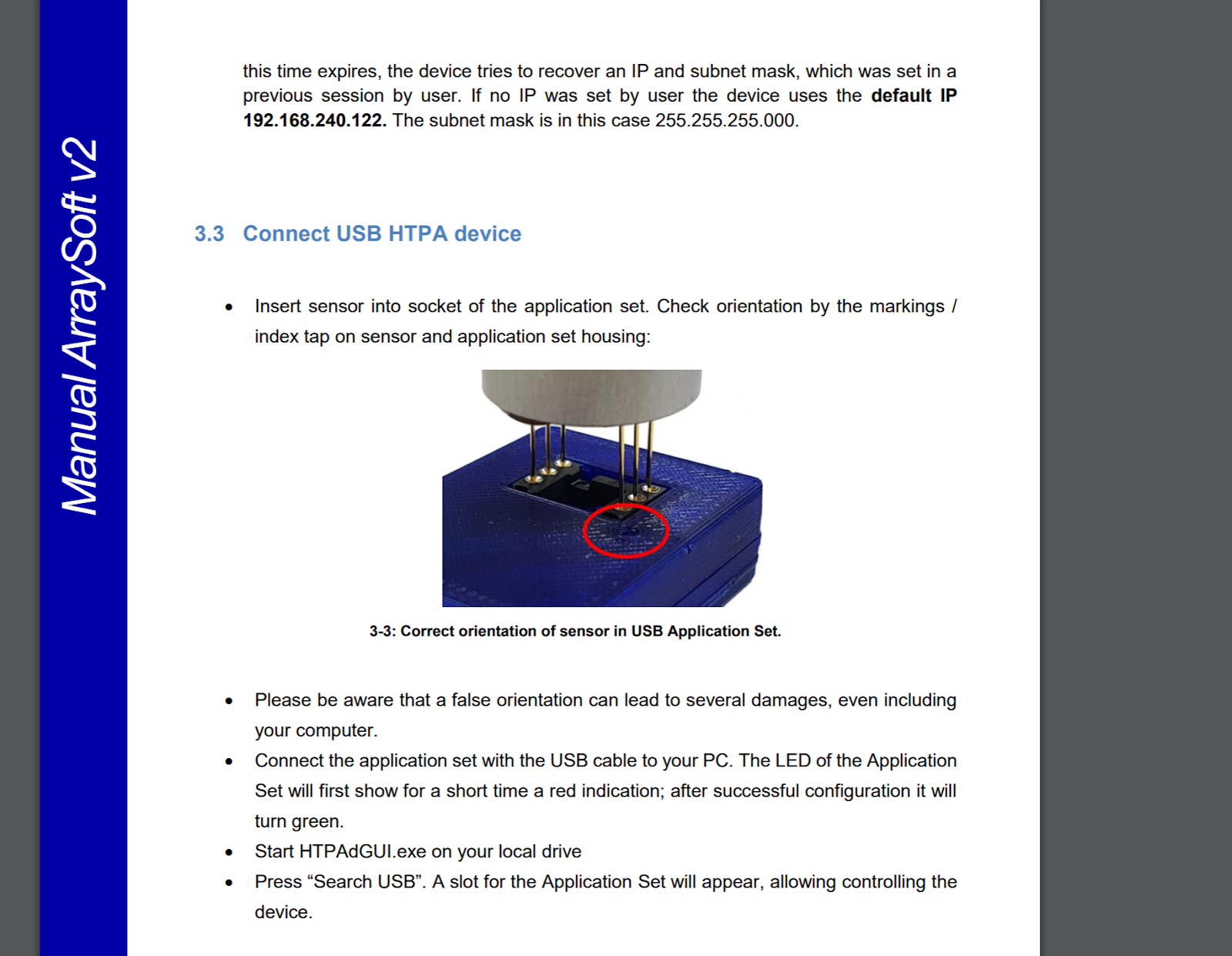


Figure. 26: Showing the USB Application Set we should have Received.

## **6.2. Future Work**

### 6.2.1. Web Application

As for the web application part of the system, what we have so far is capable of doing everything that was completely necessary for the development of the rest of the system however there are many more use cases that could be implemented that would make the system more ready for deployment. One major thing would be to change how incidents are cleared, at the moment an incident can be cleared by a worker once it is dealt with and it is no longer displayed to the psw as an ongoing incident. This could be changed so that when a PSW clears an incident it would then be sent to the PSWs supervisor for review and depending on what the supervisor thinks the incident would then be approved to be cleared or denied and added back into the list of incidents. Another thing would be to assign certain PSWs to a specific supervisor, this would then mean that the supervisor can only deal with their own PSWs and only be able to see the incidents that are assigned under them. This would help to simplify and declutter the incidents page of data that may be irrelevant to one supervisor but not to another.

### 6.2.2 Fall Detection

Looking into the future for fall detection our goal would be to first transfer our existing model into the raspberry pi and perform tests to see how the computation power of the raspberry pi affects the algorithm. From here we would have a working model which we would then iterate again and try to remove the dependency of using the video camera and instead use a better thermal sensor. This would enable us to have a compact system design that only uses one sensor to do all the tasks. Lastly, we would want to have some type of preprocessing of the thermal images since from the Literature Review we can see that preprocessing increased the performance significantly. In order to accomplish this with limited resources, we would have another algorithm that would be used for human motion detection and only once a human is found to be moving will the fall detected be activated. This will allow resources to be saved by not constantly running the fall detection algorithm. Note that this is our design from our diagrams but was not included for the final version of the code submission. The code submission featured each situation (fall and fever) separately in its own section of the code.

