GNG 2101 Deliverable J

User Manual

Gabriel, 300129767

Gianluca Coletti, 300065278

Hiruni Senarath, 300044216

Thuy-Vi Le, 300119477

Tony Zhao, 300129784

Due: December 3rd, 2020

University of Ottawa

Abstract

This document will outline in detail the features and functionalities of the device as well as the problem it was intended to solve. It will start by defining the problem clearly and its importance. Further, the document will go into features and functions of the product and the processes and tools used to create it. Any risks to health and safety will be addressed and precautions will be outlined should they occur. Finally, the document will show the conclusions which were derived from the end result and recommend future avenues for work related to the project. All of this will be done using in-depth explanations of the relevant information, a collection of our tabulated results and data, and the relevant design files relating to the course. All this information should make for better use of the product and better understanding of the work and decisions that went into designing it.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

The purpose of this project was to help people who are visually impaired locate elevator buttons with reduced need for touching and feeling around on surfaces. This is important because currently the placement of these buttons is inconsistent with respect to the elevators themselves and feeling around surfaces, the current predominant way of finding them for these users, is dangerous when those surfaces are likely to be covered in germs or droplets that might contain COVID-19. Our solution was a phone app that uses image identification to identify the button using the users camera and guide them to it. Though there are an increasing number of products attempting to help visually impaired users locate, or use remotely, different types of buttons, our product is currently the only one we could identify covering elevator buttons specifically, and our solution has an inherent advantage over remote triggering solutions because, although they forgo the problem of locating the buttons entirely, they also require an additional device to be integrated directly with each individual button. This means not only that they have to make a lot more individual devices, whereas ours is 1 app, but they also have to get permission to implement it from the owner of the button, be it a private organization or the government, which means they have to fight legal battles, that we do not, in some capacity for the implementation of each button.

2.1 Important features, functions and capabilities

The main features of Elevaider is the detection of elevator buttons and guiding the user towards those elevator buttons. The app accesses the IOS device's camera and identifies the elevator buttons based on models made in Core ML and using Apple's Vision framework. The app employs Apple's accessibility feature, VoiceOver, to notify the user when an elevator button has been or has not been detected. The app consists of a single view, making the app efficient and easy to navigate, and can be launched via Siri shortcuts.

At its current stage of development, Elevaider is able to detect LRT elevator buttons in good lighting and from a maximum distance of about 1 meter. It is also able to notify the user whether a button is or isn't detected.

2.2 How prototypes were made

Our first prototype was a wireframe designed for our original application layout (seen in Figure 1). This included the launch screen with approval for camera access, a tap button to track elevator buttons, and a tap button to return to the home screen.

