Final Year Project 2015/16

Smartphone Based Vehicle Recorder

Appy Driving

Final Report

Student: Chung Man Ip, Clement

Supervisor: Dr H.Y. Chung, Ronald

Date: Apr 17, 2016

Table of Contents

Project Background

Introduction

Vehicle Recorders, also known as Dash Cams, are becoming more and more popular in the Auto Accessories Industry. Currently, there is already a wide variety of devices existing on the market, however, most, if not all, of them has really similar characteristics such as having a camera which supports looped recording, with an external storage card as its primary storage. Although the installation cost for a vehicle recorder is not cheap, there are great reasons for investing in this gadget. For instance, the video footage captured can be used as video evidence in cases of claiming accident insurance, or even proof of innocence in legal inquiries. This provides a solid proof other than words from the driver, victim, or other witnesses, and is also a good explanation for the fact that vehicle recorders are ubiquitous in Russia, which has recorded the highest number of car accidents and fatalities in 2009[1].

With the unstoppable advancement in the Smartphone Industry, both the hardware and software capabilities of them improve rapidly in recent years. Some developers have started to implement camera-related apps to bring the vehicle recorder features into smartphones. They made use of the built-in camera and internal storage of the phone to virtualize a vehicle recorder in the driver's smartphone, making the installation of vehicle cameras much easier and cheaper. However, a simple camera application does not fully utilize the true power of smartphones, especially in regards with the aspect of social networks.

In this information era, smartphones are actually a portable hotspot for data sharing through the Internet. Push notifications and social networks have established various channels to send and receive information via the Internet. On the other hand, hardware capabilities offer a better user experience as well as value-added functions compared with a conventional vehicle recorder. For instance, with the data gathered

[1] http://www.forbes.com/2009/05/19/dangerous-countries-roads-lifestyle-travel-dangerous-roads_slide_11.html

by an accelerometer and Global Positioning System (GPS), mobile applications could provide extra useful information such as car speed and current position.
The aim for this project is to design and develop a Smartphone Vehicle Recorder, with add-on features that are designed for general drivers.

Conventional vehicle recorder

Vehicle recorders were introduced to police cars in Texas in the 1980s, at that time, the primary function was to capture everything ahead of the vehicle. With the fading out of VHS cassettes, the size of the vehicle recorder has become much smaller so that installation became much easier as well. In the 2000's, vehicle recorders entered the domestic auto market, getting more and more common among general drivers. Drivers who look for extra protection would invest in this gadget in order to guarantee that there is a video footage of what happened when there are unfortunate cases of traffic accident.

Variations among different manufacturers of entry-level vehicle recorders are just mainly between camera quality and size of the recorder[2]; with some suggesting a better night view camera but with relatively larger body, while the others offer a stealth-looking recorder with a smaller size so that it will not block the view of the drivers. As regular recorders, those with a camera capturing movements ahead and storing footages in their internal storages, has become saturated in the market, manufacturers start to implement more value-adding functions to their products. In an effort to differentiate themselves, products with multiple cameras capturing front and back views, and products equipped with GPS or G-Sensor to support geotagging and motion detection start to appear on the market.

