6-2022

An iOS Application for Visually Impaired Individuals to Assist with Crossing Roads

Ali Khan
Union College - Schenectady, NY

An iOS Application for Visually Impaired Individuals to Assist with Crossing Roads

Ali Ahmed Khan

Ece 499, Capstone Project

Advisor: Liu, Yan

Summary:

In day-to-day life, visually impaired individuals face the problem of crossing roads by themselves. This project was designed and built to solve this key issue. The system is supposed to give the user a warning before approaching a crosswalk for their safety and also give information about when it is safe to cross the road. An iOS application was developed to address the problem since recent studies have discovered that a vast number of visually impaired individuals are using smartphones (iPhones in particular) due to the ease and convenience it brings to their daily life. The application should be able to: notify the user through audio and haptic vibration, inform the user about crossing with 100% confidence with audio, all the input should be voice-based. In addition, it shouldn't need any internet, shouldn't consume much battery power, and run in the background with other applications. The application goal is to be used in Queens, NY in particular. For warning notification, we used iBeacon to know if a user is approaching a crosswalk. Currently, the iBeacon detection has only an error distance of +/-3 feet. In our implementation, every output of the application is audio and haptic vibration, and the inputs are all voice and haptic touch which are standard for the visually impaired. The application informs the user when it is safe to cross by detecting traffic lights 30 frames per second using the phone's built-in camera and an offline image processing model. The model was built in Caffe2 and then converted to CoreML for this project. The model alone has an accuracy of 92.9% and on top of that, we added cross-checking which increases the confidence level according to Bayes Theorem. Also, there is an additional layer of state machine which refines the output using possible timing of the traffic light to give 100% confident signal to cross the road and makes sure we don't make the user wait more than 10 seconds without any decision. The application is only 30MB in size, doesn't need any internet connection, and has low to average power consumption. The application can also run in the background allowing low power consumption and the capability to work with other applications. We have done both unit testing and integration testing to confirm every component of the application. The only thing that is left is to test the application in the real world by visually impaired individuals to confirm this initial prototype.

Table of Contents:

List of Figures, Tables and Code Snippets:

Introduction:

For orientation and mobility (O&M), one of the common problems faced by visually impaired individuals is crossing roads. It hinders their ability to be independent even though most of the cities and suburban areas have made different accommodations for O&M. Recent studies have discovered that a vast number of visually impaired individuals are using smartphones due to the ease and convenience smartphone brings. An iOS mobile application is proposed to accommodate visually impaired individuals to crossroads which will be low cost and implementable with current traffic systems in Queens Blvd, NY.

This report will go into detail for requirements for the project, design decisions are taken so far, design, and scheduling for the project.

Background:

In our fast and growing tech world, we often forget some of our members of the community. We develop technologies surrounding only majorities and thus the world gets narrower for others.

One minor group that is often forgotten about is the visually impaired community. Moving around day-to-day is a bit difficult for them as the systems are designed with vision capability in consideration. For example, crosswalks have lights showing if it is safe to cross or not. One may argue that having a sound continuously going on is noise pollution, thus the engineers didn't implement anything that could help vision-impaired people. This becomes a challenge for the individual to have an independent flow of life.

The talk about the inclusion of vision-impaired individuals has been an ongoing topic for a while, almost 50 years. It is mentioned in engineering ethics and practices to include individuals such as the visually impaired. For new technologies related to orientation and movement (O&M), engineers are trying to add more additional features like tactile paving.[1] Tactile Paving is one strategy that is used on the sidewalks to allow the visually impaired to stay on the sidewalk and let them know when to turn. It also has a different end of the sidewalk pattern to let the person know about the main street starting. Everything is good and all but there is the key missing part for crossing the road.[2] Currently, there are very few solutions that have been implemented to address this problem. One of them is a crosswalk button but that is not seen everywhere and for the visually impaired, locating the button is a challenge by itself.

Many research-related to orientation and mobility (O&M) for the visually impaired have been done. One research focused on making a tracking system indoor. To track the individual's location and inform them about it, they use a phone camera to receive and give feedback to the user. The camera's purpose is to scan images near the individual. But these large tags have two issues; one, the camera doesn't pick an image continuously and precisely, second, the room doesn't seem regular anymore.[3]

Another research by M. Poggi was done to develop a system that can detect objects in the path. The system consists of a wearable device that has cameras that sends the data to a portable computer (can also work with a phone). The camera creates a 3D vision and the computer uses deep learning to determine and gives the user feedback through headphones about the object in front. They also added a color sensor on the cane to give more information to the user about the object's color. This system is really a great idea, the only problem is wearing a different kind of headset that has the cameras and also carrying a separate battery for it.[4]

One research and implementation that was done in Panama to support the mobility of visually impaired people relates to my project definition pretty well. They created and implemented a system for Panama public transportation where vision-impaired individuals can independently locate bus stops and also know the arrival of the bus. The system works with radio-frequency (RF) modules which are cheap in cost and implemented at bus stops. This module sends radio signals that an android application receives and gives the user audio feedback. This is a very normalized model since the only thing an individual would need is a smartphone which most visually impaired use nowadays.[5]

Specifications:

The application will be multifunctional with different requirements. The requirements can be divided into three following subsystems:

Notification System for approaching a crosswalk:

1. Need to notify the user before approaching a street corner/cross-section.
2. The notification should be triggered within a 5 meter radius of the corner.
3. The notification should be audio based and may have haptic vibration.
4. Through voice command, the notification should open the application.

Detecting Traffic Light:

1. Need to identify a 12 inch 3 light traffic light system standardized by the Department of transportation for Queens BLVD.[6]
2. The application will identify the traffic within a 65 to 90degree angle from the user.
3. Should be able to determine the traffic light states between RED, GREEN, and YELLOW.
4. Detection time of the traffic light or the state of traffic light should be within 60 sec. If it ellipses, then timeout instruction should trigger. The timeout should notify the user that the application can not give direction for the traffic light.
5. The system should only notify the user when the traffic light turns Yellow to Red. As there is a possibility of Red light turning into Green directly while the user tries to cross the road. When the light turns from yellow to red, it would allow enough time for the user to cross.
6. If there is any ambiguity, the system should not give any output till time out.
7. The accuracy of the output that notifies the user to cross the road needs to be 100% accurate.
8. The application needs to work in iOS 9 and later. Which is after iPhone 6s and 6 Plus.[7]

Application:

1. The application needs to run as a background application when the user uses some sort of map application i.e google maps.
2. The application should not discharge more than 15% of the battery while running as a background application for 24 hours.[8]
3. The application UI should be visually impaired friendly where inputs will be through voice commands and output should be audio, and haptic responses.
4. The application should work without a network/internet connection.

Constraints:

1. The installation and system cost shouldn’t be more than $2 approximately. The installation cost was fixed to $2 as the estimated time for installation is 8min and the hourly wage is $15.[9]

2. The system needs to be capable of running on solar energy and operating within 3.3V. Total power consumption shouldn't be more than 1W. This power and voltage value was decided from single solar cell energy production.[10]
3. The application will only locate the traffic light from a corner of the traffic light. As below figure 1, the user needs to stand in the red circle corner to operate the app and identify the traffic light.[6]

Fig 1: Traffic light from Briarwood, Queens NY. Indicating a street corner where the application will be able to detect the traffic light. (Google Maps Street View)

4. Due to time constraints, the only traffic light dimension it might work is defined in Queens Blvd standard Drawing SE-002. The traffic light pole height needs to be 20'-2" with a hang of 20"-0. The cable connection should be within 13" to 13 ¼ ". The dimension is as

follows:

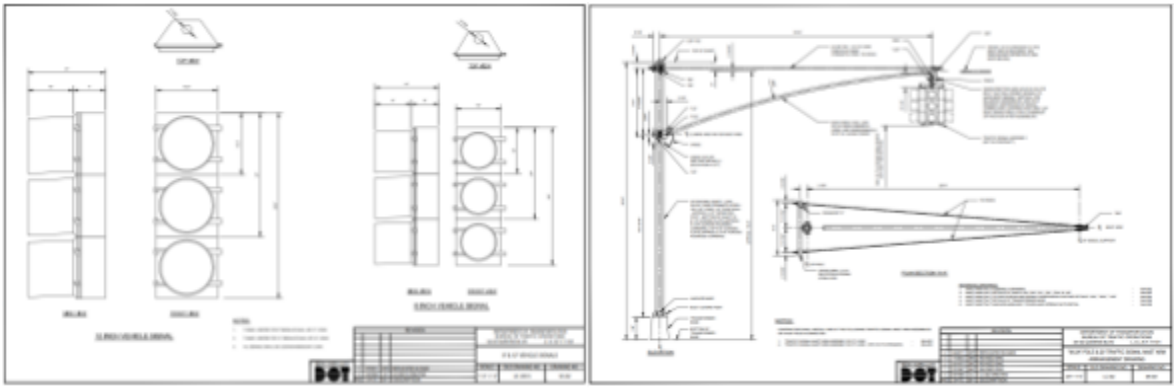

Fig 2: (Left) 12 Inch Vehicle Signal. Standard traffic light dimension for Queens Blvd. (Right) Elevation and extension of traffic light for Queens Blvd. Drawing MA:001.[6]

5. The system may only work in a cross-section that is close to a rectangle mentioned in DelDOT Road Design Manual.[11]

Fig 3: Example of a rectangular-quadrilateral cross section. Queens Library, New York City.(Google map)

6. It is limited to 3 light traffic systems which have solid red, green, and yellow lights. So it won't be able to detect arrow signals or bike signals.

Fig 4:Example of Bicycle Traffic Signal

Fig 5:Example of directional traffic light

7. The system will not be able to detect crossing light due to the limitation of training data or trained model.

Fig 6: Crosswalk light that the system cannot detect due to limited image data.

8. The system may not be able to detect traffic lights in heavy rain or snowstorms due to the regular visibility problem. So more clear weather is preferred for the camera to detect.

Fig 7: Snowstorm (left) and rain storm (right) example where the application may not detect the traffic light.

9. The application size should not be more than 100MB over the air according to Apple's App Store Standards.[12]

Standards:

To accomplish this project, the following standards were followed:

Design Pattern (Bridge Pattern):

The bridge pattern falls under behavioral design patterns. In this process the client code is decoupled using abstraction like interfaces and inheritances are created to have several possible implementations. The key goal of the bridge pattern is to relate the classes and objects. The bridge pattern does it by separating the abstraction and the implementation in separate class hierarchies. Through composition, it connects the hierarchies.

The design pattern helps to manage code at a low level, making code more flexible and less fragile to changes. With this pattern, doing unit tests is easier as it doesn't require superclass and each system can run independently also.[13]

Test-Driven Development:

Test-driven development is a concept where all production code is written in response to a test case. It is rooted in extreme programming where code must satisfy certain requirements and behaviors. The aim of this concept is not to pass the optimum solution in the first pass, but rather in an iterative manner to pass one case at a time to solve the main problem.

It follows three laws:

- Not allowed to write any production code unless it is to make a failing unit test pass.

- Not allowed to write any more of a unit test than is sufficient to fail, and compilation failures are failures.
- Not allowed to write any more production code than is sufficient to pass the one failing unit test.

This concept improves code quality, faster and continuous delivery, and adding new features is easier.[14]

Agile Methodology:

One version of Agile methodology known as Simplified Agile Methodology for Ontology Development (SAMOD) works well with Test Driven Development for faster product delivery. To understand SAMOD, some terminology needs to be familiarized with:

- A test case T_n, produced in the n^{th} iteration of the process, is a sextuple.
- A motivating scenario (MS) is a small story problem that provides a short description and a set of informal and intuitive examples about it.
- An informal competency question (CQ) is a natural language question that represents an informal requirement within a particular domain.
- A glossary of terms (GoT) is a list of term-definition pairs related to terms that are commonly used for talking about the domain in consideration.
- TBox is a formal model written in a proper language, implementing the description introduced in the motivating scenario.[15]

The SAMOD follows three crucial steps as follows:

1. Define a New Test Case:

   First, the motivating scenario needs to be written down with the input MS_n. From the motivation scenario competency question (CQ_n) is extracted. From the MS_n and CQ_n the glossary of terms (GoT_n) are extracted. From the GoT_n, the test cases are created. The test cases are kept small and simple, use patterns, and are self-explanatory. Within this step, a test-driven development process could be implemented.

2. Merge the Current Model With the Modelet:

   In this step, the merge of $modelete_n$, included in the new test case T_n with the current model, is done. To accomplish this step the following steps needs to be followed:

   a. Need to add all axioms in the current model and modelet to the new $TBox_n$. Then collapse all the semantically identical entities.

b. To update all the test cases in BoT, swapping the TBox of each test case with $TBox_n$ and refactoring each ABox and SQ according to the new entity names if needed, so as to refer to the more recent model.
c. To run the model test, the data test, and the query test on all the test cases in BoT, according to their formal requirements only.
d. Set $TBox_n$ as the new current model.

3. Refactor the Current Model:

   In this step, the current merged product is refactored. In the refactoring process the following principles are followed:

   - Reuse existing knowledge: It is a good practice to reuse concepts and relations defined in other models. It allows to align other models or include external entities.
   - Document it: The code should be annotated with labels, comments, and source link for outside entities.

By following these steps, an iteration of the product is achieved which is viable and more iteration with adding new features is possible without interrupting the production product. Overall the production speed increases and there is always some form of product always available that guarantees delivery. In addition, adding on to the project becomes very easy.[16]

Swift Design Guidelines:

For making the code more readable, and manageable, swift has certain conventions. Following the convention allows other developers to understand the code easily and using outside entities becomes easier. Also, following these conventions allows built-in debuggers and app testers to give better outputs. Here are some of the conventions that will be strictly followed:

- Naming:
  - Include all the words needed to avoid ambiguity for a person reading code where the name is used.
  - Omit needless words. If any word does describe more about the functionality or characteristics then don't include it.
  - Name variables, parameters, and associated types according to their roles, rather than their type constraints.
  - Clarify weakly typed information.

    - Name functions and methods according to their side-effects.
    - Uses of Boolean methods and properties should be read as assertions about the receiver when the use is non-mutating.
    - Avoid abbreviations.
    - Prefer method and function names that make use of sites form grammatical English phrases.
- General Convention:
    - Document the complexity of any computed property that is not O(1). This is important to make sure so the app doesn't take a lot of processing time without knowing.
    - Methods can share a base name when they share the same basic meaning or when they operate in distinct domains.
    - Follow case conventions. Names of types and protocols are UpperCamelCase. Everything else is lowerCamelCase.
- Argument and Parameters:
    - Choose parameter names to serve documentation.
    - Take advantage of defaulted parameters when it simplifies common uses.
    - Omit all labels when arguments can't be usefully distinguished.
- Additional:
    - Label tuple members and name closure parameters.
    - Take extra care with unconstrained polymorphism to avoid ambiguities in overload sets.[17]

Apple's Accessibility Standards:

Apple's accessibility standards cover a wide range of users with the need of vision, hearing, mobility, and cognition. The vision human interface guideline allows creating an accessible application for blind and visually impaired individuals. Some standards and practices that were considered are:

1. Simplifying the interface which enables familiarity, keeps the interaction consistent, and complex tasks simpler and straightforward.
2. Making sure that all content can be perceived.

3. Support the system-defined haptics. Some individuals require haptic responses only, Apple has defined certain haptic protocols that allow certain users to know the notification meaning.
4. Let people input information by speaking instead of typing.
5. Support Siri or Siri Shortcuts for performing important tasks by voice alone.
6. Make sure VoiceOver users can navigate to every element.
7. Notify VoiceOver when on-screen content or layout changes.
8. Use closed captions which gives people a textual equivalent for the audible information in a video.[18]

Design Alternatives:

From the design specification, the following major systems in figure 8 were extracted. The main system is divided into two major components of System Platform and Application. The Application's design depends on the System Platform, so for making any design decision for application components, first the Platform was decided. Under the Application component, there are four independent sub-components and each of them is connected with input-output relation. The sub-components have a particular goal and could be used as an independent project also. Each component for the system was chosen in the order listed below.

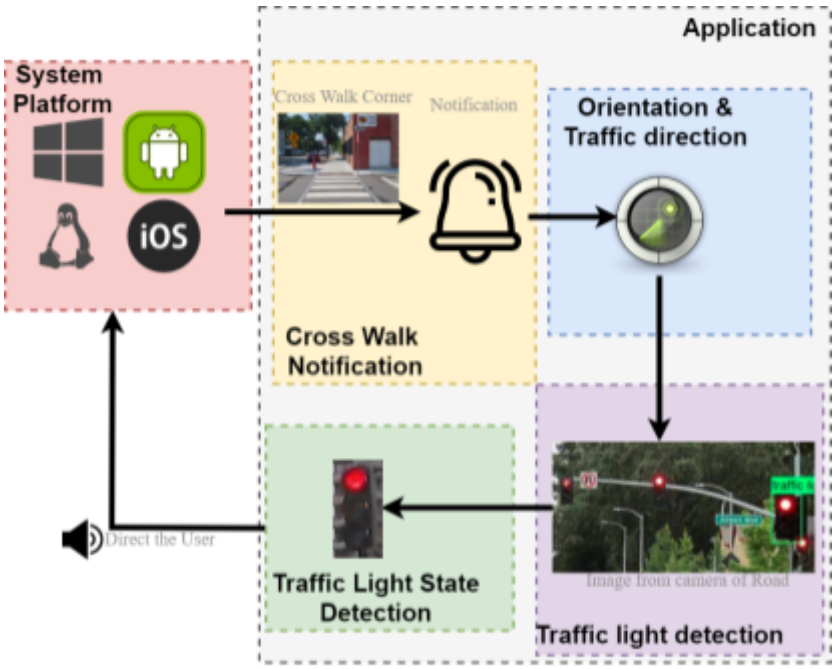

Fig 8: Initial Top Level Design

System Platform:

The system platform is the device that the whole project will be built on and the operating system is also part of it. First, we had to decide if we were going for a mobile or Unix platform. Both of them have advantages and disadvantages in many criteria, but since our focus was for

visually impaired individuals, the decision table criteria were shortened. When we thought about the Unix system, we were considering single board computers like nano jetson and raspberry pi. For mobile OS, we considered between Android and iOS since they are the most popular ones. Here is the decision table that was used to select the Operating system and the Platform. Note: Higher value means that the particular component does well in that certain criteria.