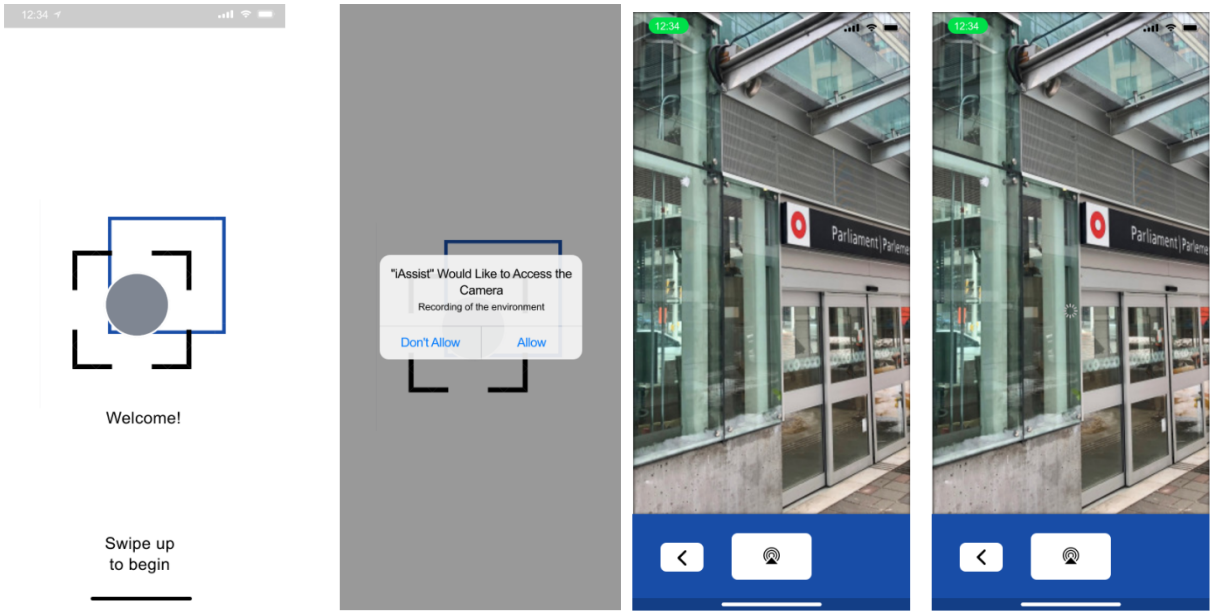


Figure 1. Prototype 1 (Wireframe)

In our final wireframe (seen in Figure 2), we simplified the design further by removing the on-screen tap buttons and including a text message on the screen notifying the user (visually and audible) when an elevator button is in view or not.

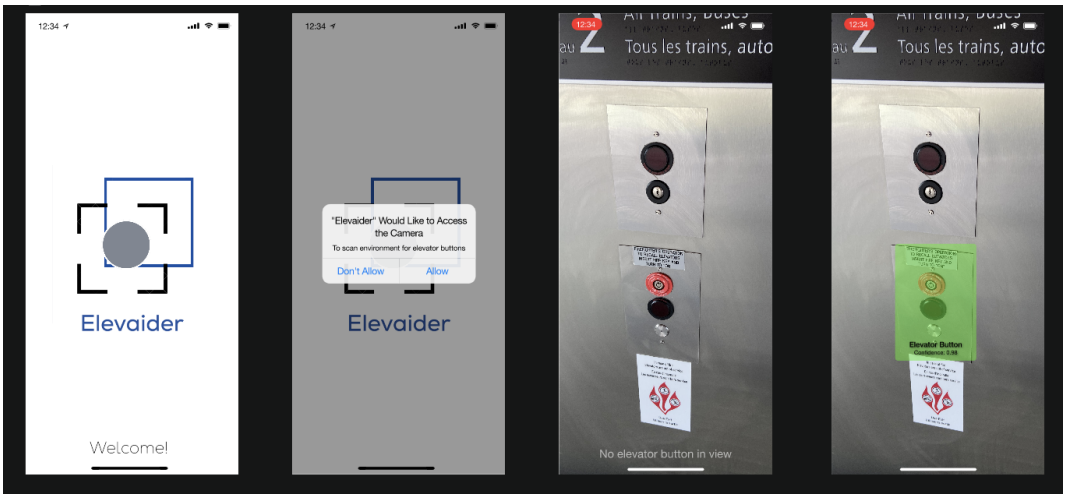


Figure 2. Final wireframe design for our application.

2.2.1 How the Final Prototype was made

To create the final version of the elevator button recognition model we followed these steps:

1. Take pictures of LRT buttons so that we had a dataset of about 200 pictures.
2. Split the dataset into a file containing about 20% of the images as test images and 80% as images to use to train the model.
3. Use Create ML to train the model using the training dataset. Set the number of iterations to 4000, choose Full Network (based on YOLOv2 architecture) and autobatch size.
4. Once the model is complete, you can test it in Create ML using the test images.
5. Our app foundation was implemented from Apple's Recognizing Objects in Live Capture. We then added code which updated an on-screen text whenever

For more information on the construction of the project visit: GitHub_GNG2101_Elevaider

2.2.2 How to Install

For installation of our application, we have chosen to make it downloadable from our groups Makerepo project site (GNG2101-B23-Elevaider) and then can be locally installed onto the phone.

The steps are as follows:

1. Go to Makerepo and click on the GitHub link.
2. Clone the GitHub repository to download the files for the app onto a Mac/iOS device.
3. Open the .xcodeproj file with the Xcode IDE.
4. While the device is connected to the computer, choose the device from Product > Destination and then Product > Run. Make sure that the computer has been trusted.

5. Once successfully built onto the local device, open the app.

For maintenance, the app will have to be updated and re-uploaded onto Makerepo or updated via TestFlight or the Apple App Store if the app does get published in the future.

