SE 101 Introduction to Methods of Software Engineering

Fall 2020

Course Project Final Report

Group Name: five guys

Team Member Full Name	userID	UserName
Harrison Chiu	20897272	h28chiu
Roger Li	20898772	r447li
Ryan Li	20896908	r433li
Ming Ju Yin	20915737	mj2yin
Umar Yousafzai	20890193	uyousafz

1. Introduction

Our product is a portable device that can keep users safer from the threat of COVID-19. The device detects social distancing infractions using a headless Raspberry Pi and PiCamera module and reports this information to the user through an accessible app, which is being expanded into a comprehensive personal health tool. We call it the StayAway.

The StayAway detector is written in Python with the OpenCV library for object detection. Using YOLOv3 weights and configurations, the Raspberry Pi loops the detection code on each image frame of the PiCamera's video feed. It first processes the image with convolutional neural networks (CNNs) before finalizing predictions of the positions and sizes of people in the frame; these predictions are then run through a distance calculator that evaluates each person's position in three-dimensional space and returns the people who are within 2 metres of each other, thereby breaking social distancing rules. Appropriate boxes are drawn on every person in the frame, and this new processed frame is sent to a Google Cloud Storage (GCS) bucket to be displayed on the app.

The StayAway app is built with React Native and Expo CLI on Node. It features a home page and navigation buttons to key pages such as the live feed display, COVID stats page, and news articles page. When the detector activates its PiCamera feed and runs the detection code, the app checks the GCS bucket for an updated image and updates its display with this new image whenever it finds one. This allows the app to offer remote access to a real-time feed of the detector, granting the user a variety of ways to use the detector to manage their safety as they navigate the COVID-19 timeline. React Native also allows users to access the app from any device and receive up-to-date information on their surroundings, as well as other COVID stats and news that are scraped with helpful APIs.

StayAway users can move around and have the StayAway keep track of their contact with other people and crowded areas, or they can opt to use it as surveillance. Whatever the function, the StayAway is easily configured and suits all social distancing applications, which is especially important now as we continue to approach record-breaking numbers around the world every day.

2. Background Research

The assembly of the hardware was mostly based on official Raspberry Pi guides hosted on their website, which detailed not only how to assemble the Pi and PiCamera but how to make the setup headless as well. Third party guides were also consulted for easier methods of achieving these tasks, as we did not require a lot of headless functionality.

For our detection code, our main inspiration was a project article which included source code that was, for the most part, incompatible with our hardware and thus only acted as a boilerplate for our program. Research was conducted on the underlying framework of YOLOv3 and the OpenCV library through articles on YOLOv3 theory, which gave us a better understanding of the template code and allowed us to rework the detection loop to function

properly for our PiCamera feed. Then, comparisons were made with various other real-time object detection tutorials, such as those hosted on PyImageSearch, to determine how to best configure the code for our purposes of detecting people and extracting information on their location; snippets were taken from each of these sources as we saw fit and incorporated into our function. The distance calculator also sourced average male/female heights and relied on basic optics relations to calculate depth.

For the app, we first researched frameworks and technologies that we could use to build a mobile app. We considered how well group members knew the development stack, how intuitive the app development would be, how group members could test the app remotely and how well the stack could integrate other mobile and web frameworks. We first considered Google Flutter, which is a UI toolkit for building mobile and web apps from a single codebase. However, Flutter uses the Dart programming language, which none of our group members had experience with. We also considered creating an iOS-only app using Swift, however, this would be unfeasible as not all group members had access to macOS.

In the end, we settled on using React Native with Expo CLI. React Native offered cross-platform capabilities with iOS and Android, and Expo CLI created an easy way to build cross-platform apps; using Expo, we could test by scanning a QR code and building the app on our phones instead of using an emulator. Node was also mentioned by many to be the best web framework for APIs and component rendering. We looked at some React Native UI kits, hoping to get access to UI elements and pre-built components to bypass styling; however, these UI kits turned out to be quite complicated to implement for our needs and therefore, we decided to code our own basic components to keep the app lightweight and clean.

3. Implementation

3.1. Hardware

The principal hardware component is the device consisting of the Raspberry Pi and PiCamera module, which we call the detector. After flashing the image of the Raspberry Pi OS onto a micro-SD card, initially our Pi did not boot up, leading to us exploring many different methods of installing the OS. However, after switching to a new Pi, this was no longer an issue as the previous SD cards were functional. On the Pi, several libraries and packages were installed including a virtual environment, the appropriate versions of python and pip, the necessary libraries for OpenCV, as well as for the PiCamera. The PiCamera case was put together and then the camera was inserted into the camera slot on the Pi. Once enabled, the camera was tested using camera preview.