Platform	**Devices (OS)**	**Criteria**					
		Portability	Additional Devices	Image Processing	Available To Users (Visually impaired)	Built in solutions for visually impaired	OS Total
Unix (Single Board Computer)	**Raspberry PI** (Debian OS)	4/10	1/10	8/10	0/10	0/10	13
	Jetson Nano (Linux Tegra)	3/10	1/10	10/10	0/10	0/10	14
Mobile	**iPhone** (iOS)	9/10	9/10	6/10	8/10	9/10	41
	Android	9/10	9/10	7/10	7/10	7/10	39

Table 1: Decision Matrix for Platform and OS selection.[19,20,21]

First, let us see the difference in each platform then we will compare individual platforms to understand the reason behind each score above. We will only show the difference where they are much different.

Raspberry Pi vs Jetson Nano:

Criteria	Raspberry Pi[20]	Jetson Nano[19]
Portability	Power consumption is 3.8-4 W. **Thus this requires a smaller battery.**	Power consumption is 5-10W.
Image Processing	Uses only internal max 8GB DDR RAM	Has a 128-core Maxwell GPU specific for Image processing. **Best portable platform for image processing**

Table 2: Comparison between Raspberry Pi and Jetson Nano.

From table 2, it is clear that in the Unix platform, Jetson Nano is the best choice.

iOS vs Android:

Criteria	iOS	Android
Image Processing	Doesn't allow open source modules to run i.e OpenCV was discontinued. Prefers their own built-in modules.	Allows third party modules like OpenCV.[22]
Availability to Users	Most visually impaired in the USA uses iPhones for reading purposes.[21,23]	In the USA, the number of Visually impaired are less.[21]
Built-in solutions for visually impaired	● Apple created a whole Accessibility module in which developers can transform any iOS application into visually impaired friendly.[18] ● Setting up the accessibility option for users is easier.[21]	● There are modules for accessibility in Android but need different packages to add to the project. ● Takes more steps for users to set up the accessibility option.[21]

Table 3: Comparison between iOS and Android.[22,21,18,23]

Due to iOS being so accessible for the visually impaired, in mobile platforms, this is the best choice.

Comparing Unix and Mobile platforms, mobile platforms simply win due to two reasons. One it is compact and no external battery or camera is required, and secondly, it is available to most visually impaired people. Surely mobile applications have limitations of image processing

libraries for a developer, but from the user's perspective, it makes a great difference. Since our users are very sensitive to work with, we have kept their experience first in mind.

Crosswalk Notification:

This system is required for warning the user if they are coming close to a crosswalk for their safety. To notify them, we have to find their exact location, precise location may not even work as it's a safety reason. For finding the location we looked into GPS protocol and iBeacon protocol since they are most commonly used for this purpose. Here is the difference:

Criteria	GPS	iBeacon
Additional Device	No need.	Requires installation of iBeacon Bluetooth low energy modules
Phone Battery Discharge	Consumes 4 Watt power[25]	For 24hr, only consume 1.8-6.6%[24]
Precision	Uses approximation from nearby signals.	Get the exact distance through Bluetooth 4.0 protocol.[24]

Table 4: Comparison between GPS and iBeacon.

Since iBeacon gives us a more precise location, for the safety of the user we decided that iBeacon will be the best option.

Orientation and Traffic Detection:

This system was designed for visually impaired individuals to locate which side the traffic light would be. After making the decision about the iBeacon system, it was found that using iBeacon and gyroscope sensor within the phone, we can easily locate the direction of the traffic light. Thus no alternative design was chosen. If the current design doesn't work, then simply iBeacon can send the direction to the user about the traffic position.

Traffic Light Detection:

For detecting the traffic light through mobile cameras, three widely used machine learning frameworks for mobile: Caffe2, Pytorch Mobile, and Tensorflow Lite. We had already omitted any frameworks that require cloud services since we are making sure that the user doesn't require an internet connection. In addition, using a cloud system will add an extra delay to the system and in our real-time system, that is significant.

From the research done by Chunjie Luo, Xiwen He, and other researchers on "Comparison and Benchmarking of AI Models and Frameworks on Mobile Devices", it was

easier to differentiate each framework. Here is one of the decision matrices they created which helped to decide for our design:

Framework	Model	Source	Model File Size (MB)
Tensorflow Lite	ResNet50	Converted from Keras Application	98
	InceptionV3	Tensorflow lite model zoo	91
	DenseNet121	Converted from Keras Application	31
	SqueezeNet	Tensorflow lite model zoo	4.8
	MobileNetV2	Tensorflow lite model zoo	14
	MnasNet	Tensorflow lite model zoo	17
Caffe2	ResNet50	Caffe2 model zoo	123
	InceptionV3	Converted from Caffe model	115
	DenseNet121	Caffe2 model zoo	39
	SqueezeNet	Caffe2 model zoo	5.9
	MobileNetV2	Caffe2 model zoo	17
	MnasNet	Converted from Pytorch Torchvision	22
Pytorch Mobile	ResNet50	Converted from Pytorch Torchvision	98
	InceptionV3	Converted from Pytorch Torchvision	105
	DenseNet121	Converted from Pytorch Torchvision	32
	SqueezeNet	Converted from Pytorch Torchvision	4.9
	MobileNetV2	Converted from Pytorch Torchvision	14
	MnasNet	Converted from Pytorch Torchvision	18

Table 5: The source and model file size of the different model implementations.[26]

If we observe table 5, we can realize that TensorFlow lite models take less memory space, and taking less memory space means the ram processing speed will be faster for Tensorflow lite. Caffe2 framework already fails with a good difference compared to others in terms of model size. Even though Tensorflow lite is more optimized the accuracy graph shows a different sight:

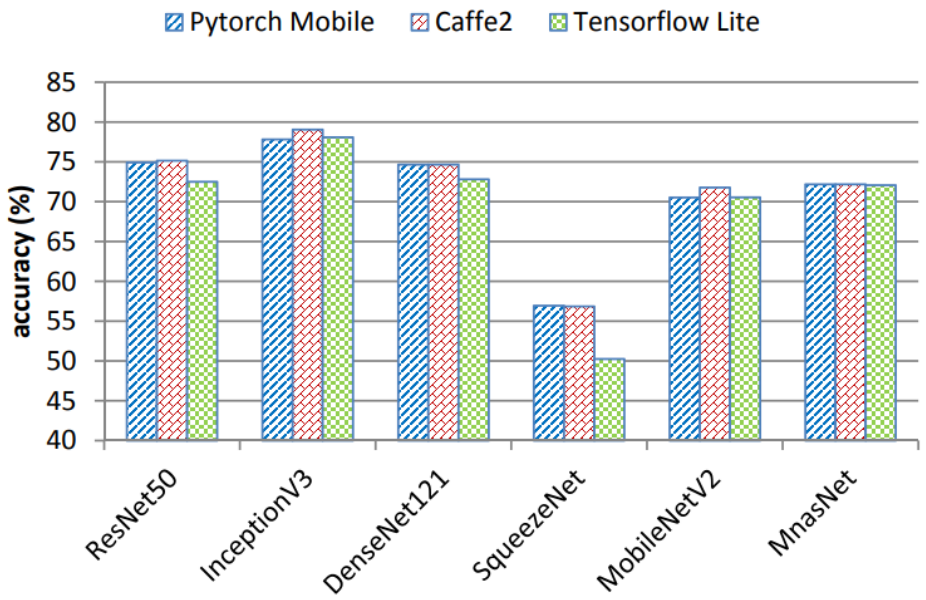

Fig 9: The accuracies (%) of different models with different frameworks. The results are the same on the devices of Galaxy s10e, Honor v20, Vivo x27, and Vivo nex. The results on Oppo R17 are slightly different.[26]

The accuracy graph shows a good competition between Pytorch mobile and Tensorflow lite. Although Caffe2 model wins among all of them. After looking into pre-trained models by Pytorch mobile and Tensorflow lite for the traffic light, we found that there is already an object detection model of traffic light for iOS. Since our project's main goal is not to train a model for traffic light detection, we planned to use Tensorflow Lite. Since our project will be modular, one can easily train a model with Pytorch mobile or Caffe2 and drop it into the project to use. If time permits, one of the additional goals is to look into other models.

Traffic Light State Detection:

The object detection model only gives us the position of the traffic light within a frame but doesn't tell us the state of the traffic light e.g red or green light. One strategy we used was to calculate the RGB in the cropped image from object detection. In our testing in figure 20, we found that it was not reliable so we had to use some form of vision processing. The three vision processing we looked into were CIColor, CIColor Kernel, and OpenCV. Here are the differences:

Criteria	CIColor[27]	OpenCV[22]	CIColor Kernel[28]
Customizability	It is a built-in module for doing image filters. It can only filter 3 colors.	Open-source library with many built-in and customizable modules.	Native bare skeleton for image processing built by apple.
Speed	Depends on RAM and processor only.	GPU process could be added.	Directly access the GPU of the phone to process images faster.
iOS Compatibility	Made by Apple for faster production and simple products.	Discontinued by Apple to promote CIKernel.	Apple is promoting the use of it for vision processing.

Table 6: Comparison between CIColor, CIColor Kernel, and OpenCV.

From Table 6 it was clear that the CIColor Kernel is a better choice since OpenCV discontinued and CIColor has customization limitations.

Preliminary Proposed Design:

After comparing different alternative designs for each component, the following top-level design was extracted:

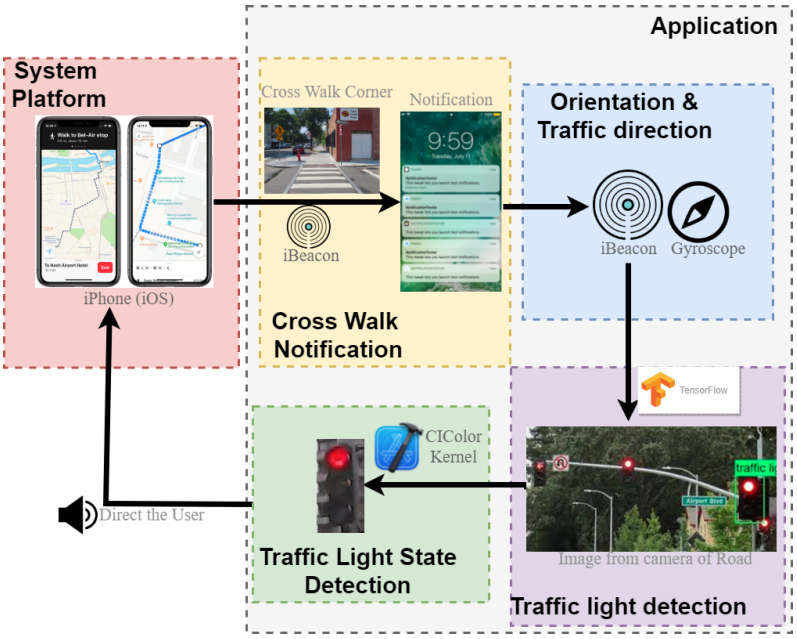

Figure 10: Preliminary top-level Design.

Each of the sub-systems is independent of each other but they are connected through input-output relationships. To understand how the design will satisfy the design specifications, we have to first understand the system flow.

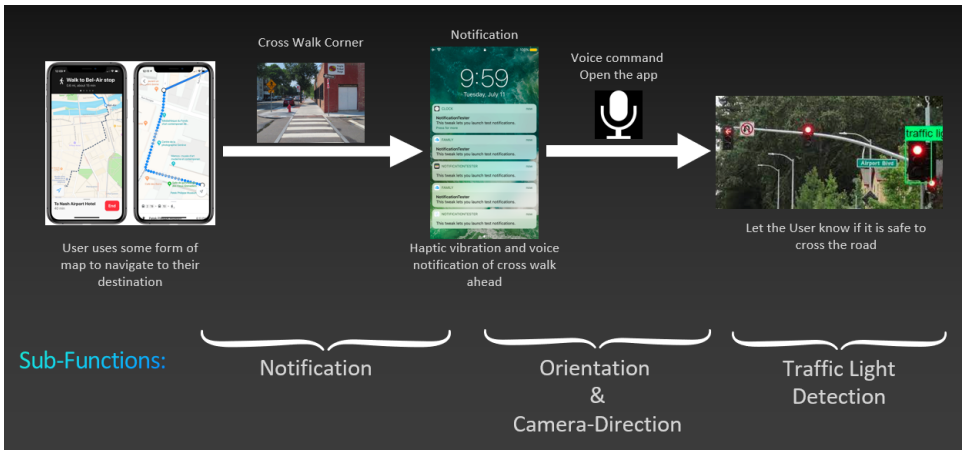

Figure 11: Application Flow Diagram

As in figure 11, users will be using some form of map application i.e Google Maps and the app will be running in the background. When the user comes close to a crossroad we will have a notification subroutine. The user can open the application with voice and the application camera will turn on. Within that step, the user will also get the traffic light position. From the camera, the user will get directions if they should cross the road or not. From this flow, we found three crucial subfunctions that need to be achieved:

1. Notification
2. Orientation & Camera Direction
3. Traffic Light Detection (merge of traffic light state & detection Component)

Notification:

As discussed before, Crosswalk notification is an important component of the design. The functionality of notifying the user would increase safety. This functionality requires to be running as a background application since the user needs map applications for getting direction and also running on background allows low battery usage.[24] Here is the flow chart for this function:

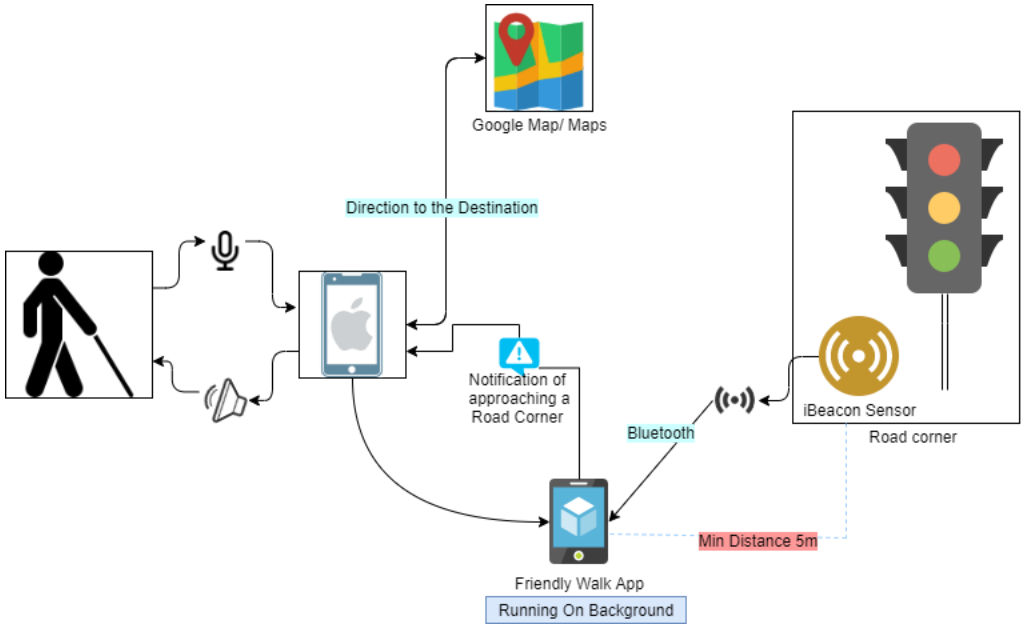

Figure 12: Notification sub-function flow chart.

From figure 12, we can see that our application runs in the background, and on top of it, a map application runs. When the user comes close to a traffic light pole, the phone will receive a Bluetooth signal. The iBeacon has the following property:

Field	Size	Description
UUID	16 bytes	Application developers should define a UUID specific to their app and deployment use case.
Major	2 bytes	Further specifies a specific iBeacon and use case. For example, this could define a sub-region within a larger region defined by the UUID.
Minor	2 bytes	Allows further subdivision of region or use case, specified by the application developer.

Table 7: iBeacon Advertising property.[19]

The Minor will contain the range of the mobile and the iBeacon device and the UUID will allow the user to know which traffic pole they are getting close towards. The ranging provides an approximation of the distance from the iBeacon transmitter using the information of the TX

Power field. The following equation is used for calculating the distance between the phone and iBeacon device[30]:

$$Loss = 32.44 + 10nlog(d) + 10nlog(f). \quad (1)$$

$$P_L(d) = P_L(d_0) + 10nlog(d/d_0) + X\sigma. \quad (2)$$

$$RSSI = Pt - P_L(d). \quad (3)$$

$$d = 10^{(A-RSSI)/10n} \quad (4)$$

In 1 equation above,$P_L$indicated the path loss of receiving sign when the measuring distance is $d(m)$, it indicated the absolute power value, and it is in dBm; $P_L(d_0)$ indicated the path loss of receiving sign when the reference distance is d_0; n indicated the path loss index in a specific environment; it indicated the speed of the path loss, which is increased along with increasing distance; Xσ is in dB; it is a cover factor when the range of standard deviation σ is 4~10 and the mean value is 0. In equation 2, P_t indicated the signal transmission power, $P_L(d)$ indicated the path loss when the distance is d, and they are both in dBm. In equation 3, $\overline{RSSI}$ is regarded as RSSI value of multiple times of measurement. And lastly, in equation 4, *d* is regarded as the undetermined distance.[30]

After we know the user is around 5 meters in distance from the sensor, we will notify them with a notification to the application. This response will be a haptic vibration and audio specified for warning in Apple standards.[#] Next the user can open the application simply saying "Open Friendly Walk". This will be pursued through Apple's library *AVSpeechSynthesizer* which has var *output channels: [AVAudioSessionChannelDescription]* to recognize the command.[18]

The direction of the traffic light is an important part of the system since it decreases the ambiguity for the user to locate the traffic light. Here is a basic flow chart for getting the direction:

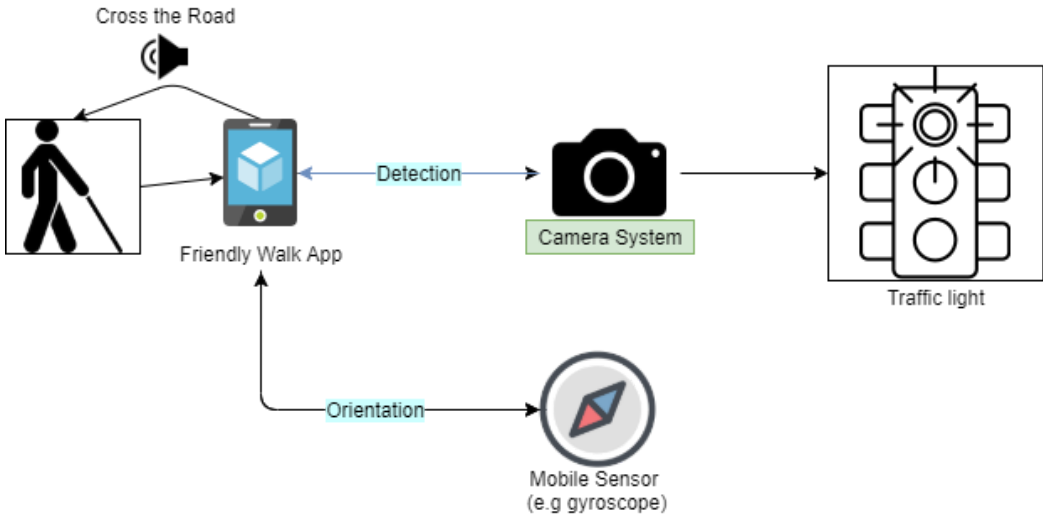

Figure 13: Flowchart for Traffic light direction.