2.2.3 How to Use

1. Open the app when near an elevator
2. Once given access to the camera,
3. Point your phone in any direction
4. Once the app detects a button, an audible notification is heard
5. If the button falls out of frame, the app will also notify the user
6. To stop the app, simply return to the home screen or close the app

2.3 Health & Safety Precautions

- When using Elevaider please be aware of the space around you
- Ensure that your phone is held firmly
- Try not to use the app in high traffic areas

2.4 Troubleshooting

How come the app isn't working?

- Try rebooting your phone
- Check for an update in Makerepo

Why is the camera not working?

- Make sure the app is allowed camera access
- Try closing the app and running it again

How come there isn't any audio?

- Make sure your volume is on
- Ensure that Siri Voiceover is enabled

How much is your app?

- It is a free application for IOS devices
- To upgrade to premium it costs $1.99

Why is it not recognizing elevator buttons?

- As of right now the app only works on LRT elevator buttons, sorry for any inconvenience

For further information please contact elevaider@gmail.com or please call 1-800-ELE-VAID

3.0 Conclusion

This user manual serves as a guide to users on how to use Elevaider, an app that guides visually impaired users to elevator buttons. It includes troubleshooting ideas for issues with the app, highlights key functionalities and informs users of the health precautions to take while using the app. This manual also details the design process accomplished when coming up with this app to solve the problem of visually impaired individuals needing to locate buttons without touching public surfaces, due to health concerns. It outlines the information received from clients, metrics identified and most importantly the steps taken to create the prototype. The main learning taken when creating the prototype is to make sure your dataset of pictures is concise and of good quality. Otherwise retraining the model can cut into a lot of the design time. Thus this manual is not only useful to future users, but also to future development teams that may want to recreate our app or build up on it using our suggestions for future work.

4.0 Recommendations for Future Work

In general, in the future we would like to improve the accuracy of detecting buttons. This can be done by retraining the machine learning model using a larger dataset of button images with more testing iterations in Create ML. We would also like to expand our product line into other buttons types, such as accessibility buttons or crosswalk buttons. This can be done by applying the same design process we followed for elevator buttons. In short, we would take pictures of the new button types and train a model using Create ML to recognize these new buttons then implement this model within the existing app using Swift.

4.1 Improve Navigation

The current navigation system of Elevaider is in its simplest form. If the button is found on the screen, the user will be informed to move forward. In the future, to be able to give users

more specific directions we would like to split up the screen into 5 regions (as seen in Figure 3). Thus, if the button were to be located in one of these regions, the app would give you more specific directions on where to go depending on the region it was found in. For example, imagine the button was found in region 4 (as seen in Figure 3), the app would tell the user to move to the right and down.

Figure 3. Phone screen split into regions to improve navigation system

Our current machine learning model already provides the x and y coordinates of any object it detects relative to the phone screen. Thus, to implement this new navigation system we can use these coordinates and then write code in Swift to set different areas of the screen as regions and indicate what the navigation response should be when a button is found in each specific region.

4.2 Increase the Distance the App is Usable From

Currently Elevaider can only detect elevator buttons when the user is about 1 meter away from the panel. Our assumption for why we are limited to this distance is because a majority of the images taken for the dataset that our button detection model was trained on, were taken from a 1 meter distance. Thus, the machine learning model was mainly accurately trained to detect buttons from only this distance. This isn't ideal because you would already need to have a pretty accurate idea of where the button is to start using the app, which is counterintuitive.

Therefore, a very important future development for this app would be to increase the radius that it's usable from, preferable to about a 6 meter distance. This can be done by retraining the button detection model in Create ML using a new dataset of images taken from further distances away from the elevator button.

5.0 Appendix

Table 1. Client Statements

Meeting	Statements
1	● Main problems ○ Touch is the most used sense but it is dangerous touching everything with COVID. ○ Difficulty with distancing because sometimes they take someone's arm and generally they don't know exactly where everyone is all the time. ● App/wearable technology that press automatic buttons for them. ● Buttons can be located anywhere in the vicinity of the related object. Inconsistent and may be obstructed. ○ Crosswalk button issues are not exclusive to only blind/low-vision people and can also affect people in wheelchairs during the winter. ● Usable while wearing gloves and in winter. ● Should be functional with only one-hand (because usually one hand is preoccupied with other tools, example: Cane) ● Wants a wearable device because having just an app brings the problem of potentially losing the phone or damaging it when it constantly gets pulled out to use. Having a cheaper wearable product would allow customers to use it without the free of having to spend a lot of money to replace it if needed. ● Cheaper, below $50. She wants it not to be too expensive so that more people have access to it. ● Product could be set to different notification options. ○ She said for areas she is more familiar with a vibration is good but in a new area a speaking function would be best. ● Customers could map their route beforehand. ● Could have a product that can link to an app via bluetooth. ● Key2Access tried to solve the issue before (required replacement of Ottawa crosswalk call buttons). ● Client says there are no current similar products that she endorses or has used
2	● Sensors were used for people with disabilities that restricted their range of motion. ● Audio options with accessibility ● Personal devices at most frequently visited places (Offices, home, etc.) ○ Tile like devices to track accessible buttons