The Pi was then transitioned off the monitor and keyboard setup to a headless one. A casing was then fashioned for the detector, which is made from a tissue box wrapped in purple fabric. The box and fabric have snug holes cut into the front for the PiCamera to rest and are padded to minimize drop damage. However, the Pi does rely on its Wi-Fi connection to upload to the Google Cloud bucket, which limits the portability of the product. However, with an appropriately rated battery pack and an initial setup, the product can be worn around and detect what it sees.

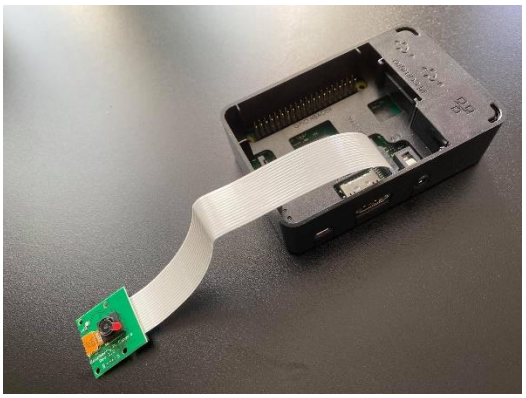
(a)  (b) 

Figure 1. (a) The assembled detector, consisting of the Raspberry Pi (right) and PiCamera module (left). (b) The back of the detector casing with the StayAway logo.

3.2. Software

3.2.1. Detection Program

The detection program employs the OpenCV library and YOLOv3 for the detection of people caught on the PiCamera live feed. We began by scraping source code online that was able to take an existing video file and write a new video file with boxes drawn on relevant objects. We tested various YOLOv3 weights for the CNNs that process images and settled on "tiny" weights to optimize runtime and reduce latency.

Figure 2. A frame written by the detection program with bounding boxes and infractions data in the top left. Observe that partly obscured or distant people have lower confidence scores and are detected less often.

This code was then adjusted to take a live webcam as input, before sending it to the Pi where it would later have to be reconfigured for the PiCamera input. As it turns out, the PiCamera was not a valid input for the method we were calling, so we did some research on more

suitable libraries for the Pi and its camera module. After the code accepted the PiCamera input, we had to adjust some fine details regarding the object detection algorithm that processed each frame, including dimensions of the frame and other information that was needed to avoid errors (of which there were many). We then tweaked the function to store the positions and sizes of the bounding boxes that were drawn in each frame, which we intended to use in our distance calculating function.

Our initial distance calculator, for testing purposes, calculated the two-dimensional Euclidean Distance between the centre of two boxes as the distance between the two people. This was then improved upon to account for depth. In each frame, we take the size and position of the people detected and use this information to then compute their distances from each other in three-dimensional space, accounting for depth by comparing their heights to an expected height. Upon calculation, these figures and the live feed can be displayed to the user and updated in real-time.

3.2.2. App

We first built a skeleton app by coding some basic components, such as buttons and cards, and establishing a layout of the interface that was sufficient to test for integration between the app and detector. We made the interface simple and intuitive with a home page and multiple navigation routes; it was designed to function well on all platforms and to allow routes and features to be easily added later. We proceeded to build on top of the skeleton, adding routes for pages that were expected, such as the video display page, the COVID-19 stats and news pages, and so on. At this time, we used spare routes to test various APIs for video displaying to select the most suitable method of integration, discussed in Section 3.2.3.