As figure 13 shows, the application will access the gyroscope of the phone to know which direction the user is heading. By the heading direction, and distance from each iBeacon, we can create a basic layout of the user's position. The logic is shown in figure 14. By finding that D is the minimum distance and B, C is the second closest. We could confirm the position of the user. Distance A will just confirm the position. And having a direction from the gyroscope will let the user know if the traffic light is possibly on the left or right side of the user. Then users can simply point their camera to detect the traffic light.

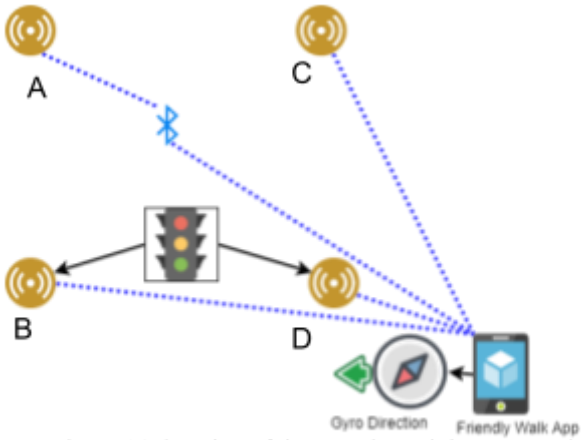

Figure 14: location of the user through iBeacon and mobile Gyroscope.

Traffic Light Detection:

The traffic light detection has two parts into the system, one for detecting the traffic light and cropping, and next returning the state of the traffic light. Here is the main flow diagram of the detection system:

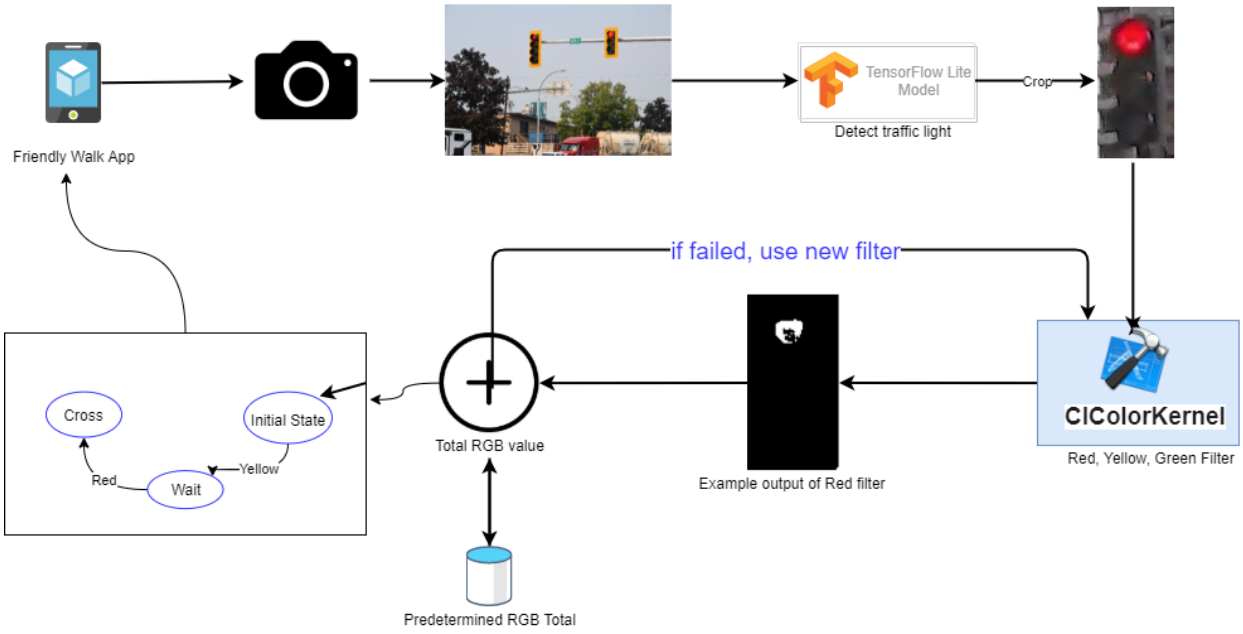

Figure 15: Flow Chart of object detection and image processing for traffic light.

So first the camera from the application takes real-time video and TensorFlow lite process 30 frames/s and detects the objects available in the image. From there we would extract the traffic light portion with their x,y coordinates. Here is a basic pseudo code:

```
class traffic_detection{
     func traffic_location(){
       object_image= tensorflowLite.model(image);
       for obj in object_image{
          if ("traffic_light"= obj.name){
                retrun (obj.x, obj.y);
          }
       return null;
       }
     }
}
```

Code 1: Pseudocode of determining the position of traffic light cropping

The above code gets the list of objects detected by TensorFlow lite and from there we try to find if the traffic light is in the objects list. One thing to note, we are treating TensorFlow lite as a black box since our work doesn't concern how TensorFlow lite is detecting the traffic light. But for basic understanding, we know it is a Recurrent Neural Network and from testing we found approximate accuracy to be around 63%.

From the cropped image, we have to find the state of the traffic light and for that, we are using masking of colors to output a black and white image. In masking colors, it takes a certain range of colors and compares them with each pixel color. If the pixel colors are not within the range then we change the color to black, else white. In figure 21, we can see a practical demonstration where a cropped red traffic light image went through red masking and the output only has white pixels at the red pixel.

From the masking, the simple idea is to calculate the average RGB value in images. The logic is simple:

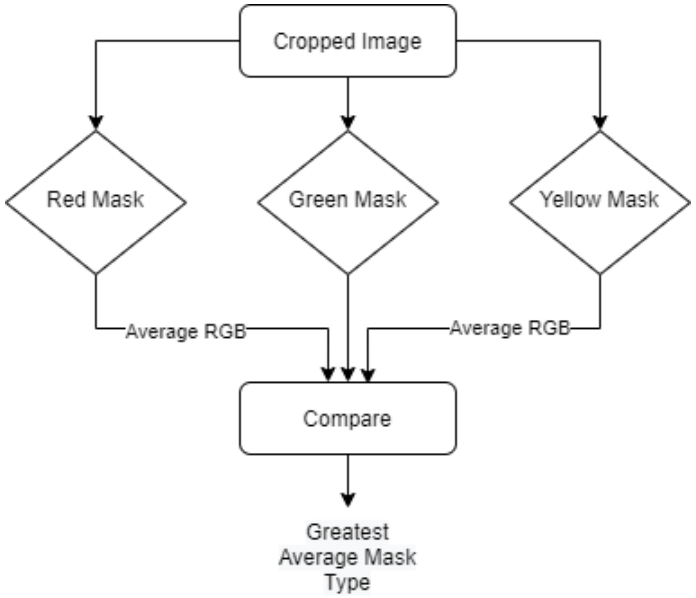

Figure 16: Decision flow of masking color.

As in figure 16, we calculate the average RGB in the image. Since white is the maximum RGB value, only correct masking will give the highest average. From there we would confirm the color of the traffic light.

The traffic light state is then sent to a state machine as an input that determines if the user should cross the road.

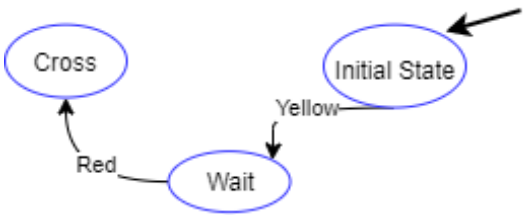

Figure 17: Decision state machine.

So the initial state is just awaiting phase where we wait for the camera to detect a traffic light. From the image processing we wait till the light turns yellow and we go to a waiting state. When the light changes from yellow to red, the system will go to the cross-state where we will tell the user to cross the road through the audio module described before.

That's how we will accomplish all the sub-functions mentioned above.

Preliminary Test Plan and Results:

For each design part we had created a design plan and also did some testing. Here are the tests depending on the design functionality:

Notification Testing:

For the notification, first we were going to create a simulation button within XCode which will allow us to simulate the connection between iBeacon and phone. From that simulation, we will see if our application receives and pops a notification.

The next key important thing is to check if the actual iBeacon gives us a distance. To understand if the measured distance is correct from the iBeacon, we wanted to match the following graph provided by the manufacturers:

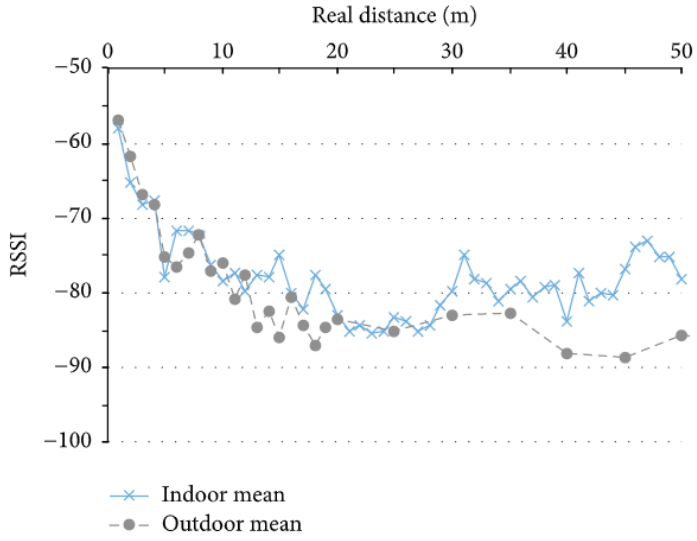

Figure 18: Distance vs RSSI value. [31]

From the comparison between our graph and figure 18, we can find the accuracy of the system.

Orientation Testing:

For orientation, we simply wanted to find the N-S of our testing room with a magnetic compass and match it with our measured gyroscope reading. For finding the accuracy, we might generate

multiple test cases where the phone will be in different orientations to check if the direction stays consistent.

Figure 19: Phone in a different orientation for testing direction.

Traffic Light Detection:

From the *Next* 2017 competition hosted by Nexar we were able to get 30,000 street images of NYC roads. We planned to run a validation test over the images where we will have sorted folders of red, green, and yellow traffic lights. For each folder, we would see if our object detection model can detect the traffic light. By seeing how many traffic lights are being detected, we can understand the accuracy of the model.[31]

The second test for traffic light detection is to test the traffic light state. For this, we found 1300 images of traffic lights which are already cropped and sorted. To test the accuracy we need to go through each color traffic light and see what our model predicts. If it gets the traffic light color correct then we give 1 as a score and for the wrong we give 0 as a score. We will add the scores for overall accuracy. From there we can find the accuracy of the model. In addition, if the image fails to get a score, we will save those in another folder to inspect the reasoning behind the failure.[32]

For integration testing, we were creating a pseudo traffic light system within the campus. There will be two traffic lights at a 90-degree angle for two different paths and the 4 corners will have iBeacon sensors to simulate the poles for traffic lights. Using that, we will do real-life testing where we will do 10 tries with 5 voluntary users with the application and see how accurate our application is from each result.

One thing to be noted, after integration, we can still do the unit testing that we discussed before to make sure none of the unit systems are broken. We had done testing on the traffic light state detection with and without masking. The first approach was to do it without any masking but we found that there are images where the traffic system is ambiguous. Here is the test result where we tested red, and green light with the filter and predict the color of the light:

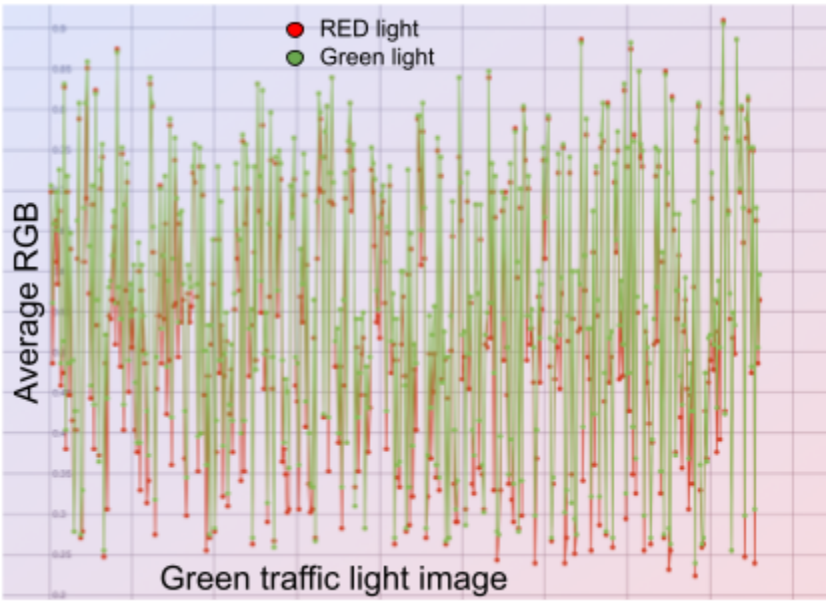

Fig 20: Prediction of green light from CIColor system. Higher the average RGB, the possibility of that light is higher.

In figure 20, the negative average shows that the output is less likely to be that color. But as you can observe, even though the images were for the green light, we had received some positive average for red light which was a wrong prediction. So we went to create color masking. We had created two masks, one for the green light and another for the red light. Here are the results:

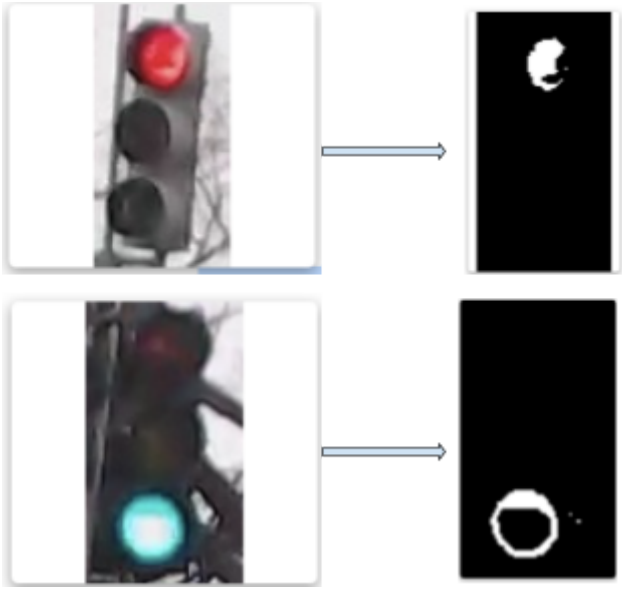

Figure 21: Testing of color masking system.

Final Design and Implementation:

The design has been altered just for the image detection subsystem due to iOS providing less access to lower-level data. As by the preliminary design, we were detecting the traffic light with the TensorFlow Lite model and then masking the image for different colors (e.g red). The masking was working as desired, shown in figure 21. From those masking, the plan was to extract the average RGB color from each pixel to determine which masking had the highest value and give a score according to them to evaluate the traffic state. But Apple has created their own image systems: UIImage and CoreImage which don't give access to lower-level information due to information hiding.[34] We tried to access the kernel going around OS-provided functions through C code which is provided in the appendix(AverageColor). Then for each red, green, and yellow image data set, we tested the accuracy of the RGB calculation and the prediction. We got the following results:

Traffic light color in the Image	Total Image	Number Correct Prediction	Predicted (# of Wrong)
Green	560	382	Red (72) Yellow (106)
Red	600	358	Green (87) Yellow (155)
Yellow	142	84	Red (23) Green (35)

Table 8: Prediction testing after correct masking and average RGB calculation.