	** More interested in development of indoor door accessibility (used example of elevators at Parliament LRT station).* ● Malls/doors inside buildings ● Elevator buttons ● Tight/crowded places ● LRT station with elevators ● Mentioned using motion detectors ● Beats headphones have option to cycle through different songs ○ Can implement that except for crosswalk buttons to select which button to push ● Can have different notifications for different buttons ● "Luggage locator" ● Movable locator ○ Problem: theft ● Mentioned using Blindsquare, an app used by people with little to no vision ○ Uses pause & play buttons to list options/choose option ● There are sensors for people in wheelchairs that detects specific motion movements or the specific area they would approach from ● The pro to having the device incorporated into the phone is that you only have to carry one device versus the con of having to use battery life ● Look into blindsquare usability (liked how it works) ● Likes the idea of having a personal device that you can keep on you or move around throughout the day ● Want to focus on elevator buttons
3	● Our client relies heavily on Apple voiceover so compatibility with it would be a great asset. ● No need for initiation sequence/button to start, client would prefer if it started finding buttons automatically after opening the app. ● Elevator buttons are so close together that the use would touch it anyways to differentiate between the different buttons. Thus an app that tells just the general location of the panel is good enough and then users can find the specific button on their own. ● Leave the app running even after the user finds the button in case they want to relocate the button. App will only turn off once the user hits the 'home' button or uses Siri shortcut to close the application.

	• Seeing AI (flip to product mode) is a good example of how instructions should be relayed.

Table 2. Prioritized Needs

#	**Need**	**Importance**
1	Is usable with one hand	5
2	Can be used while wearing gloves	4
4	Provides notifications by vibration or audio cues	5
5	Allows pre-mapping of routes	2
6	Can be integrated with smart devices	4
7	Is affordable for everyone	5
8	Is easy to learn to use	5
9	Is affordable to replace if damaged or worn out	3
10	Eliminates the need to touch surfaces to locate objects	5
11	Resistant to damage from accidental dropping	2
12	Is resistant to cold temperatures so it's functional in the winter environment	3
13	Functions as normal in rain	3
14	Targets a range of people with disabilities	2

5 = Most Important 1 = Least Important

Table 3. Metrics

#	Need #	Metric	Importance	Units
1	7, 9	Cost	3	$
2	1, 2, 3, 8, 14	Usability	4	1- Very user friendly 2- Moderately user friendly 3- Not user friendly
3	2, 6	Phone integration	4	# of platforms
4	4	Time to notify user	5	Time delay (s)
5	4	# of notification systems	5	# of ways to notify
6	5	Route planning feature	2	Y/N

5 = Most Important 1 = Least Important

Table 4. Target Specifications

Target specification	Unit	Accepted (marginal)	Ideal
Cost	$	≤ 100	≤0
Usability	1- Very user friendly 2- Moderately user friendly 3- Not user friendly	2	1
Phone integration	# of platforms	1	1
Time to notify user	Time delay (s)	≤ 15s	≤10s
# of notification systems	# of ways to notify	≥ 1	≥2
Route planning feature	Y/N	N	Y

Table 5. Bill of Materials

Item Number	Part Name	Description	Quantity	Unit Cost (CAD$)	Extended Cost (CAD$)
1	Publishing Fee	Uploading App to the app store	x1	(131.22)	(131.22)
2	Swift	This was used to program the app and houses the libraries used.	x1	0	0
Total					0(131.22)

$131.22 represents the cost of publishing the app on the Apple Store but we did not get one as it exceeds the limits of our budget, though that would be a logical next step.