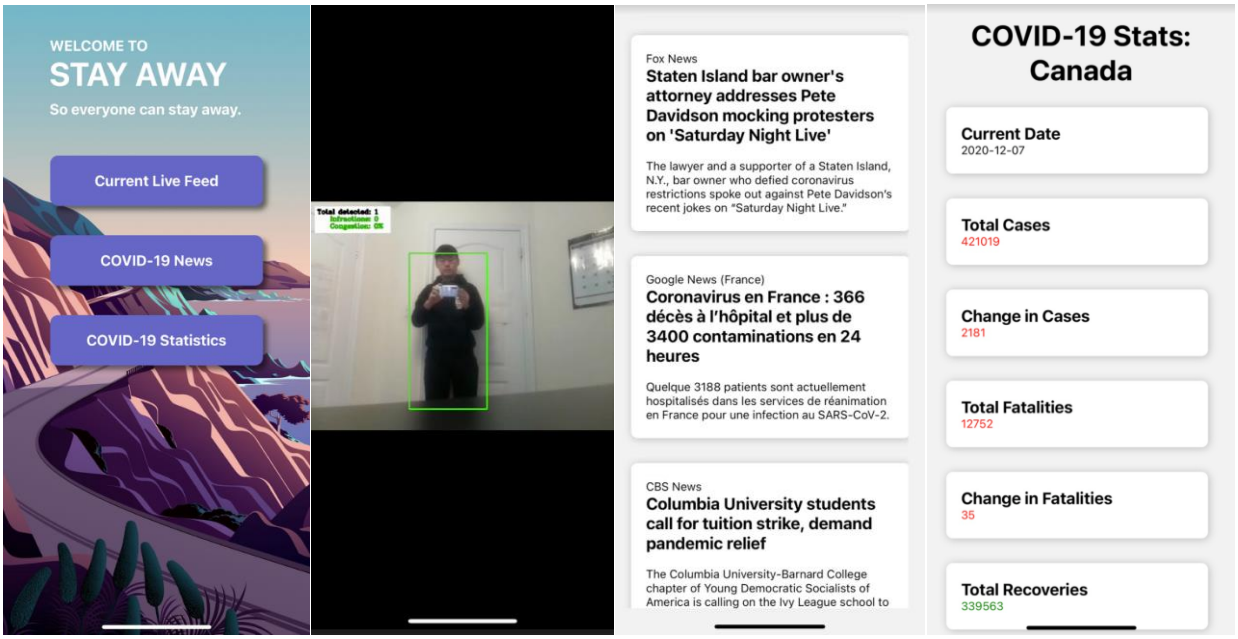


Figure 3. All currently operational app pages, including the home page and each of its routes in order. The live feed is given as a frame-by-frame display. The News and Statistics pages each update daily from their respective APIs.

As the Pi dealt with many troubles, we worked on adding pages that utilize APIs to display relevant information regarding COVID-19. Using the simple fetch function in Node.js offered allowed us to write various REST API calls for things to display on our pages. Most importantly, the COVID19 API is used to make GET requests for COVID-19 statistics for Canada for each day. This includes detailed statistics such as new fatalities and hospital cases, as well as statistics for each province. Furthermore, the News API is used to fetch the top daily headlines from all major news organizations regarding COVID-19. Users can at a quick glance look at the summary of each article or read it further in their browser.

3.2.3. Integration

The bulk of our project relied on developing a low-latency method of transferring information from the detector to the app, including videos/images and social distancing information and statistics. We first tried a variety of methods, including a live streaming API called Mux. We ultimately decided that we would keep processed videos for display on the app, while also offering a live frame-by-frame display which would take the most recent processed frame (with boxes drawn on and social distancing numbers) and update when a new frame was sent. This would relieve some of the expensive processing costs, which took a lot of time when testing and did not respond reliably.

Instead, we hosted a storage bucket on Google Cloud Storage with a free trial. The online setup is quick, and we chose it because it maintains an API that makes it easy to automate GET and POST requests from the detection code and app. After a frame is processed, it is pushed to the storage bucket using a Python script. The pushed frame is automatically made public, meaning that anyone with the link can access it. The app can then fetch this frame upon update and display it in the app. The same process is performed for videos at the end of the run after the video has been processed.