After testing the system, it was clear that we couldn't rely on this accuracy so we went to our alternative list. Reviewing figure 9, it was clear that we should go with the Caffe2 framework but the issue is, Apple is not compatible with it. Apple has created its own machine learning framework called CoreML. So we took an approach to use the Caffe2 model by converting it to the CoreML model. One thing that needs to be noted is that Apple's M1 chip is not able to do this conversion and requires one to use the UNIX system so we used Linux (Ubuntu). Here is a python code snippet of conversion of Caffe2 model to CoreML:

```
import coremltools

caffe_model = ('train_squeezenet_scratch_trainval_manual_p2__iter_8000.caffemodel',
'deploy.prototxt')
labels = 'labels.txt'
coreml_model =
coremltools.converters.caffe.convert(caffe_model,class_labels=labels,
image_input_names='data')
coreml_model.save('TFM.mlmodel')
```

Code 2: Conversion of Caffe2 Model to CoreML Model.[35]

After the change, this the finalized top-level design:`

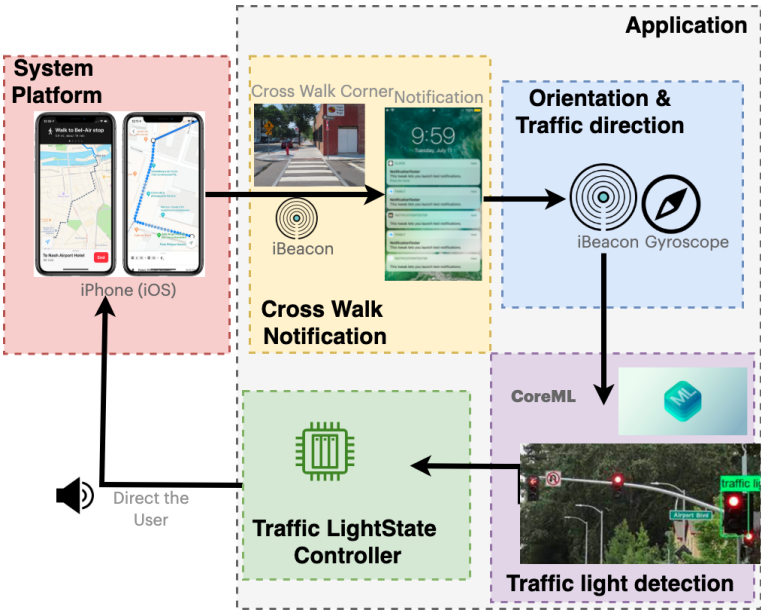

Figure 22: Finalized Top-Level Design.

So our application (figure 11) and subsystems (figure 12, 13, and 14) stayed the same. The only design change is how we detect the traffic light and its state. Both of those issues are simply solved by the CoreML model as we get direct results. This is the current traffic light detection flow chart:

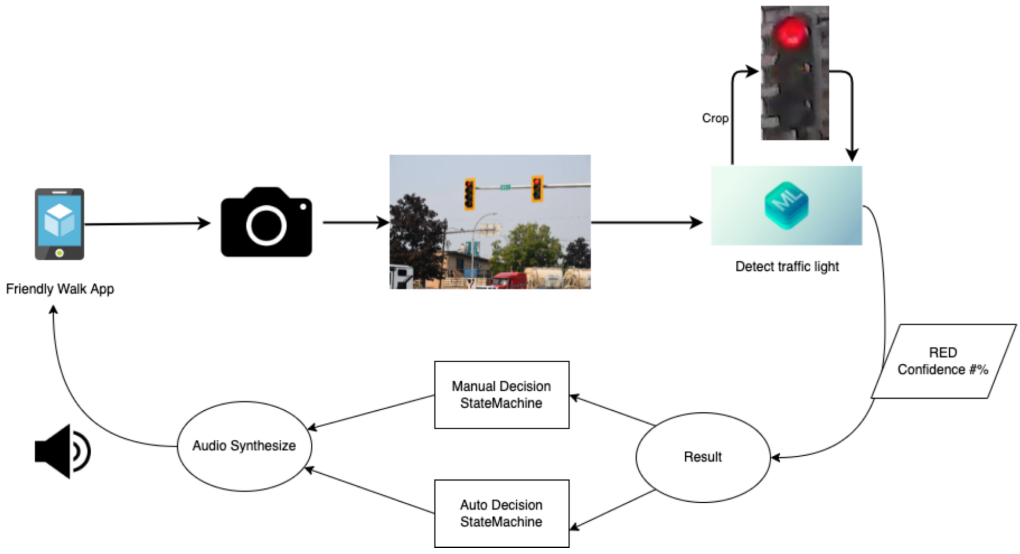

Figure 23: Finalized traffic light detection flow chart.

As shown in figure 23, we can detect the traffic light and the state of traffic light including the confidence level from the CoreML model.

After receiving the result from the model, the application is designed to have two options for users to choose from for getting the information. One is called Manual Decision and the other is Auto Decision. The auto decision system is for the application to make decisions on behalf of the user and notify the user when to wait or cross. This is done by the following state machine:

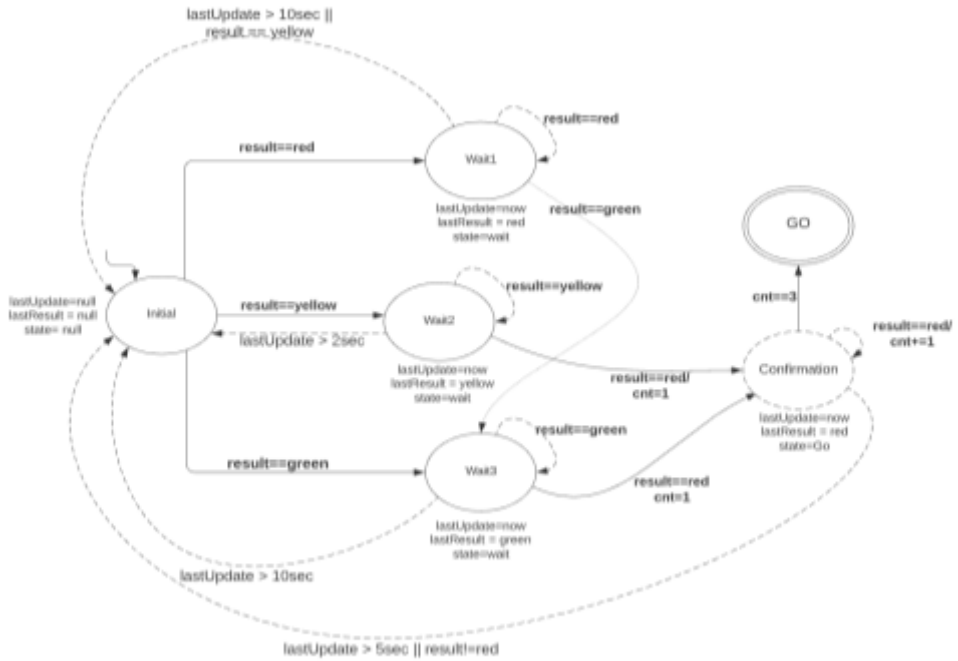

Figure 24: Auto Decision StateMachine.

The camera takes 30 frames per second and we analyze those frames through our model which is based on the YOLO model. Due to this fast and only analyzing ones, the chance of ambiguity in frames is higher and the rate of detection is not constant. In addition, there is a chance that the user's hands shake or move and we don't get consistent detection. The above state machine in figure 24 takes care of all those edge cases, including the timing delay. We also took into consideration different scenarios e.g when a user arrives at the corner of the street when there is a red light. In that case, there is still a chance for the light to turn green while crossing. So we kept those scenarios in mind and designed them so the system only gives notification of crossing when it is really safe.

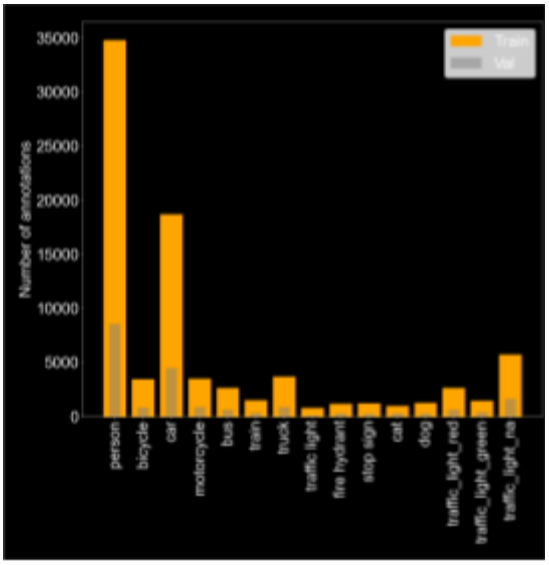

Figure 25: Data distribution of the trained model.[37]

While testing the model we realized that the dataset used for creating our model had the dataset from figure 25. It became apparent that we can not rely on yellow light detection since the dataset didn't consider the presence of yellow light. Not having yellow light detection is not a major problem, it just gives us a guarantee that the next light is red. So to work out that, we kept a filter for only red and green results. Then we simplified our state machine as follows:

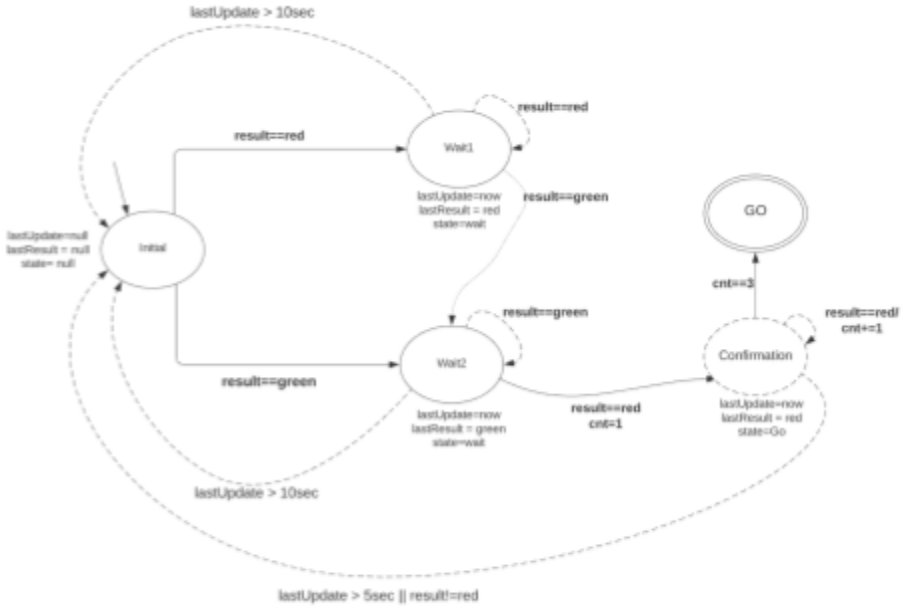

Figure 26: Auto Decision StateMachine without Yellow Traffic light detection.

Here is a small snippet of the implemented state machine described above:

```
func stateWiseUpdate(result: String){
            switch currentState{
            case .initial:
                if result == "red"{
                 ...
                }else if result == "green"{
                   ...}
            case .wait1:
                if result == "green"{
                  ...}
                else if result == "red"{
                   ...
               }
            case .wait2:
                if result == "red"{
                    ...
                }else if result == "green"{
                    ...
            case .confirmation:
                if (result == "red"){
                ...
            case .go:
                if confirmCnt < GO_REPEAT{
                    if(lastSpoke.distance(to: nowTime) >= SPEAK_UPDATE_TIME){
                        speak(toSay: "Go!")
                    }
                    break....
```

Code 3: Auto Decision StateMachine Implementation snippet.

The manual decision system directly just informs the state of the traffic light. But one thing needs to be noted, both of the systems make sure that we don't notify the user repeatedly which might bother the user. We repeat in a certain timeline and when we are confirmed that it could be repeated. If there is any new state of traffic light we also notify the user immediately. Figure 27 shows the state machine that takes care of all these decisions for notifying through speech.

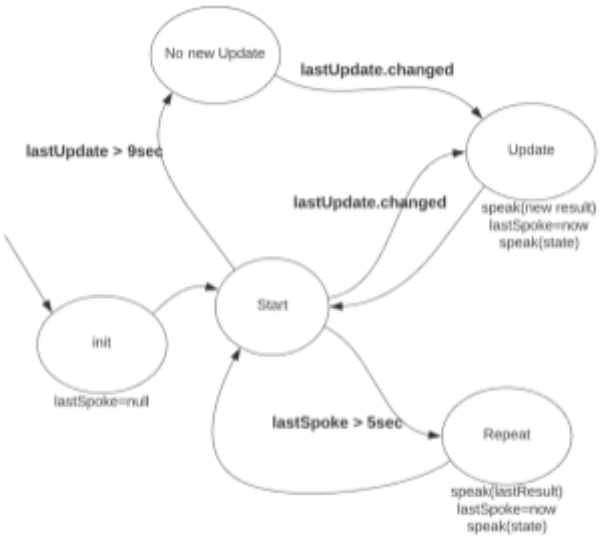

Fig 27: Manual Decision and Speech timing

For running the application on the background and also fetch location,

we had to use the background location mode which makes our application seen as a navigation application due to Apple's policy.[36] Thus, in addition, we had to ask users to give us permission to run in the background and also track their coordinates. Since we get the background running capability from the permission, it allows us to check if we are connected to the iBeacon or not. For robust testing of the notification system, we also used an iBeacon simulator on an iPad which transmitted the same iBeacon frequencies that we desired from the actual device. This was important for testing as we didn't want any fault from iBeacon to be miss interpreted for the notification system fault.

Figure 28: iBeacon Simulator. Capable of configuring same UUID and (Major, Minor) of the device.

To connect the iBeacon we used Apple's built-in library called CoreLocation which has a location manager module. This module took care of connecting and giving information about how far the iBeacon is as follows:

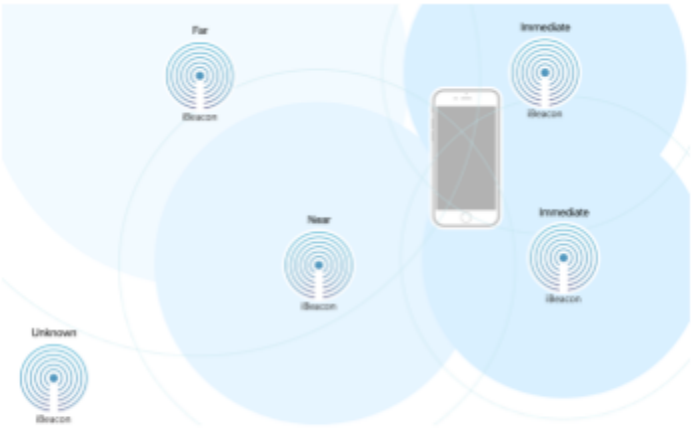

Figure 29: CoreLocation module output proximities for iBeacon

So when someone is within range it will be Far, Near, or Immediate but if it is out of range we would get Unknown. One thing needs to be noted, we didn't know the distance (in feet or meters) for each output e.g. within 1m is Immediate. So we had to test which also varied from device to device.

For the application, we also had created a UI flow diagram and for internal implementation, we used two design patterns: MVC and Delegation. Here is the top view of the UI flow Diagram:

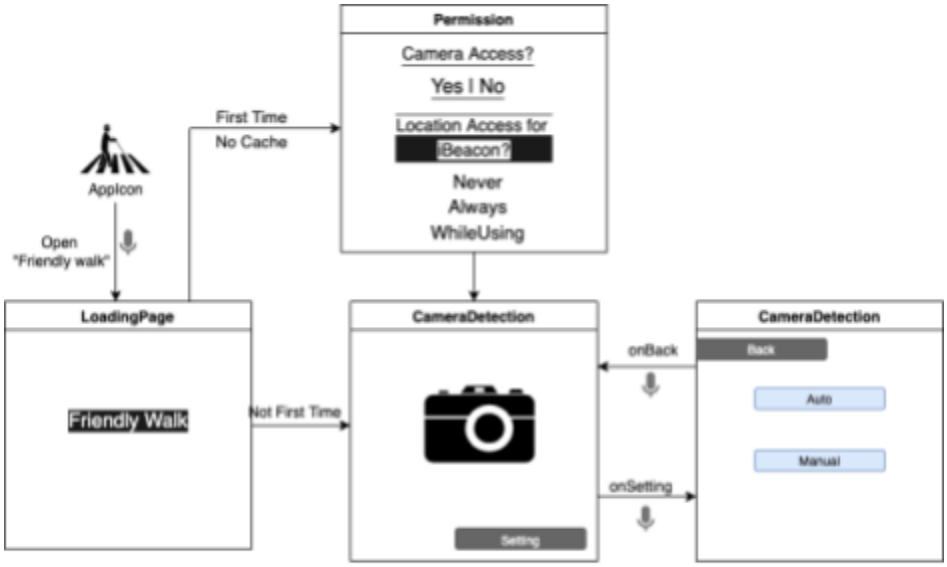

Figure 30: UI flow diagram of Friendly Walk

Here if the user doesn't accept any of the permission, then we error handle internally and just don't give any output to make sure the application doesn't crash and inform the user why there is no output.

Final Design Results and Design Evaluation:

This project was based on the test-driven application development standards so each subsystem was tested individually and also after integration.

Notifications Sub-system:

The notification subsystem consisted of iBeacon and local notification.

iBeacon:

First we implemented the simulator by turning on and off the iPad's iBeacon simulator to make sure that the system connects properly. As figure 31 shows, when the simulator was off we had a gray screen and when we turned it on, we got the red screen on the phone showing that the system connected as desired. (Note: The wire is just charging the phone.)

Figure 31: iPad turning iBeacon simulator on from off and Phone detecting it.

After that, we recreated the different proximities that are mentioned in figure 29 by moving the iPad at certain distances from the phone

Figure 32: Screenshots from the phone when the iPad was moved to different distances.

In figure 32, when the iPad was not simulating, we had the Unknown screen. After turning it on, since both iPhone and iPad were sitting side by side, we had a close reading in red corresponding to immediate in figure 29. As we moved away from the beacon, we got near value (yellow) and moving further gave us far (blue). When we almost exited the building, around 50 ft away from the iPad, we got unknown(gray) again as the phone couldn't detect the iPad.

Since this simulation worked, we did a similar test on our hardware iBeacon which we set up according to the manual of the manufacturer. When we tested it, we noticed that the detection was fluctuating between Close to Unknown periodically in a uniform pattern. We realized from the manual that the device has short sleep and output burst timing of every 250ms. This means, our application in the future, needs to incorporate this sleep depending on the manufacturer we select for mass implementation.

Figure 33: Detecting low-powered iBeacon module.