### 6.2.3 Fever Detection

Considering fever detection there are a few things that could be improved if we were going to use this sensor in an actual business setting. Since we know we cannot use this sensor if we want to turn this into a business, we did not focus on some of these improvements as some are hardware dependent. An example of this is the various hyperparameters found in ArraySoftV2. It was found that if we adjusted the FIR and IIR hyperparameters we could improve the accuracy slightly. We could have also focused on other hyperparameters like average adaptive filter, median filter, and framestack. There are many more, but we figured that since the sensor doesn’t have a high enough resolution anyways that we shouldn’t waste time for these marginal improvements.

A feature that we were unable to implement was sub-incidents. We wanted to be able to add additional information about an incident. An example would be when there is a fever, the tenant will likely continue to have a fever over time. Instead of creating a new incident each time the sensor detects a human over the temperature threshold, we wanted to create sub-incidents for an incident that will show updated temperatures every few minutes, to track if their temperature is rising or falling over time. This never got implemented due to time constraints, but we had a workaround to ensure that PSWs didn’t get spammed with new reports of incidents for the same fever, so we decided to lock the fever detection algorithm for 6 hours after a fever is detected. This would be removed and replaced with our sub-incident functions provided we had more time.

# 

# 

# 

# 

# **References**

[1] A. Abobakr, M. Hossny and S. Nahavandi, "A Skeleton-Free Fall Detection System From Depth Images Using Random Decision Forest," in *IEEE Systems Journal*, vol. 12, no. 3, pp. 2994-3005, Sept. 2018, doi: 10.1109/JSYST.2017.2780260.

[2] Kwolek, B., & Kepski, M. (2015, June 5). *Improving fall detection by the use of depth sensor and accelerometer*. Neurocomputing. Retrieved April 3, 2022, from https://www.sciencedirect.com/science/article/pii/S0925231215007572?via%3Dihub

[3] Adrián Núñez-Marcos, Gorka Azkune, Ignacio Arganda-Carreras, "Vision-Based Fall Detection with Convolutional Neural Networks", *Wireless Communications and Mobile Computing*, vol. 2017, Article ID 9474806, 16 pages, 2017. <https://doi.org/10.1155/2017/9474806>

[4] S. Jeong, S. Kang and I. Chun, "Human-skeleton based Fall-Detection Method using LSTM for Manufacturing Industries," *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, 2019, pp. 1-4, doi: 10.1109/ITC-CSCC.2019.8793342.

[5] Y. Zheng, D. Zhang, L. Yang and Z. Zhou, "Fall detection and recognition based on GCN and 2D Pose," *2019 6th International Conference on Systems and Informatics (ICSAI)*, 2019, pp. 558-562, doi: 10.1109/ICSAI48974.2019.9010197.

[6] *Requirements management in github with a ... - ceur-ws.org*. (n.d.). Retrieved April 3, 2022, from http://ceur-ws.org/Vol-1525/paper-12.pdf

[7] ISO 9001:2015 - NQA. (n.d.). Retrieved April 3, 2022, from https://www.nqa.com/medialibraries/NQA/NQA-Media-Library/PDFs/NQA-ISO-9001-Implementation-Guide.pdf

[8]Ozdemir AT, Barshan B. Detecting Falls with Wearable Sensors Using Machine Learning Techniques. Sensors. 2014; 14(6):10691-10708.<https://doi.org/10.3390/s140610691>

[9]Fitwi A, Chen Y, Zhu S, Blasch E, Chen G. Privacy-Preserving Surveillance as an Edge Service Based on Lightweight Video Protection Schemes Using Face De-Identification and Window Masking. Electrons. 2021; 10(3):236. <https://doi.org/10.3390/electronics10030236>

[10]A. Gupta, S. Maurya, N. Mehra and D. Kapil, "COVID-19: Employee Fever detection with Thermal Camera Integrated with Attendance Management System," 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2021, pp. 355-361, doi: 10.1109/Confluence51648.2021.9377079

[11]Guan, Ziyi et al. “A Video-based Fall Detection Network by Spatio-temporal Joint-point Model on Edge Devices.” 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE) (2021): 422-427.

[12] E. Auvinet, C. Rougier, J. Meunier, A. St-Arnaud, and J. Rousseau, “Multiple cameras fall dataset,” DIRO-Universite de Montreal, Tech. Rep ´ , vol. 1350, 2010

[13] B. Kwolek and M. Kepski, “Human fall detection on embedded platform using depth maps and wireless accelerometer,” Computer methods and programs in biomedicine, vol. 117, no. 3, pp. 489–501, 2014.

[14] T. Ruffin *et al*., "Non-invasive Low Cost Fever Detection Systems," *SoutheastCon 2021*, 2021, pp. 1-7, doi: 10.1109/SoutheastCon45413.2021.9401821.

[15] Government of Canada, S. C. (2015, November 27). *Health at a glance*. Understanding seniors' risk of falling and their perception of risk. Retrieved April 25, 2022, from https://www150.statcan.gc.ca/n1/pub/82-624-x/2014001/article/14010-eng.htm