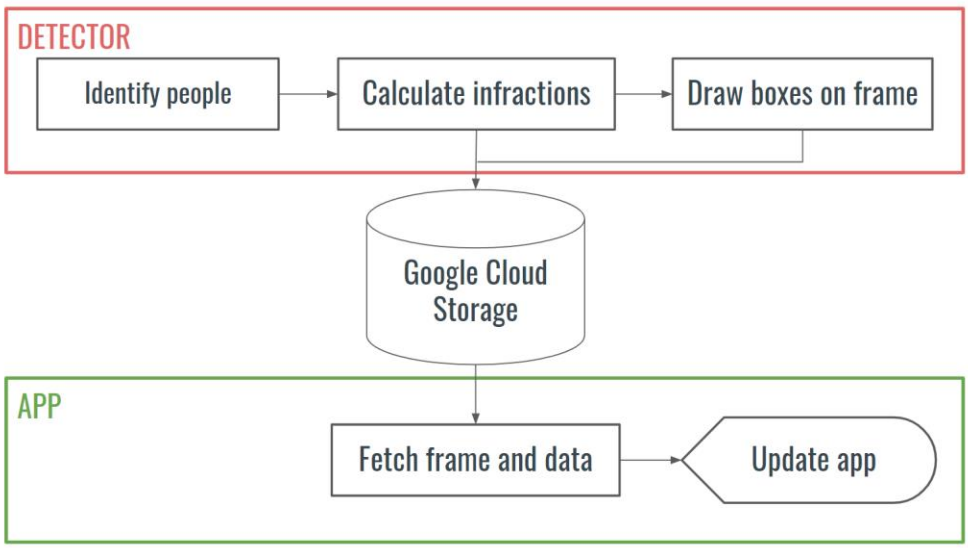


Figure 4. Flowchart of the processing and displaying of a single frame of the live feed. The app fetches and updates its display in real-time as new frames are pushed from the detector.

4. Group members' contribution

- Harrison Chiu
  - Configured detection code for the Raspberry Pi and fixed related errors
  - Optimized distance calculator for three dimensions
  - Tested the detector on the Pi and added features, such as a moving average of "congestion" on the live feed
- Roger Li
  - Assembled hardware and configured headless Raspberry Pi
  - Ran troubleshooting of detection code on the detector
  - Spearheaded integrated testing of the detector and app combined
  - Wrote detector-side API calls for the frame transfers to Google Cloud Storage
- Ryan Li
  - Researched YOLOv3 and wrote the initial detection code
  - Worked with app side to set up frame-by-frame transfer from detector to app and tested the Google Cloud Storage API
  - Tested COVID API calls with Insomnia and searched for APIs for the app
- Ming Ju Yin
  - Created components and routes for the app and built skeleton app
  - Styled user interface and planned video/image displays and other APIs
  - Assisted with Google Cloud Storage API calls and formatting on the app
- Umar Yousafzai
  - Created Google Cloud Storage bucket and wrote API calls for detector and app
  - Helped with app styling and navigation for extra features
  - Added APIs for country COVID info and news articles
  - Helped to test the integrated performance of the detector and app

_____ _____

Harrison Chiu Roger Li

_____ _____

Ryan Li Ming Ju Yin

_____

Umar Yousafzai

5. Final Product Evaluation

The detector performs its intended function reasonably well. That is, it can detect people with relative accuracy and then calculate if they are too close to each other. It is, however, not as reliable for long-distance images with smaller people, as calculations become more skewed by approximations made throughout and the object detection overlooks false negatives because of low confidence scores. This was an expected result in some ways because we anticipated that it would not be failproof, but it is still a point of improvement.

In terms of sending the processed PiCamera feed to an app, compromises were made to accommodate for the high latency and low quality of livestreams, especially when sent from the Pi. In this regard, the product fails to live up to the original goal; however, we also appreciate that the frame-by-frame updating method we devised works as intended with a few seconds of delay. As a substitute for the livestream we had originally planned for, this function works well and satisfies the needs of the product, and the fast reload of the React Native app gives the impression of a live feed update.

For the app itself, we believe that it contains useful features that cater well to our original goal. The app can display each rendered frame live while also providing the user with relevant information regarding COVID-19, helping them in their day-to-day lives. Moreover, the app can be easily scaled up to include more features such as support for multiple users and a COVID-19 screening tool. Compared to our objective, the app satisfies and exceeds expectations with its additional features.

Overall, we are pleased with the overall performance of the product. It falls short in quality in some of the aforementioned areas, but it also exceeds our initial expectations in other areas, and when testing we were able to move the product around and have it update the app to notify the user that social distancing infractions were detected. With subtle tweaks, the product can easily surpass our original goals and expand into a long-term vision.

6. Design Trade-offs

6.1. Mask Detector

The original proposal discussed that, if time permitted, we would try tweaking our algorithm to detect whether subjects in the feed were wearing masks. Upon testing our code on people using the YOLOv3 weights, we recognized that training our own weights would be a costly operation and would not be likely to function well. For one, frames often feature people who are far away, and the camera resolution would make detecting masks very unreliable. This would offer limited benefit, so we instead focused on improving other areas of our detector.