As we were able to read different proximities of near, far, etc. we wanted to find the reading distances. So we tested by moving away from the iBeacon at different distances and got the following results:

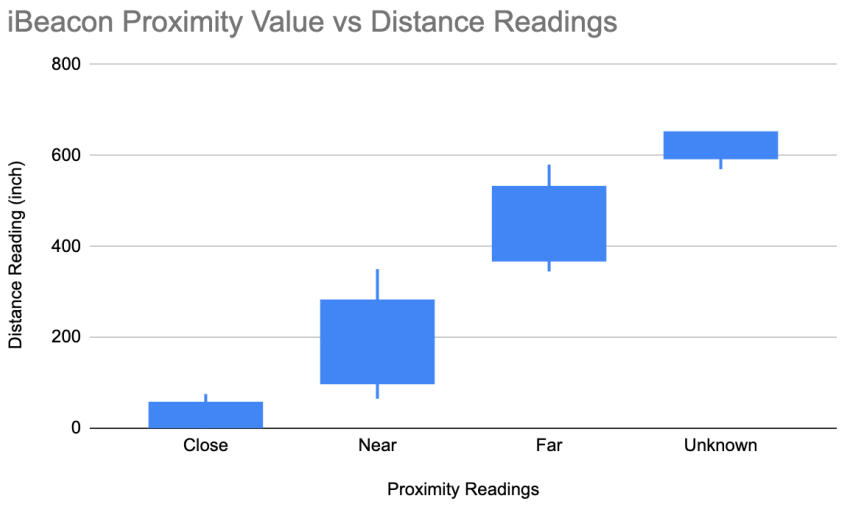

Figure 34: iBeacon Proximity in terms of distance.

As shown above, the distance is not always consistent. For example in one reading the Near could be read within 72 inches or 6ft and next it could be around 8ft. From those errors, we were able to get solid values where it is for sure Near. For Unknown, we didn't go further after a couple of feet from 50 ft since it would always give us Unknown. Thus, it doesn't have any upper bound. This result could be used to set at what distance we want the user to be notified that they are approaching a traffic light. Current implementation notifies the user when they are within the proximity of Close or Near.

Local Notification and Integration:

Since our application doesn't need any internet connection, we kept the iBeacon detection notification local. For enabling this voice notification, first, we always ask the user for permissions for both location and notification (figure 36). Asking for the location to be always on was necessary for the iBeacon to be detected in the background. Figure 35 shows how we detected a near iBeacon in the background. We were also able to retrieve the longitude and latitude location (42.8186., -73.9331.) from this detection.

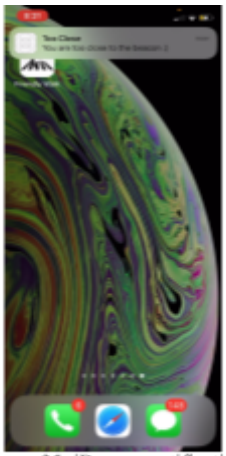

Figure 35: iBeacon notification after detecting from the background.

Due to time constraints, we were not able to integrate multiple iBeacon which resulted in the Orientation system not being implemented. This means we can not tell the user which side is the traffic light and how to adjust their hand to have the camera directly point at the traffic light.

Application Flow and Performance:

To understand further integration and development, first we need to understand how the application is working. As we designed the UI flow in figure 30, we implemented the flow exactly. When the user says “Open Friendly Walk” in the home menu of their phone, the app opens up. The loading section can be found in the appendix figure 42. If it is the user’s first time, we ask for some permissions and access.

Permission:

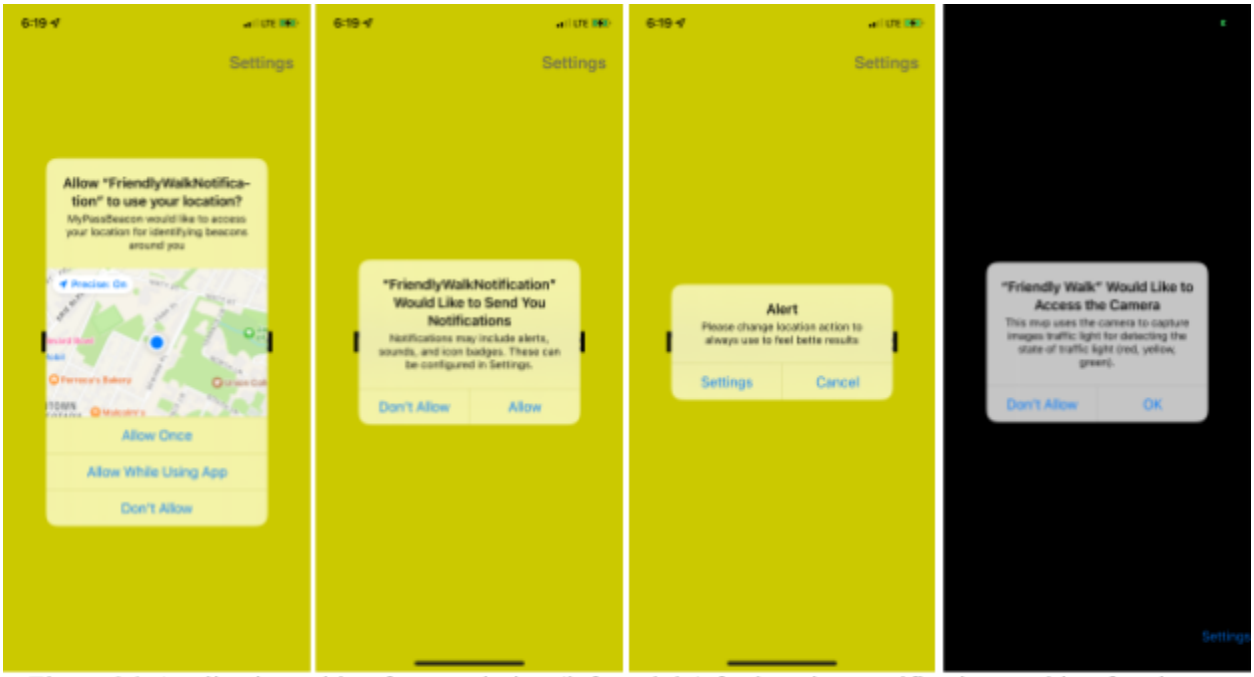

Figure 36: Application asking for permission (left to right) for location, notifications, asking for always location access for better result, and lastly getting camera access permission.

The above figure shows how to implement the permission system to ask the user for access to the location and camera. We also asked for permission to give notifications. For now, we set a system that if we don’t get any of the permissions e.g. location, we don’t give any results from any of the subsystems since everything is intertwined.

Settings:

Now on the main page, which includes the camera, there is a transparent settings button covering the main view. Upon the page loading, the application says “Settings”. When a user

says “Tap Settings” or “Click Settings”, it goes to the setting option. Upon arriving at the setting option, the user hears: “Decision settings, A menu to choose the type of decision”. Then we have the two options of automatic or manual which gives detail before clicking. The automatic button reads out: “Decides and informs you automatically when to cross or wait. Say Tap auto”, and the manual button reads out: “Just tells you the state of the traffic light and you make the decision. Say Tap manual”. After they say either of the commands, the user will hear what settings have been selected as confirmation. To get out of the setting option and return to the camera, the user needs to say “Go back” or “tap back”.

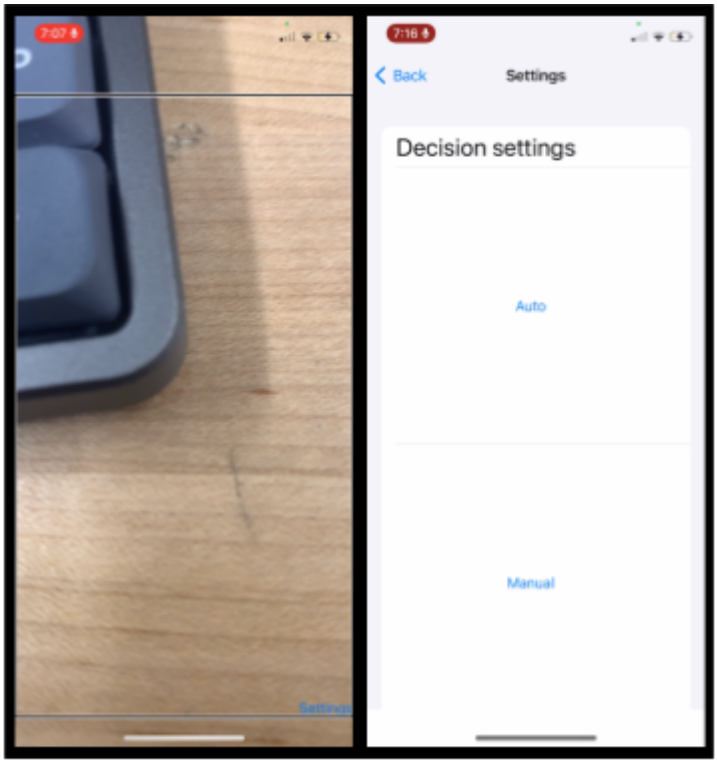

Figure 37: (Left) The main page with border setting button. (Right) The setting page for different setting options.

The application is also very optimized for both power and memory consumption. The application drains at low to average battery power most of the time. Also, the application's total size is only 30MB only.

Figure 38: Energy and Memory used by the application Xcode debugger analysis.

Traffic Light Detection:

Vision Model:

The Caffe model we found was trained in the Nexar dataset that we were using for testing before. They found the validation loss they found is as follows:

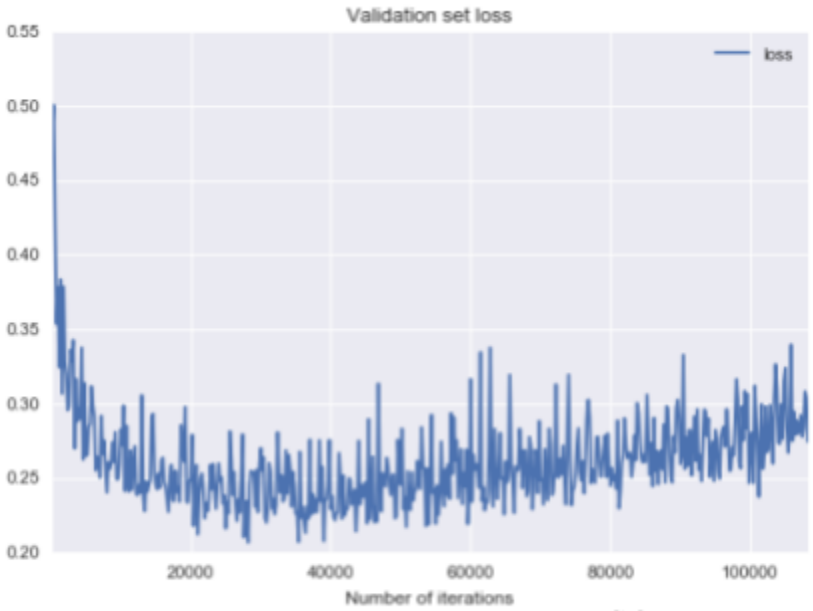

Figure 39: Validation loss of CaffeModel.[37]

As shown in figure 39, the validation loss was very low and when the model was overshooting, the developer stopped the training. After doing a validation test, they found the accuracy is around 92.9% for the Nexar dataset which is pretty high. To confirm this accuracy didn't change in the conversion process from Caffe to CoreML we did another test using the same dataset and found the following:

Figure 40: (Left) CoreML confidence to be green for Green traffic light. (Right) CoreML confidence to be red for Red Traffic light

Figure 40 shows when we gave green traffic light images, our prediction confidence was around 91%. The red light test gave us the confidence of ~92.5%. So the model wasn't really deprecated much between the conversion. Also, the green light confidence was lower as they had fewer data points as shown in figure 25 for green.

Vision Integration with Application:

After connecting the application with the model, we tested the application with our model traffic light.

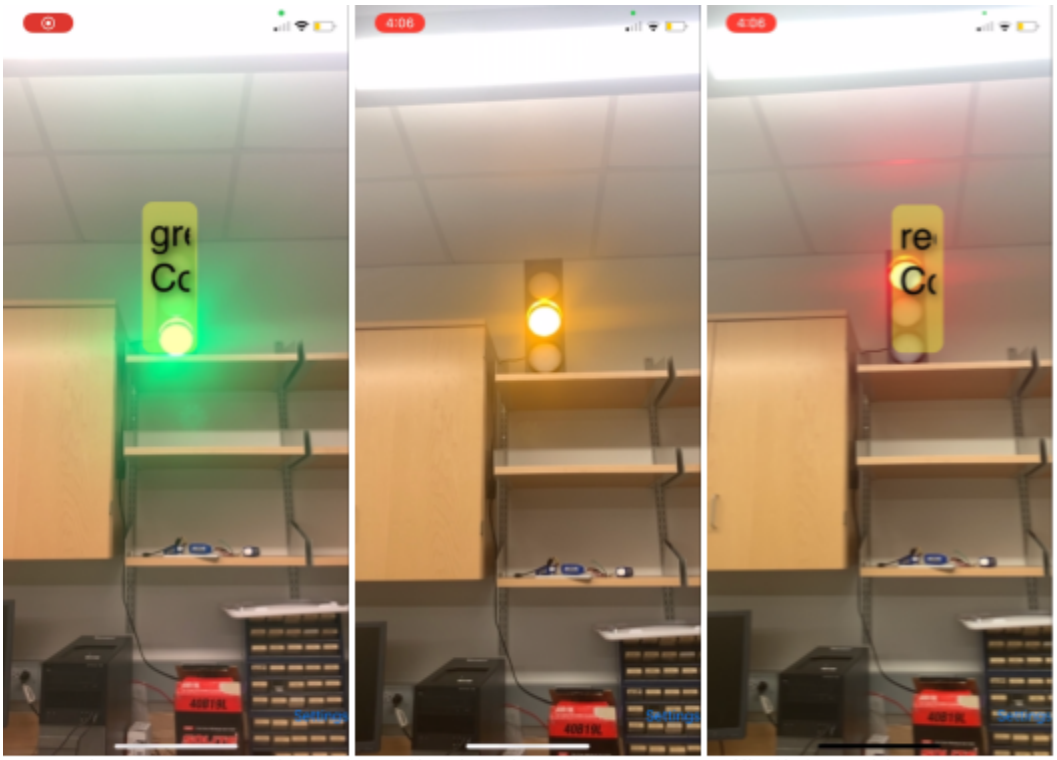

Figure 41: FriendlyWalk application detecting model traffic light and its states.

Figure 41 shows how we are able to read the red and green lights from the traffic light. We intentionally took out the yellow light detection since the confidence level was low. From these readings, we also saw that the confidence level was above 91% which is pretty high for a model traffic light detection. We also observed that when the light is above the eye level or when taking readings horizontally, or both, the confidence level increases significantly. It is mostly due to the training since all traffic lights in the training dataset were above the camera position (top section of the frame) and the images were horizontal. Thus, providing an image where the traffic light was above and having a wide angle gives us more precision.

Currently, we are reading and processing at a rate of 30 frames per second. giving the result to one of the selected options: auto, and manual. The manual decision is just doing a second check of the result and having enough delay between speaking so the wrong result is not spoken from the application. The auto decision state-machine in figure 25 was designed to make sure first that the user doesn't have to wait for more than 10 seconds which accomplishes our goal of not making the user wait more than 60 seconds. When the application doesn't find any traffic light, we also notify the user. For user's protection, when the state-machine receives a result, we always cross-check with 3 more frames which increases our confidence level according to Bayes Theorem.[38] If any cross-check doesn't validate, we repeat the process by removing caches. The system also has an additional rigorous checking for the Wait to Go state so we don't tell the user to cross at the wrong light. This overall allows us to accomplish our 100% confidence level goal when notifying the user.

Discussion, Recommendation, And Conclusion:

The application we built solves the intended problem of helping visually impaired individuals to crossroads. The system fulfills almost all the specifications and goals we set during the design phase. Here is a detailed discussion and recommendations on the subsystems for improvements:

Notification System for approaching a crosswalk:

The application solves the intended problem of advance notification while approaching the crosswalk for the safety of the user within 5 meters. The system had no error during the testing phase and had distance readings only off by +/- 3 feet which is negligible in a real-world implementation.

Due to time constraints, we were only able to integrate only one iBeacon to the app which didn't allow us to implement an orientation subsystem. This takes away the capability of letting the user know which side the traffic light is and also how they should adjust their hands to detect the traffic light. Although this shouldn't be difficult to build upon the current code as we already have the structure for a single iBeacon. The only thing needed for the next iteration is to find a way to label 4 different iBeacon such that we understand the exact positioning of the user. From there we would do distance calculations for each beacon and we would get the orientation.

Application:

We are able to run the application in the background which lets us accomplish our goal to save battery and not use unwanted processes other than iBeacon detection. This background running capability also unblocked the goal of running other navigation applications to navigate and also get crosswalk awareness notifications from our app. The whole application is also running without any need for an internet connection other than downloading to the phone for the first time.

For user inputs, we have implemented both voice and haptic touch. So every type of visually impaired individual can use the application with their own preference. Every output from the application and any commands the user selects, we give both haptic and audio responses as it was intended; even opening the application.

We have tried to give as much detail as possible for the user but this application has not been tested by any visually impaired individuals. It would be great if we could get visually impaired volunteers who would test and give us some feedback for the application.

Detecting Traffic Light:

The Caffe model we found already gave us an accuracy of around 91% within the application without any tweaking. We also implemented a cross-check of the results and found that our confidence level should go higher than 91% by Bayes theorem.[35] In addition, the state machine keeps the order of the traffic light and makes sure of two things: we only notify with 100% confidence, and don't make them wait more than 10 seconds for any output.

Since we didn't incorporate the yellow light for low accuracy, we are missing some information that would have made our system faster. For example, now when a user arrives at a yellow light, they have to wait till it goes from green to red again. This could be improved by adding more data to the datasets for training the model.

Fine-tuning the model is always helpful. Since we didn't match up the training image ratio with the phone camera, we are not getting the best out of the model. So tuning the model will surely help. Another testing and tuning are needed for the weather. Since we never tested outside or in any different weather conditions, we don't know the accuracy of the application. Thus, we need to test the application in the real world in different weather conditions.

Another great solution for improving the model would be to create a community version of the application where sighted users can capture traffic lights in their regular day and our system will automatically train from those data. This will allow both testing and training in different real-world conditions.

Even though the application was implemented and tested as it was intended, it could still be improved as this was the prototype phase. Building on it shouldn't be difficult since all the code and design followed every standard that was mentioned and kept on GitHub with proper comments.