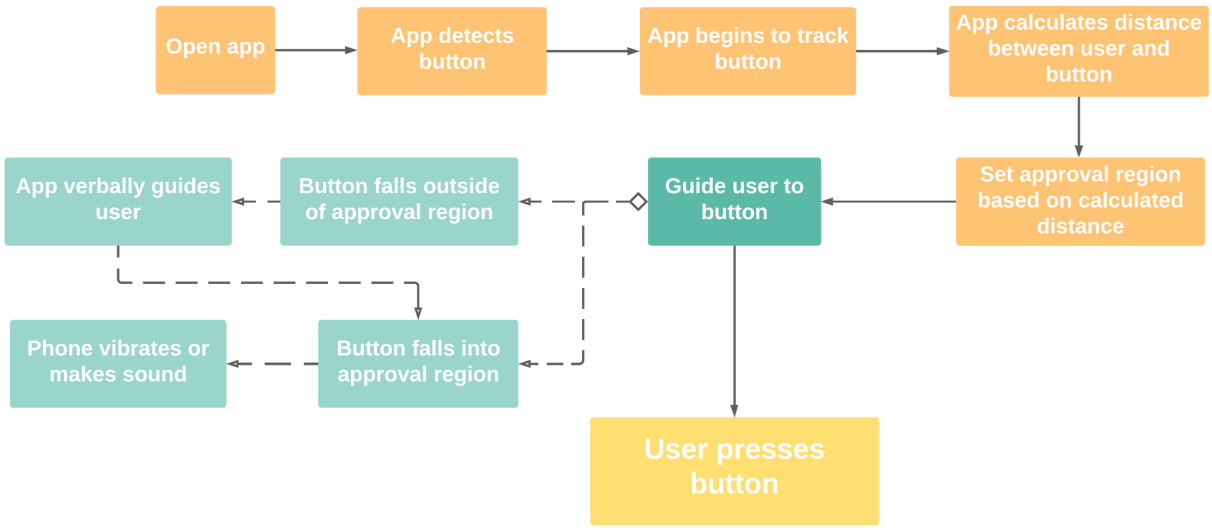
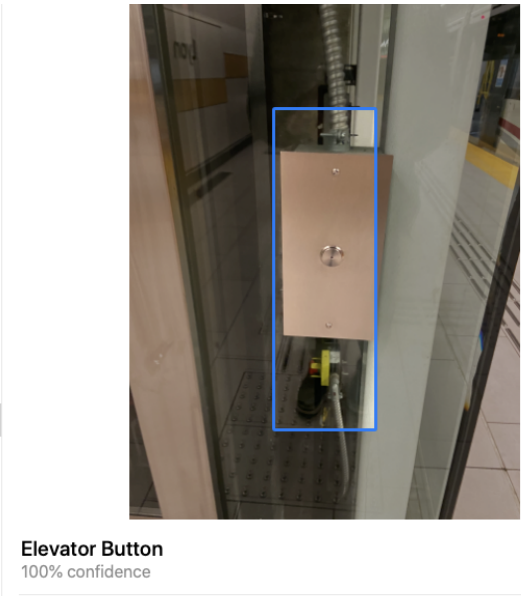


Figure 4. Functional Decomposition diagram

(a)



(b)

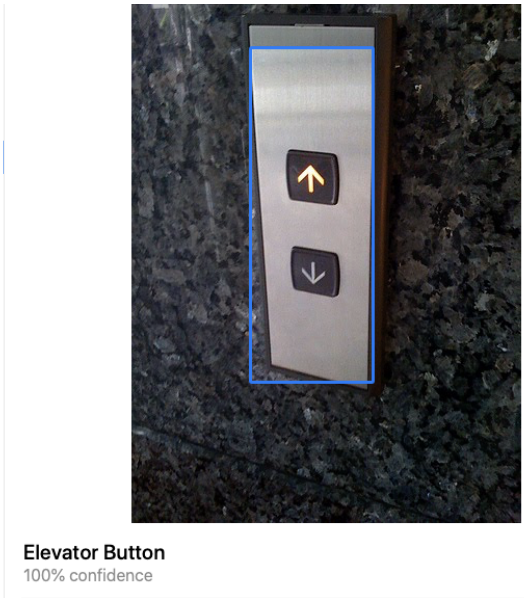


Figure 5. Prototype 2 (Tests of image recognition software) (a) Elevator button in Ottawa LRT (b) Generic elevator button