6.2. App Video Display

The biggest design trade-off with the app was displaying the video from the Raspberry Pi. Originally, we wanted to live stream the video directly to the app so that a user would be able

to see the video on their app in real-time as the Pi was running. However, we ran into two major challenges. The first one was that the Pi was not able to render individual frames quickly enough for live streaming to be feasible. Secondly, live streaming required setting up an RTMP server on the Pi which would POST the video to the Mux API. The app would then be able to make an easy GET request and use the Mux video player to display it in the app.

While we were able to test the latter by uploading a sample video to Mux and then displaying it in the app, we were not able to implement the former as setting up an RTMP server turned out to be quite complicated and time intensive. Therefore, we decided that live streaming the video was not a feasible idea and, instead, opted for displaying each frame as an image when it was rendered by the Pi.

6.3. Mask Detector

The transition to a headless Pi was not a main focus when we arrived at that stage of development. In part due to the previously mentioned trade-off with livestreaming, as well as with the required Wi-Fi connection, the Pi becomes more limited in its movement range. Due to the reasonable heft of the Pi itself, and the risk of dropping the product, it was deemed less desirable to manufacture a casing to contain the product and allow it to be worn. As a trade-off, a less sturdy casing was made with the idea that the Pi would not be worn for long distances nor for any motion more rigorous than walking.

7. Future Work

To improve on our current product, we want to make it more compact and accessible for people who go outside. It is currently not very portable, but we are looking to explore the possibility of a more convenient wearable component for it; we also want to investigate putting it on glasses so that it follows that user's field of view faithfully. To go with this, we would like to improve the accuracy of the detector and incorporate the mask detector we discuss above. In addition, we want to improve the process of sending images/videos to have higher quality and less buffering, because the current system is visibly choppy.

Out of personal interest, we want to try live streaming the video by figuring out how to make the Pi render frames faster and how to set up an RTMP server on the Pi. We also hope to add support for multiple users with JWT authentication for API authentication so that it can take on the form of a true personalized COVID-19 health application, after which we would add COVID screening and other healthcare features to assist medical professionals and at-risk populations alike. We had also attempted to use the Twitter API to add features that we expect users would enjoy, but some of the additional requirements of the API made it difficult to implement. If possible, we want to continue to test these and make a more robust application overall.

8. References

(Source Code) https://circuitdigest.com/microcontroller-projects/social-distancing-detector-using-opencv-and-raspberry-pi
(Mask Detector Source Code) https://medium.com/microsoftazure/corona-face-mask-detection-with-custom-vision-and-tensorflow-js-86e5fff84373
(YOLOv3 Darknet Weights) https://pjreddie.com/darknet/yolo/
(YOLOv3 Theory Guide) https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e
(YOLOv3 Theory) https://pylessons.medium.com/yolo-v3-theory-explained-33100f6d193
(YOLOv3 & OpenCV Tutorial) https://www.youtube.com/watch?v=1LCb1PVqzeY
(Object Detection with Deep Learning Guide) https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/
(Threading Model for Raspberry Pi) https://www.pyimagesearch.com/2016/01/04/unifying-picamera-and-cv2-videocapture-into-a-single-class-with-opencv/
(Real-Time Object Detection) https://www.pyimagesearch.com/2017/09/18/real-time-object-detection-with-deep-learning-and-opencv/
(OpenCV dnn Module) https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/
(OpenCV Resources) https://www.pyimagesearch.com/opencv-tutorials-resources-guides/
(COVID-19 Stats API) https://documenter.getpostman.com/view/10808728/SzS8rjbc
(News API) https://newsapi.org/
(Raspberry Pi Setup) https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up
(Raspberry Pi Headless Setup) https://www.raspberrypi.org/documentation/configuration/wireless/headless.md
(Google Cloud Storage Uploading Source Code) https://cloud.google.com/storage/docs/uploading-objects#storage-upload-object-code-sample
(Google Cloud Storage Upload/Download Guide) https://cloud.google.com/storage/docs/uploads-downloads
(Google Cloud Storage Class Documentation) https://cloud.google.com/storage/docs/changing-default-storage-class
(Picking a Mobile Dev Framework) https://www.aalpha.net/blog/how-to-know-the-right-mobile-app-development
(React Native Basics) https://reactnative.dev/docs/tutorial
(React Native UI Kits) https://medium.com/flatlogic/top-react-native-ui-component-kits-a9a3cbf45b04
(Expo CLI Documentation) https://expo.io/tools