References:

1. Pissaloux, Edwige, and Ramiro Velazquez, eds. “Mobility of Visually Impaired People.” (2018): n. pag. Crossref. Web.
2. *What is Tactile Paving? - CABVI - Central Association for the Blind and Visually Impaired*. CABVI. (2020, September 14). https://www.cabvi.org/articles/what-is-tactile-paving/.
3. Fusco G., Cheraghi S.A., Neat L., Coughlan J.M. (2020) An Indoor Navigation App Using Computer Vision and Sign Recognition. In: Miesenberger K., Manduchi R., Covarrubias Rodriguez M., Peňáz P. (eds) Computers Helping People with Special Needs. ICCHP 2020. Lecture Notes in Computer Science, vol 12376. Springer, Cham. https://doi.org/10.1007/978-3-030-58796-3_56
4. M. Poggi and S. Mattoccia, "A wearable mobility aid for the visually impaired based on embedded 3D vision and deep learning," 2016 IEEE Symposium on Computers and Communication (ISCC), 2016, pp. 208-213, doi: 10.1109/ISCC.2016.7543741.
5. Montes, H., I. Chang, G. Carballeda, J. Muñoz, A. Garcia, Vejarano+R., y Y. Saez. Design of a System to Support the Mobility of Visually Impaired People. Memorias De Congresos UTP, Vol. 1, n.º 1, Nov. 2018, pp. 37-44, https://revistas.utp.ac.pa/index.php/memoutp/article/view/1874.
6. *New York City Department of Transportation Traffic Signal Standard Drawings*. (n.d.). Retrieved November 24, 2021, from https://www1.nyc.gov/html/dot/downloads/pdf/nyc-dot-traffic-signal-standard-drawings.pdf.
7. *About IOS 9 updates*. Apple Support. (2020, March 2). Retrieved November 24, 2021, from https://support.apple.com/en-us/HT208010.
8. *Analyzing-your-apps-battery-use*. Apple Developer Documentation. (n.d.). Retrieved November 24, 2021, from https://developer.apple.com/documentation/xcode/analyzing-your-app-s-battery-use.
9. *Specifications - welcome to nyc.gov | city of New York*. (n.d.). Retrieved November 24, 2021, from https://www1.nyc.gov/html/dot/downloads/pdf/nycdot-traffic-signal-specifications.pdf.

10. S. Y. Fadhlullah and W. Ismail, "Solar energy harvesting design framework for 3.3 V small and low-powered devices in wireless sensor network," 2015 1st International Conference on Telematics and Future Generation Networks (TAFGEN), 2015, pp. 89-94, doi: 10.1109/TAFGEN.2015.7289583.
11. *DelDOT road design manual*. (n.d.). Retrieved November 24, 2021, from https://deldot.gov/Publications/manuals/road_design/pdfs/revisions062811/07_Intersections.pdf.
12. *Is the IOS app size limit 100 MB, 2 GB or 4 GB?* Is the iOS app size limit 100 MB, ... | Apple Developer Forums. (n.d.). Retrieved November 24, 2021, from https://developer.apple.com/forums/thread/12455.
13. Nayrolles, M. (2018). *Angular Design Patterns: Implement the gang of four patterns in your apps with angular*. Packt.
14. freeCodeCamp.org. (2020, March 11). *Test driven development: What it is, and what it is not.* freeCodeCamp.org. Retrieved November 24, 2021, from https://www.freecodecamp.org/news/test-driven-development-what-it-is-and-what-it-is-not-41fa6bca02a2/.
15. Peroni S. (2017) A Simplified Agile Methodology for Ontology Development. In: Dragoni M., Poveda-Villalón M., Jimenez-Ruiz E. (eds) OWL: Experiences and Directions – Reasoner Evaluation. OWLED 2016, ORE 2016. Lecture Notes in Computer Science, vol 10161. Springer, Cham. https://doi.org/10.1007/978-3-319-54627-8_5.
16. A. Srivastava, S. Bhardwaj and S. Saraswat, "SCRUM model for agile methodology," *2017 International Conference on Computing, Communication and Automation (ICCCA)*, 2017, pp. 864-869, doi: 10.1109/CCAA.2017.8229928.
17. Inc., A. (n.d.). *Api Design Guidelines*. Swift.org. Retrieved November 24, 2021, from https://www.swift.org/documentation/api-design-guidelines/.
18. *Accessibility Guidelines*. Apple Developer Documentation. (n.d.). Retrieved November 24, 2021, from https://developer.apple.com/documentation/accessibility.
19. *Jetson Nano Developer Kit*. NVIDIA Developer. (2021, April 14). Retrieved November 24, 2021, from https://developer.nvidia.com/embedded/jetson-nano-developer-kit.
20. Nayrolles, M. (2018). *Angular Design Patterns: Implement the gang of four patterns in your apps with angular*. Packt.

21. *Madison Storm and Katie Wolfkiel JANUARY 5. (2021, January 6). Best cell phones for the visually impaired [2020]. UpPhone. Retrieved October 22, 2021, from https://www.upphone.com/learn/deals/buyers-guides-ld/best-cell-phone-for-visually-impaired/.*
22. *Android*. OpenCV. (2019, April 19). Retrieved November 24, 2021, from https://opencv.org/android/.
23. *The Smartphone: a Revolution for the Blind and Visually Impaired! Inclusive City Maker.* (2021, May 6). https://www.inclusivecitymaker.com/the-smartphone-a-revolution-for-the-blind-and-visually-impaired/.
24. *Does bluetooth drain your battery?* Tile. (n.d.). Retrieved November 24, 2021, from https://www.thetileapp.com/en-us/blog/does-bluetooth-drain-battery.
25. Shimpi, A. L., & Klug, B. (2011, October 31). *Apple iPhone 4S: Thoroughly reviewed*. RSS. Retrieved November 24, 2021, from https://www.anandtech.com/show/4971/apple-iphone-4s-review-att-verizon/15.
26. Isuyama, V. K., & Albertini, B. D. (2021). Comparison of convolutional neural network models for mobile devices. *Anais Do XX Workshop Em Desempenho De Sistemas Computacionais e De Comunicação (WPerformance 2021)*. https://doi.org/10.5753/wperformance.2021.15724
27. *CIColor.* Apple Developer Documentation. (n.d.). Retrieved November 24, 2021, from https://developer.apple.com/documentation/coreimage/cicolor.
28. *CIColor Kernel*. Apple Developer Documentation. (n.d.). Retrieved November 24, 2021, from https://developer.apple.com/documentation/coreimage/cicolorkernel.
29. *Getting started with iBeacon - Apple Developer*. (n.d.). Retrieved November 24, 2021, from https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf.
30. Fengjun Shang, W. S. (2012, August 1). *A location estimation algorithm based on RSSI vector similarity degree - Fengjun Shang, Wen Su, Qian Wang, Hongxia Gao, Qiang Fu, 2014*. SAGE Journals. Retrieved November 24, 2021, from https://journals.sagepub.com/doi/full/10.1155/2014/371350.
31. Paek, J., Ko, J. G., & Shin, H. (2016, October 24). *A measurement study of Ble iBeacon and geometric adjustment scheme for indoor location-based mobile applications*. Mobile

Information Systems. Retrieved November 24, 2021, from https://www.hindawi.com/journals/misy/2016/8367638/.

32. *Nexar Challenge #1 using Deep Learning for traffic light RECOGNITIONR*. Nexar. (n.d.). Retrieved November 24, 2021, from https://challenge.getnexar.com/challenge-1/.
33. Silver, D. (2018, February 7). *The "Traffic Sign Classifier" project*. Medium. Retrieved November 24, 2021, from https://medium.com/udacity/the-traffic-sign-classifier-project-1c85a2eb9db5#:~:text=Traffic%20Sign%20Classifier%20is%20the,Traffic%20Sign%20Recognition%20Benchmark%20dataset.
34. *IOS Swift uiimage subscript extension to get pixel color*. Gist. (n.d.). Retrieved March 12, 2022, from https://gist.github.com/marchinram/3675efc96bf1cc2c02a5
35. Cooper, C. C. (2021, September 21). *Convert a Caffe model to core ML format*. WWT. Retrieved March 13, 2022, from https://www.wwt.com/article/convert-a-caffe-model-to-core-ml-format
36. Handling Location Events in the Background. Apple Developer Documentation. (n.d.). Retrieved March 16, 2022, from https://developer.apple.com/documentation/corelocation/getting_the_user_s_location/handling_location_events_in_the_background
37. Brailovsky, D. (2017, April 19). Recognizing traffic lights with deep learning. Medium. Retrieved March 17, 2022, from https://medium.com/free-code-camp/recognizing-traffic-lights-with-deep-learning-23dae23287cc
38. Brownlee, J. (2019, December 3). A gentle introduction to Bayes theorem for Machine Learning. Machine Learning Mastery. Retrieved March 17, 2022, from https://machinelearningmastery.com/bayes-theorem-for-machine-learning/

Appendix:

Loading Page:

Figure 42: Loading page of the application

Personal filter:

```
import CoreImage
var redKernel_str =
  """
    kernel vec4 thresholdFilter(__sample textureColor) {

      if (textureColor.r > 0.6 && textureColor.g < 0.1 && textureColor.b < 0.4) {
        textureColor.rgb = vec3(1.0, 1.0, 1.0);
      } else {
        textureColor.rgb = vec3(0.0 , 0.0, 0.0);
      }

      return textureColor;
    }
    """

var greenKernel_str =
  """
    kernel vec4 thresholdFilter(__sample textureColor) {

      if (textureColor.r < 0.08 && textureColor.g > 0.1 && textureColor.b < 2) {
        textureColor.rgb = vec3(1.0, 1.0, 1.0);
      } else {
        textureColor.rgb = vec3(0.0 , 0.0, 0.0);
      }
```

```
        return textureColor;
    }
    """

public class CustomFilters {
    public var redKernel: CIColorKernel;
    public var greenKernel: CIColorKernel;
    public init() {
        self.redKernel = CIColorKernel(source: redKernel_str)!
        self.greenKernel = CIColorKernel(source: greenKernel_str)!
    }

    public func redFilter(ciInputImage: CIImage)->CIImage{
        let ciImageFiltered = self.redKernel.apply(
            extent: ciInputImage.extent,
            arguments: [ciInputImage])!
        return ciImageFiltered
    }

    public func greenFilter(ciInputImage: CIImage)->CIImage{
        let ciImageFiltered = self.greenKernel.apply(
            extent: ciInputImage.extent,
            arguments: [ciInputImage])!
        return ciImageFiltered
    }
}
```

Key ViewController Code:

```
// Copyright 2019 The TensorFlow Authors. All Rights Reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//    http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
```

```
import UIKit

class ViewController: UIViewController {

  // MARK: Storyboards Connections
  @IBOutlet weak var previewView: PreviewView!
  @IBOutlet weak var overlayView: OverlayView!
  @IBOutlet weak var resumeButton: UIButton!
  @IBOutlet weak var cameraUnavailableLabel: UILabel!

  @IBOutlet weak var bottomSheetStateImageView: UIImageView!
  @IBOutlet weak var bottomSheetView: UIView!
  @IBOutlet weak var bottomSheetViewBottomSpace: NSLayoutConstraint!

  // MARK: Constants
  private let displayFont = UIFont.systemFont(ofSize: 14.0, weight: .medium)
  private let edgeOffset: CGFloat = 2.0
  private let labelOffset: CGFloat = 10.0
  private let animationDuration = 0.5
  private let collapseTransitionThreshold: CGFloat = -30.0
  private let expandTransitionThreshold: CGFloat = 30.0
  private let delayBetweenInferencesMs: Double = 200

  // MARK: Instance Variables
  private var initialBottomSpace: CGFloat = 0.0

  // Holds the results at any time
  private var result: Result?
  private var previousInferenceTimeMs: TimeInterval = Date.distantPast.timeIntervalSince1970 * 1000

  // MARK: Controllers that manage functionality
  private lazy var cameraFeedManager = CameraFeedManager(previewView: previewView)
  private var modelDataHandler: ModelDataHandler? =
    ModelDataHandler(modelFileInfo: MobileNetSSD.modelInfo, labelsFileInfo: MobileNetSSD.labelsInfo)
  private var inferenceViewController: InferenceViewController?

  // MARK: View Handling Methods
  override func viewDidLoad() {
    super.viewDidLoad()

    guard modelDataHandler != nil else {
      fatalError("Failed to load model")
```

```
  }
  cameraFeedManager.delegate = self
  overlayView.clearsContextBeforeDrawing = true

  addPanGesture()
}

override func didReceiveMemoryWarning() {
  super.didReceiveMemoryWarning()
  // Dispose of any resources that can be recreated.
}

override func viewWillAppear(_ animated: Bool) {
  super.viewWillAppear(animated)
  changeBottomViewState()
  cameraFeedManager.checkCameraConfigurationAndStartSession()
}

override func viewWillDisappear(_ animated: Bool) {
  super.viewWillDisappear(animated)

  cameraFeedManager.stopSession()
}

override var preferredStatusBarStyle: UIStatusBarStyle {
  return .lightContent
}

// MARK: Button Actions
@IBAction func onClickResumeButton(_ sender: Any) {

  cameraFeedManager.resumeInterruptedSession { (complete) in

    if complete {
      self.resumeButton.isHidden = true
      self.cameraUnavailableLabel.isHidden = true
    }
    else {
      self.presentUnableToResumeSessionAlert()
    }
  }
}
```

```
  func presentUnableToResumeSessionAlert() {
    let alert = UIAlertController(
      title: "Unable to Resume Session",
      message: "There was an error while attempting to resume session.",
      preferredStyle: .alert
    )
    alert.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))

    self.present(alert, animated: true)
  }

  // MARK: Storyboard Segue Handlers
  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    super.prepare(for: segue, sender: sender)

    if segue.identifier == "EMBED" {

      guard let tempModelDataHandler = modelDataHandler else {
        return
      }
      inferenceViewController = segue.destination as? InferenceViewController
      inferenceViewController?.wantedInputHeight = tempModelDataHandler.inputHeight
      inferenceViewController?.wantedInputWidth = tempModelDataHandler.inputWidth
      inferenceViewController?.threadCountLimit = tempModelDataHandler.threadCountLimit
      inferenceViewController?.currentThreadCount = tempModelDataHandler.threadCount
      inferenceViewController?.delegate = self

      guard let tempResult = result else {
        return
      }
      inferenceViewController?.inferenceTime = tempResult.inferenceTime

    }
  }
}

// MARK: InferenceViewControllerDelegate Methods
extension ViewController: InferenceViewControllerDelegate {

  func didChangeThreadCount(to count: Int) {
    if modelDataHandler?.threadCount == count { return }
```

```
    modelDataHandler = ModelDataHandler(
      modelFileInfo: MobileNetSSD.modelInfo,
      labelsFileInfo: MobileNetSSD.labelsInfo,
      threadCount: count
    )
  }

}

// MARK: CameraFeedManagerDelegate Methods
extension ViewController: CameraFeedManagerDelegate {

  func didOutput(pixelBuffer: CVPixelBuffer) {
    runModel(onPixelBuffer: pixelBuffer)
  }

  // MARK: Session Handling Alerts
  func sessionRunTimeErrorOccurred() {

    // Handles session run time error by updating the UI and providing a button if session can be manually resumed.
    self.resumeButton.isHidden = false
  }

  func sessionWasInterrupted(canResumeManually resumeManually: Bool) {

    // Updates the UI when session is interrupted.
    if resumeManually {
      self.resumeButton.isHidden = false
    }
    else {
      self.cameraUnavailableLabel.isHidden = false
    }
  }

  func sessionInterruptionEnded() {

    // Updates UI once session interruption has ended.
    if !self.cameraUnavailableLabel.isHidden {
      self.cameraUnavailableLabel.isHidden = true
    }

    if !self.resumeButton.isHidden {
```

```
    self.resumeButton.isHidden = true
  }
}

func presentVideoConfigurationErrorAlert() {

  let alertController = UIAlertController(title: "Configuration Failed", message: "Configuration of camera has failed.",
      preferredStyle: .alert)
  let okAction = UIAlertAction(title: "OK", style: .cancel, handler: nil)
  alertController.addAction(okAction)

  present(alertController, animated: true, completion: nil)
}

func presentCameraPermissionsDeniedAlert() {

  let alertController = UIAlertController(title: "Camera Permissions Denied", message: "Camera permissions have been
      denied for this app. You can change this by going to Settings", preferredStyle: .alert)

  let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
  let settingsAction = UIAlertAction(title: "Settings", style: .default) { (action) in

    UIApplication.shared.open(URL(string: UIApplication.openSettingsURLString)!, options: [:], completionHandler: nil)
  }

  alertController.addAction(cancelAction)
  alertController.addAction(settingsAction)

  present(alertController, animated: true, completion: nil)

}

/** This method runs the live camera pixelBuffer through tensorFlow to get the result.
 */
@objc  func runModel(onPixelBuffer pixelBuffer: CVPixelBuffer) {

  // Run the live camera pixelBuffer through tensorFlow to get the result

  let currentTimeMs = Date().timeIntervalSince1970 * 1000

  guard  (currentTimeMs - previousInferenceTimeMs) >= delayBetweenInferencesMs else {
    return
```

```
    }

    previousInferenceTimeMs = currentTimeMs
    result = self.modelDataHandler?.runModel(onFrame: pixelBuffer)

    guard let displayResult = result else {
      return
    }

    let width = CVPixelBufferGetWidth(pixelBuffer)
    let height = CVPixelBufferGetHeight(pixelBuffer)

    DispatchQueue.main.async {

      // Display results by handing off to the InferenceViewController
      self.inferenceViewController?.resolution = CGSize(width: width, height: height)

      var inferenceTime: Double = 0
      if let resultInferenceTime = self.result?.inferenceTime {
        inferenceTime = resultInferenceTime
      }
      self.inferenceViewController?.inferenceTime = inferenceTime
      self.inferenceViewController?.tableView.reloadData()

      // Draws the bounding boxes and displays class names and confidence scores.
      self.drawAfterPerformingCalculations(onInferences: displayResult.inferences, withImageSize: CGSize(width:
        CGFloat(width), height: CGFloat(height)))
    }
  }

  /**
   This method takes the results, translates the bounding box rects to the current view, draws the bounding boxes, classNames
      and confidence scores of inferences.
   */
  func drawAfterPerformingCalculations(onInferences inferences: [Inference], withImageSize imageSize:CGSize) {

    self.overlayView.objectOverlays = []
    self.overlayView.setNeedsDisplay()

    guard !inferences.isEmpty else {
      return
    }
```

```
 var objectOverlays: [ObjectOverlay] = []

 for inference in inferences {

  // Translates bounding box rect to current view.
  var convertedRect = inference.rect.applying(CGAffineTransform(scaleX: self.overlayView.bounds.size.width /
   imageSize.width, y: self.overlayView.bounds.size.height / imageSize.height))

  if convertedRect.origin.x < 0 {
   convertedRect.origin.x = self.edgeOffset
  }

  if convertedRect.origin.y < 0 {
   convertedRect.origin.y = self.edgeOffset
  }

  if convertedRect.maxY > self.overlayView.bounds.maxY {
   convertedRect.size.height = self.overlayView.bounds.maxY - convertedRect.origin.y - self.edgeOffset
  }

  if convertedRect.maxX > self.overlayView.bounds.maxX {
   convertedRect.size.width = self.overlayView.bounds.maxX - convertedRect.origin.x - self.edgeOffset
  }

  let confidenceValue = Int(inference.confidence * 100.0)
  let string = "\(inference.className)  (\(confidenceValue)%)"

  let size = string.size(usingFont: self.displayFont)

  let objectOverlay = ObjectOverlay(name: string, borderRect: convertedRect, nameStringSize: size, color:
   inference.displayColor, font: self.displayFont)

  objectOverlays.append(objectOverlay)
 }

 // Hands off drawing to the OverlayView
 self.draw(objectOverlays: objectOverlays)

}

/** Calls methods to update overlay view with detected bounding boxes and class names.
```

```
  */
  func draw(objectOverlays: [ObjectOverlay]) {

    self.overlayView.objectOverlays = objectOverlays
    self.overlayView.setNeedsDisplay()
  }

}

// MARK: Bottom Sheet Interaction Methods
extension ViewController {

  // MARK: Bottom Sheet Interaction Methods
  /**
   This method adds a pan gesture to make the bottom sheet interactive.
   */
  private func addPanGesture() {
    let panGesture = UIPanGestureRecognizer(target: self, action: #selector(ViewController.didPan(panGesture:)))
    bottomSheetView.addGestureRecognizer(panGesture)
  }


  /** Change whether bottom sheet should be in expanded or collapsed state.
   */
  private func changeBottomViewState() {

    guard let inferenceVC = inferenceViewController else {
      return
    }

    if bottomSheetViewBottomSpace.constant == inferenceVC.collapsedHeight - bottomSheetView.bounds.size.height {

      bottomSheetViewBottomSpace.constant = 0.0
    }
    else {
      bottomSheetViewBottomSpace.constant = inferenceVC.collapsedHeight - bottomSheetView.bounds.size.height
    }
    setImageBasedOnBottomViewState()
  }

  /**
   Set image of the bottom sheet icon based on whether it is expanded or collapsed
```

```
 */
private func setImageBasedOnBottomViewState() {

  if bottomSheetViewBottomSpace.constant == 0.0 {
    bottomSheetStateImageView.image = UIImage(named: "down_icon")
  }
  else {
    bottomSheetStateImageView.image = UIImage(named: "up_icon")
  }
}

/**
 This method responds to the user panning on the bottom sheet.
 */
@objc func didPan(panGesture: UIPanGestureRecognizer) {

  // Opens or closes the bottom sheet based on the user's interaction with the bottom sheet.
  let translation = panGesture.translation(in: view)

  switch panGesture.state {
  case .began:
    initialBottomSpace = bottomSheetViewBottomSpace.constant
    translateBottomSheet(withVerticalTranslation: translation.y)
  case .changed:
    translateBottomSheet(withVerticalTranslation: translation.y)
  case .cancelled:
    setBottomSheetLayout(withBottomSpace: initialBottomSpace)
  case .ended:
    translateBottomSheetAtEndOfPan(withVerticalTranslation: translation.y)
    setImageBasedOnBottomViewState()
    initialBottomSpace = 0.0
  default:
    break
  }
}

/**
 This method sets bottom sheet translation while pan gesture state is continuously changing.
 */
private func translateBottomSheet(withVerticalTranslation verticalTranslation: CGFloat) {

  let bottomSpace = initialBottomSpace - verticalTranslation
```

```
  guard bottomSpace <= 0.0 && bottomSpace >= inferenceViewController!.collapsedHeight -
    bottomSheetView.bounds.size.height else {
   return
  }
  setBottomSheetLayout(withBottomSpace: bottomSpace)
}

/**
 This method changes bottom sheet state to either fully expanded or closed at the end of pan.
 */
private func translateBottomSheetAtEndOfPan(withVerticalTranslation verticalTranslation: CGFloat) {

  // Changes bottom sheet state to either fully open or closed at the end of pan.
  let bottomSpace = bottomSpaceAtEndOfPan(withVerticalTranslation: verticalTranslation)
  setBottomSheetLayout(withBottomSpace: bottomSpace)
}

/**
 Return the final state of the bottom sheet view (whether fully collapsed or expanded) that is to be retained.
 */
private func bottomSpaceAtEndOfPan(withVerticalTranslation verticalTranslation: CGFloat) -> CGFloat {

  // Calculates whether to fully expand or collapse bottom sheet when pan gesture ends.
  var bottomSpace = initialBottomSpace - verticalTranslation

  var height: CGFloat = 0.0
  if initialBottomSpace == 0.0 {
   height = bottomSheetView.bounds.size.height
  }
  else {
   height = inferenceViewController!.collapsedHeight
  }

  let currentHeight = bottomSheetView.bounds.size.height + bottomSpace

  if currentHeight - height <= collapseTransitionThreshold {
   bottomSpace = inferenceViewController!.collapsedHeight - bottomSheetView.bounds.size.height
  }
  else if currentHeight - height >= expandTransitionThreshold {
   bottomSpace = 0.0
  }
  else {
```

```
            bottomSpace = initialBottomSpace
        }

        return bottomSpace
    }

    /**
     This method layouts the change of the bottom space of bottom sheet with respect to the view managed by this controller.
     */
    func setBottomSheetLayout(withBottomSpace bottomSpace: CGFloat) {

        view.setNeedsLayout()
        bottomSheetViewBottomSpace.constant = bottomSpace
        view.setNeedsLayout()
    }

}
```

AverageColor

```
import CoreImage
import CoreImage.CIFilterBuiltins
import UIKit
import Foundation


public func get_color(image: UIImage) ->[Int]?{
    guard let cgImage = image.cgImage else { return nil}
    var colorSpace = CGColorSpaceCreateDeviceRGB()

    var bitmapInfo: UInt32 = CGBitmapInfo.byteOrder32Big.rawValue
    bitmapInfo |= CGImageAlphaInfo.premultipliedLast.rawValue & CGBitmapInfo.alphaInfoMask.rawValue

    let width = Int(image.size.width)
    let height = Int(image.size.height)
    var bytesPerRow = width * 4

    let imageData = UnsafeMutablePointer<Pixel>.allocate(capacity: width * height)


    guard let imageContext = CGContext(
        data: imageData,
        width: width,
        height: height,
        bitsPerComponent: 8,
        bytesPerRow: bytesPerRow,
        space: colorSpace,
        bitmapInfo: bitmapInfo
    ) else { return nil}
```

```
    imageContext.draw(cgImage, in: CGRect(x: 0, y: 0, width: width, height: height))
    let pixels = UnsafeMutableBufferPointer<Pixel>(start: imageData, count: width * height)

    var totalRed = 0
    var totalGreen = 0
    var totalBlue = 0
    let pixelArea = width * height

    for y in 0..<height {
        for x in 0..<width {
            let index = y * width + x
            let pixel = pixels[index]
//            print("pixel red:",pixel.red)
            totalRed += Int(pixel.red)

//            print("total red",totalRed)
            totalGreen += Int(pixel.green)
            totalBlue += Int(pixel.blue)
        }
    }

    return [totalRed,totalGreen,totalBlue]
}


public struct Pixel {
    public var value: UInt32

    public var red: UInt8 {
        get {
            return UInt8(value & 0xFF)
        } set {
            value = UInt32(newValue) | (value & 0xFFFFFF00)
        }
    }

    public var green: UInt8 {
        get {
            return UInt8((value >> 8) & 0xFF)
        } set {
            value = (UInt32(newValue) << 8) | (value & 0xFFFF00FF)
        }
    }

    public var blue: UInt8 {
        get {
            return UInt8((value >> 16) & 0xFF)
        } set {
            value = (UInt32(newValue) << 16) | (value & 0xFF00FFFF)
        }
    }

    public var alpha: UInt8 {
        get {
            return UInt8((value >> 24) & 0xFF)
        } set {
            value = (UInt32(newValue) << 24) | (value & 0x00FFFFFF)
        }
    }
}
```

Friendly Walk App:

AppDelegate.swift

```
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        return true
    }

}
```

ViewController.swift

```
import UIKit
import AVFoundation
import Vision
import SwiftUI

@available(iOS 14.0, *)
class ViewController: UIViewController, AVCaptureVideoDataOutputSampleBufferDelegate {

    let navigationView = UIHostingController(rootView: NavigationView())
    var bufferSize: CGSize = .zero
    var rootLayer: CALayer! = nil

    @IBOutlet weak private var previewView: UIView!
    private let session = AVCaptureSession()
    private var previewLayer: AVCaptureVideoPreviewLayer! = nil
    private let videoDataOutput = AVCaptureVideoDataOutput()

    private let videoDataOutputQueue = DispatchQueue(label: "VideoDataOutput", qos: .userInitiated, attributes: [],
autoreleaseFrequency: .workItem)

    func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer, from connection:
AVCaptureConnection) {
        // to be implemented in the subclass
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        setupAVCapture()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
```

```
    }

    fileprivate func setupConstraints() {
        navigationView.view.backgroundColor = UIColor.clear // Needed to not hide other layers
        navigationView.view.translatesAutoresizingMaskIntoConstraints = false
        navigationView.view.topAnchor.constraint(equalTo: view.topAnchor).isActive = true
        navigationView.view.bottomAnchor.constraint(equalTo: view.bottomAnchor).isActive = true
        navigationView.view.leftAnchor.constraint(equalTo: view.leftAnchor).isActive = true
        navigationView.view.rightAnchor.constraint(equalTo: view.rightAnchor).isActive = true
    }

    func setupAVCapture() {
        var deviceInput: AVCaptureDeviceInput!

        // Select a video device, make an input
        let videoDevice = AVCaptureDevice.DiscoverySession(deviceTypes: [.builtInWideAngleCamera], mediaType: .video,
position: .back).devices.first
        do {
            deviceInput = try AVCaptureDeviceInput(device: videoDevice!)
        } catch {
            print("Could not create video device input: \(error)")
            return
        }

        session.beginConfiguration()
        session.sessionPreset = .vga640x480 // Model image size is smaller.

        // Add a video input
        guard session.canAddInput(deviceInput) else {
            print("Could not add video device input to the session")
            session.commitConfiguration()
            return
        }
        session.addInput(deviceInput)
        if session.canAddOutput(videoDataOutput) {
            session.addOutput(videoDataOutput)
            // Add a video data output
            videoDataOutput.alwaysDiscardsLateVideoFrames = true
            videoDataOutput.videoSettings = [kCVPixelBufferPixelFormatTypeKey as String:
Int(kCVPixelFormatType_420YpCbCr8BiPlanarFullRange)]
            videoDataOutput.setSampleBufferDelegate(self, queue: videoDataOutputQueue)
        } else {
            print("Could not add video data output to the session")
            session.commitConfiguration()
            return
        }
        let captureConnection = videoDataOutput.connection(with: .video)
        // Always process the frames
        captureConnection?.isEnabled = true
        do {
            try  videoDevice!.lockForConfiguration()
            let dimensions = CMVideoFormatDescriptionGetDimensions((videoDevice?.activeFormat.formatDescription)!)
            bufferSize.width = CGFloat(dimensions.width)
            bufferSize.height = CGFloat(dimensions.height)
```

```
            videoDevice!.unlockForConfiguration()
        } catch {
            print(error)
        }
        session.commitConfiguration()
        previewLayer = AVCaptureVideoPreviewLayer(session: session)
        previewLayer.videoGravity = AVLayerVideoGravity.resizeAspectFill
        rootLayer = previewView.layer
        previewLayer.frame = rootLayer.bounds
        rootLayer.addSublayer(previewLayer)

        addChild(navigationView) // Allows embedding the custom SwiftUI view
        view.addSubview(navigationView.view)
        setupConstraints()
    }

    func startCaptureSession() {
        session.startRunning()
    }

    // Clean up capture setup
    func teardownAVCapture() {
        previewLayer.removeFromSuperlayer()
        previewLayer = nil
    }

    func captureOutput(_ captureOutput: AVCaptureOutput, didDrop didDropSampleBuffer: CMSampleBuffer, from connection:
AVCaptureConnection) {
        // print("frame dropped")
    }

    public func exifOrientationFromDeviceOrientation() -> CGImagePropertyOrientation {
        let curDeviceOrientation = UIDevice.current.orientation
        let exifOrientation: CGImagePropertyOrientation

        switch curDeviceOrientation {
        case UIDeviceOrientation.portraitUpsideDown:  // Device oriented vertically, home button on the top
            exifOrientation = .left
        case UIDeviceOrientation.landscapeLeft:       // Device oriented horizontally, home button on the right
            exifOrientation = .upMirrored
        case UIDeviceOrientation.landscapeRight:      // Device oriented horizontally, home button on the left
            exifOrientation = .down
        case UIDeviceOrientation.portrait:            // Device oriented vertically, home button on the bottom
            exifOrientation = .up
        default:
            exifOrientation = .up
        }
        return exifOrientation
    }
}
```

VisionObjectDetectionController.swift

```
import UIKit
```

```
import AVFoundation
import Vision



@available(iOS 14.0, *)
class VisionObjectRecognitionViewController: ViewController {

    private var detectionOverlay: CALayer! = nil

    var crossingState = CrossingState()

    @IBAction func settingButton(_ sender: Any) {
        print("Setting button")
    }

    // Vision parts
    private var requests = [VNRequest]()

    @discardableResult
    func setupVision() -> NSError? {
        // Setup Vision parts
        let error: NSError! = nil

        guard let modelURL = Bundle.main.url(forResource: "TrafficLight", withExtension: "mlmodelc") else {
            return NSError(domain: "VisionObjectRecognitionViewController", code: -1, userInfo: [NSLocalizedDescriptionKey:
"Model file is missing"])
        }
        do {
            let visionModel = try VNCoreMLModel(for: MLModel(contentsOf: modelURL))
            let objectRecognition = VNCoreMLRequest(model: visionModel, completionHandler: { (request, error) in
                DispatchQueue.main.async(execute: {
                    // perform all the UI updates on the main queue
                    if let results = request.results {
                        self.drawVisionRequestResults(results)
                    }
                })
            })
            self.requests = [objectRecognition]
        } catch let error as NSError {
            print("Model loading went wrong: \(error)")
        }

        return error
    }

    //used for correcting out of index for deleting other labels
    func addExtrachar(str: String) -> String{
        if str.count < 4{
            return str + "****"
        }
        return str
    }
```

```
    func drawVisionRequestResults(_ results: [Any]) {
        CATransaction.begin()
        CATransaction.setValue(kCFBooleanTrue, forKey: kCATransactionDisableActions)
        detectionOverlay.sublayers = nil // remove all the old recognized objects
        for observation in results where observation is VNRecognizedObjectObservation {
            guard let objectObservation = observation as? VNRecognizedObjectObservation else {
                continue
            }
            // Select only the label with the highest confidence.
            let topLabelObservation = objectObservation.labels[0]

            /**
                        This section is not necessary but used to imporve the accuracy only for traffic light
             */
//            var i=0;
//            var labelOfTop = addExtrachar(str: topLabelObservation.identifier)
//            while (labelOfTop[labelOfTop.index(labelOfTop.startIndex, offsetBy: 3)] != "f" && (i < 3) ){
//                i+=1
//                print(labelOfTop[labelOfTop.index(labelOfTop.startIndex, offsetBy: 3)])
//                topLabelObservation = objectObservation.labels[i]
//                labelOfTop = addExtrachar(str: topLabelObservation.identifier)
//            }
//            if i == 3{
//                topLabelObservation = objectObservation.labels[0]
//            }
            /***************************/

            let objectBounds = VNImageRectForNormalizedRect(objectObservation.boundingBox, Int(bufferSize.width),
Int(bufferSize.height))

            let shapeLayer = self.createRoundedRectLayerWithBounds(objectBounds)

            let textLayer = self.createTextSubLayerInBounds(objectBounds,
                                                            identifier: topLabelObservation.identifier,
                                                            confidence: topLabelObservation.confidence)
            print(textLayer)
            shapeLayer.addSublayer(textLayer)
            detectionOverlay.addSublayer(shapeLayer)
        }
        self.updateLayerGeometry()
        CATransaction.commit()
    }

    override func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer, from connection:
AVCaptureConnection) {
        guard let pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer) else {
            return
        }

        let exifOrientation = exifOrientationFromDeviceOrientation()

        let imageRequestHandler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer, orientation: exifOrientation, options: [:])
        do {
            try imageRequestHandler.perform(self.requests)
```

```
    } catch {
        print(error)
    }
}

override func setupAVCapture() {
    super.setupAVCapture()

    // setup Vision parts
    setupLayers()
    updateLayerGeometry()
    setupVision()

    // start the capture
    startCaptureSession()
}

func setupLayers() {
    detectionOverlay = CALayer() // container layer that has all the renderings of the observations
    detectionOverlay.name = "DetectionOverlay"
    detectionOverlay.bounds = CGRect(x: 0.0,
                                     y: 0.0,
                                     width: bufferSize.width,
                                     height: bufferSize.height)
    detectionOverlay.position = CGPoint(x: rootLayer.bounds.midX, y: rootLayer.bounds.midY)
    rootLayer.addSublayer(detectionOverlay)
}

func updateLayerGeometry() {
    let bounds = rootLayer.bounds
    var scale: CGFloat

    let xScale: CGFloat = bounds.size.width / bufferSize.width
    let yScale: CGFloat = bounds.size.height / bufferSize.height

    scale = fmax(xScale, yScale)
    if scale.isInfinite {
        scale = 1.0
    }
    CATransaction.begin()
    CATransaction.setValue(kCFBooleanTrue, forKey: kCATransactionDisableActions)

    // rotate the layer into screen orientation and scale and mirror
    detectionOverlay.setAffineTransform(CGAffineTransform(rotationAngle: CGFloat(.pi / 2.0)).scaledBy(x: scale, y: -scale))
    // center the layer
    detectionOverlay.position = CGPoint(x: bounds.midX, y: bounds.midY)

    CATransaction.commit()

}

func createTextSubLayerInBounds(_ bounds: CGRect, identifier: String, confidence: VNConfidence) -> CATextLayer {
    let textLayer = CATextLayer()
    textLayer.name = "Object Label"
```

```
        var label = identifier
        if (identifier == "traffic_light_red"){
            label = "red"
        }else if (identifier == "traffic_light_green"){
            label = "green"
        }
        let autoDecide = UserDefaults.standard.bool(forKey: "autoDecide")

        if autoDecide{
            crossingState.stateWiseUpdate(result: label)
        }else{
            crossingState.update(result: label)
        }


        let formattedString = NSMutableAttributedString(string: String(format: "\(label)\nConfidence:  %.2f", confidence))
        let largeFont = UIFont(name: "Helvetica", size: 24.0)!
        formattedString.addAttributes([NSAttributedString.Key.font: largeFont], range: NSRange(location: 0, length:
identifier.count))
        textLayer.string = formattedString
        textLayer.bounds = CGRect(x: 0, y: 0, width: bounds.size.height - 10, height: bounds.size.width - 10)
        textLayer.position = CGPoint(x: bounds.midX, y: bounds.midY)
        textLayer.shadowOpacity = 0.7
        textLayer.shadowOffset = CGSize(width: 2, height: 2)
        textLayer.foregroundColor = CGColor(colorSpace: CGColorSpaceCreateDeviceRGB(), components: [0.0, 0.0, 0.0, 1.0])
        textLayer.contentsScale = 2.0 // retina rendering
        // rotate the layer into screen orientation and scale and mirror
        textLayer.setAffineTransform(CGAffineTransform(rotationAngle: CGFloat(.pi / 2.0)).scaledBy(x: 1.0, y: -1.0))
        return textLayer
    }

    func createRoundedRectLayerWithBounds(_ bounds: CGRect) -> CALayer {
        let shapeLayer = CALayer()
        shapeLayer.bounds = bounds
        shapeLayer.position = CGPoint(x: bounds.midX, y: bounds.midY)
        shapeLayer.name = "Found Object"
        shapeLayer.backgroundColor = CGColor(colorSpace: CGColorSpaceCreateDeviceRGB(), components: [1.0, 1.0, 0.2, 0.4])
        shapeLayer.cornerRadius = 7
        return shapeLayer
    }

}
```

CrossingState.swift

```
import Foundation
import AVFoundation

class CrossingState{
    let SPEAK_UPDATE_TIME: TimeInterval = 3
    let LAST_UPDATE_THRESHOLD: TimeInterval = 10
    let CONFIRMATION_CNT: Int = 3
    let GO_REPEAT: Int = 10
```

```
var lastResult:String
var lastUpdate:Date
var lastSpoke:Date
var confirmCnt:Int

enum States{
    case initial, wait1, wait2, confirmation, go
}
var prevState: States = .initial
var currentState: States = .initial


func speak(toSay: String) {
    lastSpoke = Date()
    synthSpeak(toSay: toSay)
}

init(){
    lastResult = ""
    lastSpoke = Date()
    lastUpdate = Date()
    confirmCnt=0
}


func stateWiseUpdate(result: String){
    let nowTime = Date()
    if #available(iOS 13.0, *) {
        switch currentState{
        case .initial:
            if result == "red"{
                currentState = .wait1
                lastUpdate = nowTime

                speak(toSay: result)
                speak(toSay: "Please Wait")
                break
            }else if result == "green"{
                currentState = .wait2
                lastUpdate = nowTime

                speak(toSay: result)
                speak(toSay: "Please Wait")

                break
            }
            if(lastSpoke.distance(to: nowTime) >= SPEAK_UPDATE_TIME){
                speak(toSay: "Can't detect")
            }
            break

        case .wait1:
            if result == "green"{
                confirmCnt+=1
```

```
            if confirmCnt>=2{
                currentState = .wait2
                lastUpdate=nowTime
                speak(toSay: result)
                speak(toSay: "Please Wait")
                confirmCnt = 0
            }
            break
        }else if result == "red"{
            confirmCnt=0
            if(lastSpoke.distance(to: nowTime) >= SPEAK_UPDATE_TIME){
                speak(toSay: result)
                speak(toSay: "Please Wait")
                lastUpdate=nowTime
                break
            }
        }else{
            print("Some other detection")
        }
        if (lastUpdate.distance(to: nowTime) > LAST_UPDATE_THRESHOLD){
            currentState = .initial
        }
        break

    case .wait2:
        if result == "red"{
            confirmCnt+=1
            if confirmCnt>=2{
                currentState = .confirmation
                speak(toSay: result)
                speak(toSay: "Please Wait")
                lastUpdate=nowTime
                confirmCnt = 0
            }
            break
        }else if result == "green"{
            confirmCnt=0
            if(lastSpoke.distance(to: nowTime) >= SPEAK_UPDATE_TIME){
                speak(toSay: result)
                speak(toSay: "Please Wait")
                lastUpdate=nowTime
                break
            }
        }else{
            print("Some other detection")
        }
        if (lastUpdate.distance(to: nowTime) > LAST_UPDATE_THRESHOLD){
            currentState = .initial
        }
        break
    case .confirmation:
        if (result == "red"){
            confirmCnt+=1
            lastUpdate = nowTime
```

```
                speak(toSay: "confirming")
            }
            if confirmCnt >= CONFIRMATION_CNT{
                currentState = .go
                confirmCnt = 0
                break
            }

            if (lastUpdate.distance(to: nowTime) > LAST_UPDATE_THRESHOLD){
                currentState = .initial
            }
            break

        case .go:
            if confirmCnt < GO_REPEAT{
                if(lastSpoke.distance(to: nowTime) >= SPEAK_UPDATE_TIME){
                    speak(toSay: "Go!")
                    lastUpdate=nowTime
                    confirmCnt+=1
                }
                break
            }
        }




    } else {

    }

}

func update(result: String){
    let newTime = Date()
    if #available(iOS 13.0, *) {
        if((lastSpoke.distance(to: newTime) >= SPEAK_UPDATE_TIME) ||
          result != lastResult){
            if (result=="red" || result == "green"){
                lastResult = result
                speak(toSay: lastResult)
            }
        }

    } else {
        lastSpoke = newTime
    }
}
}
```

NavigationView.swift

```
import SwiftUI
```

```
@available(iOS 14.0, *)
struct NavigationView: View {
   @State private var isHidden: Bool = false

   var body: some View {
      VStack {
         Spacer()
         HStack {
            Spacer()
               NavigationLink(destination: Settings()) {
               VStack {
                  Text("Settings")

                  }

               }

            }
         .opacity(isHidden ? 0: 1)
      }
      .contentShape(Rectangle()) // Makes full screen tapable
      .navigationBarHidden(true)
      .gesture(TapGesture()
               .onEnded{isHidden.toggle()} )
   }
}


@available(iOS 14.0, *)
struct NaigationView_Previews: PreviewProvider {
   static var previews: some View {
      NavigationView()
   }
}
```

SettingsView.swift

```
import SwiftUI

@available(iOS 14.0, *)
struct Settings: View {
   var buttonHeight = UIScreen.main.bounds.size.height/2.75
   var buttonWidth = UIScreen.main.bounds.size.width/1.25
   @State private var shareData = false

   @AppStorage("autoDecide") var autoDecide = true

   var body: some View {
      VStack {
         List {
            Text("Decision settings")
               .font(.title)
               .padding(0.5).accessibilityHint("A menu to choose type of decision")
```

```
            Button(action: {
                print("Auto")
                autoDecide = true
                synthSpeak(toSay: "Selected Auto")

            }){Text("Auto")}.frame(width: buttonWidth, height: buttonHeight, alignment: .center).accessibilityHint("Decides and informs you automatically when to cross or wait. Say Tap auto").accessibility(label: Text("Auto Decide"))
                .accessibilityInputLabels(["auto"])



            Button(action: {
                print("Second tapped")
                autoDecide = false
                synthSpeak(toSay: "Selected Manual")

            }){Text("Manual")}.frame(width: buttonWidth, height: buttonHeight, alignment: .center).accessibilityHint("Just tells you the state of the traffic light and you make the decision").accessibility(label: Text("Manually Decide. Say Tap manual"))
                .accessibilityInputLabels(["manual","self decide"])


        }

        Spacer()


    }
    .navigationBarTitle("Settings")
  }



}

struct Settings_Previews: PreviewProvider {
  @available(iOS 13.0.0, *)
  static var previews: some View {
    if #available(iOS 14.0, *) {
        Settings()
    } else {
        // Fallback on earlier versions
    }
  }
}
```

CustomSynthesizer.swift

```
import Foundation
import AVFoundation

let synthesizer = AVSpeechSynthesizer()

func synthSpeak(toSay: String) {
  let utterance = AVSpeechUtterance(string: toSay)
```

```
    synthesizer.speak(utterance)
}
```

NotificationBeacon.swift

```
import UIKit
import CoreLocation
import UserNotifications

class ViewController: UIViewController {

    @IBOutlet weak var beaconStatusLabel: UILabel!
    @IBOutlet weak var settingsButton: UIButton!
    var locationManager: CLLocationManager = CLLocationManager()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.requestNotifications()
        // Do any additional setup after loading the view.
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(true)
        self.setUpLocationManager()
    }

    private func requestNotifications() {
        if #available(iOS 10.0, *) {
            let notificationCenter = UNUserNotificationCenter.current()
            let options: UNAuthorizationOptions = [.alert, .sound, .badge]

            notificationCenter.requestAuthorization(options: options) { (granted, error) in
                // Enable or disable features based on authorization.
                if granted {
                    print("User granted user notification!!")
                    self.scheduleNotifications(title: "Hey", messageBody: "testNotification")
                } else {
                    print("User denied user notifications!!")
                }
            }
        } else {
            // Fallback on earlier versions
            //print("control is here..")
            //self.pushLocalNotification(title: "Hey", messageBody: "testNotifications")
        }

    }


    private func pushLocalNotification(title: String, messageBody: String) {
        if #available(iOS 10.0, *) {
            UNUserNotificationCenter.current().delegate = self
            let center = UNUserNotificationCenter.current()
            center.removeAllPendingNotificationRequests()
            center.removeAllDeliveredNotifications()
```

```
        let content = UNMutableNotificationContent()
        content.title = title
        content.body = messageBody
        content.categoryIdentifier = "alarm"
        content.sound = UNNotificationSound.default

        let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1, repeats: false)

        //let request = UNNotificationRequest(identifier: UUID().uuidString, content: content, trigger: trigger)
        let notificationRequest = UNNotificationRequest(identifier: UUID().uuidString, content: content, trigger: trigger)

        // Add Request to User Notification Center
        UNUserNotificationCenter.current().add(notificationRequest) { (error) in
          if let error = error {
            print("Unable to Add Notification Request (\(error), \(error.localizedDescription))")
          }
        }
      } else {
        // Fallback on earlier versions
//        let notification = UILocalNotification()
//        notification.fireDate = NSDate(timeIntervalSinceNow: 1) as Date
//        notification.alertBody = messageBody
//        notification.alertAction = title
//        notification.soundName = UILocalNotificationDefaultSoundName
//        UIApplication.shared.scheduleLocalNotification(notification)
      }
    }

    private func scheduleNotifications(title: String, messageBody: String) {
      if #available(iOS 10.0, *) {

        UNUserNotificationCenter.current().getNotificationSettings { (settings) in
          if settings.authorizationStatus == .authorized {
            // Notifications are allowed
            self.pushLocalNotification(title: title, messageBody: messageBody)
          }
          else {
            // Either denied or notDetermined
            self.directUserToSettings(alertMessage: PushNotificationMessage)
          }
        }
      } else {
//        let isRegisteredForRemoteNotifications = UIApplication.shared.isRegisteredForRemoteNotifications
//        if isRegisteredForRemoteNotifications {
//          self.pushLocalNotification(title: title, messageBody: messageBody)
//        } else {
//          // Either denied or notDetermined
//          self.directUserToSettings(alertMessage: PushNotificationMessage)
//        }
        self.pushLocalNotification(title: title, messageBody: messageBody)
      }

    }
```

```
@IBAction func settingsButtonTapped(_ sender: UIButton) {
    self.directUserToSettings(alertMessage: AlertMessage)
}

private func setUpLocationManager() {
    self.locationManager.delegate = self
    self.locationManager.requestAlwaysAuthorization()

}

private func startScanning() {
    let uuid = UUID(uuidString: "163EB541-B100-4BA5-8652-EB0C513FB0F4")!
    let beaconRegion = CLBeaconRegion(proximityUUID: uuid, major: 123, minor: 456, identifier: "MyBeacon")
    self.locationManager.allowsBackgroundLocationUpdates = true
    self.locationManager.pausesLocationUpdatesAutomatically = false
    self.locationManager.startMonitoring(for: beaconRegion)
    self.locationManager.startRangingBeacons(in: beaconRegion)
    self.locationManager.startUpdatingLocation()

}

private func updateDistance(_ distance: CLProximity) {

    print("control in updatedistance")
    UIView.animate(withDuration: 0.8) {
        switch distance {
        case .unknown:
            self.beaconStatusLabel.text = "Unknown"
            self.scheduleNotifications(title: "Unknown", messageBody: "... ... ...")
            self.view.backgroundColor = UIColor.gray
        case .far:
            self.beaconStatusLabel.text = "Far"
            self.scheduleNotifications(title: "Far", messageBody: BeaconFar)
            self.view.backgroundColor = UIColor.blue

        case .near:
            self.beaconStatusLabel.text = "Near"
            self.scheduleNotifications(title: "Near", messageBody: BeaconNear)
            self.view.backgroundColor = UIColor.orange

        case .immediate:
            self.beaconStatusLabel.text = "Close"
            self.scheduleNotifications(title: "Too Close", messageBody: BeaconImmediate)
            self.view.backgroundColor = UIColor.red
        @unknown default:
            self.beaconStatusLabel.text = "Unknown"
            print("No matching distance found \(distance)")
        }
    }
}

private func directUserToSettings(alertMessage: String) {
```

```
        let alertController = UIAlertController (title: "Alert", message: alertMessage, preferredStyle: .alert)

        let settingsAction = UIAlertAction(title: "Settings", style: .default) { (_) -> Void in

            guard let settingsUrl = URL(string: UIApplication.openSettingsURLString) else {
                return
            }

            if UIApplication.shared.canOpenURL(settingsUrl) {
                if #available(iOS 10.0, *) {
                    UIApplication.shared.open(settingsUrl, completionHandler: { (success) in
                        print("Settings opened: \(success)") // Prints true
                    })
                } else {
                    // Fallback on earlier versions
                    UIApplication.shared.openURL(settingsUrl)
                }
            }
        }
        alertController.addAction(settingsAction)
        let cancelAction = UIAlertAction(title: "Cancel", style: .default, handler: nil)
        alertController.addAction(cancelAction)
        if presentedViewController == nil {
            self.present(alertController, animated: true, completion: nil)
        } else{
            self.dismiss(animated: false) {
                self.present(alertController, animated: true, completion: nil)
            }
        }

    }
}

extension ViewController: CLLocationManagerDelegate, UNUserNotificationCenterDelegate {

    @available(iOS 10.0, *)
    func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification,
withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void) {
        completionHandler([.alert, .badge, .sound])
    }

    @available(iOS 10.0, *)
    func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive response: UNNotificationResponse,
withCompletionHandler completionHandler: @escaping () -> Void) {
        completionHandler()
    }

    func locationManager(_ manager: CLLocationManager, didChangeAuthorization status: CLAuthorizationStatus) {

        switch status {
        case .denied:
            print("user denied the permission")
            self.directUserToSettings(alertMessage: AlertMessage)
```

```
    case .notDetermined:
        print("location permission not determined")
        self.directUserToSettings(alertMessage: AlertMessage)

    case .restricted:
        print("location permission restricted")
        self.directUserToSettings(alertMessage: AlertMessage)

    case .authorizedWhenInUse:
        print("location permission given authorizedwheninuse")
        self.directUserToSettings(alertMessage: AlertMessageForInUse)

    case .authorizedAlways:
        print("location permission granted")
        DispatchQueue.main.async { [weak self] in
            if let strongSelf = self {
                strongSelf.settingsButton.isHidden = true
                if CLLocationManager.isMonitoringAvailable(for: CLBeaconRegion.self) {
                    if CLLocationManager.isRangingAvailable() {
                        strongSelf.startScanning()
                        return
                    }
                } else {
                    print("No location found")
                }
            }
        }

    default:
        print("Nothing matched!!")
    }
    self.settingsButton.isHidden = false

}

func locationManager(_ manager: CLLocationManager, didStartMonitoringFor region: CLRegion) {
    print("The monitored regions are: \(manager.monitoredRegions)")
}

func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    let locValue:CLLocationCoordinate2D = manager.location!.coordinate
    print("locations = \(locValue.latitude) \(locValue.longitude)")
}

func locationManager(_ manager: CLLocationManager, didDetermineState state: CLRegionState, for region: CLRegion) {
    print("control inside state")
}

func locationManager(_ manager: CLLocationManager, didEnterRegion region: CLRegion) {
    manager.startRangingBeacons(in: region as! CLBeaconRegion)
    manager.startUpdatingLocation()
    print("control in didEnterRegion")
```

```
    }

    func locationManager(_ manager: CLLocationManager, didExitRegion region: CLRegion) {
        manager.stopRangingBeacons(in: region as! CLBeaconRegion)
        manager.stopUpdatingLocation()
        print("control in didExitRegion")

    }

    func locationManager(_ manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], in region:
CLBeaconRegion) {

        print("control in didrangebeacons \(beacons.count)")

        if beacons.count > 0 {
            updateDistance(beacons[0].proximity)
        } else {
            updateDistance(.unknown)
        }
    }

}
```