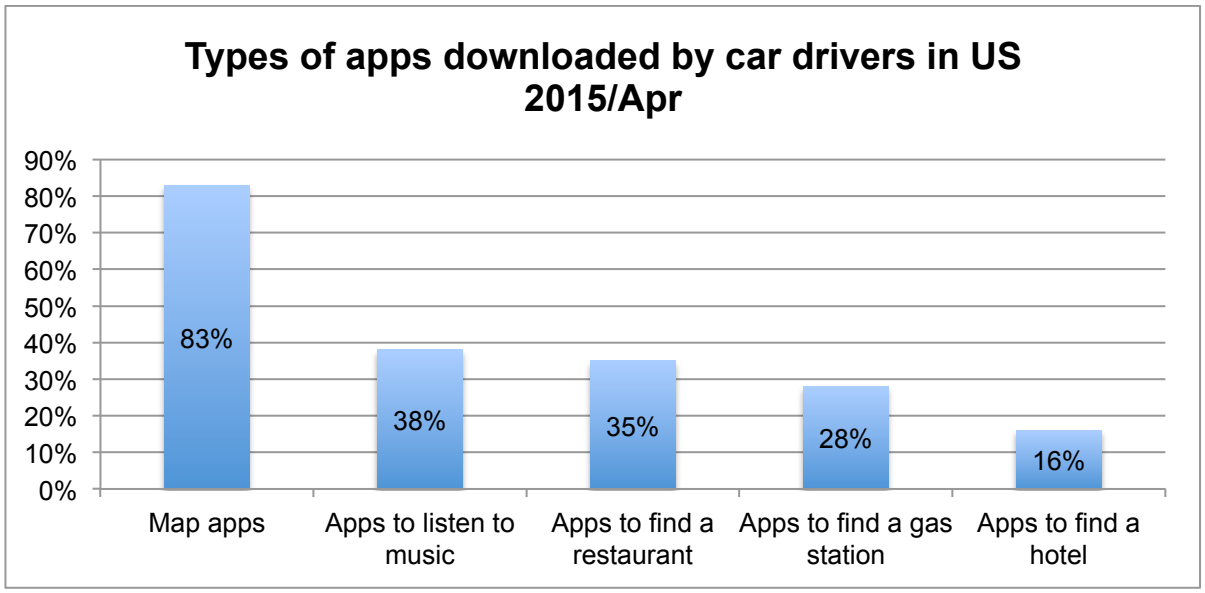
It is troublesome to configure the camera settings by just observing the LED light indicator, or to fine-tune the camera to the desired angle without a preview display,

[2] http://dashboardcamerareviews.com/dash-cam-comparison-table/dash-cam-comparison-affordable/

hence, manufacturers have started to product products which come with a small LCD panel. This design allows the drivers to check whether the camera is really working during road trips, thus having a peace of mind that every moment is being captured.

Current Market Analysis

Of over one and a half million applications available on the App Store, applications related to vehicle recording can be categorized into two main streams- one being an individual application that offers a Dash Cam function, while the other is developed to work cooperatively with a physical vehicle camera. However, only a few of the applications could leverage the true power of smartphones, such as network accessibility and multi-tasking capability, to bring added values for the users.



According to a recent research shown above[3], most drivers in the United States are looking for smartphone applications that offer a map function, while less popular functions are music-playing and place-finding. This research indicates that drivers do want to utilize their smartphones to assist themselves with their driving, hence providing additional information to them.

[3] http://www.statista.com/statistics/428070/types-of-apps-downloaded-by-car-drivers-us/

To conclude, there is clearly a gap between existing products currently on the market and user expectations. Undoubtedly, standalone applications are sufficient to fulfill the need of a particular function, but the needs may vary based on different locations and time factors. In order to bridge the gap between the existing market and user expectations, it would be advantageous to have an application that can satisfy various drivers' needs regardless of where and when they are, and utilize the full capabilities of smartphones to enhance the user experience.

Project Objective

"Appy Driving" – the name of the application of this project, aims to offer an all-in-one platform that includes multiple functions to satisfy the general drivers' needs. Various functions will be delivered as modules that run on top of the core framework. This idea of modular design is similar to relationship between applications and operating system in modern computers. Likewise, there will be multiple applications running on the smartphone platform; hence the name "Appy" literally means functions with modular design like applications installed on smartphones.

Features

1. User Customization

Users can organize their own application layout according to their preferences. Each module can be independently shown or hidden, and display in whatever order within the core framework. Instead of utilizing a view stack provided by Navigation Controller, that allows user go back and forth within the view hierarchy. Appy adopts a brand new resign pattern which enables multiple view controllers on one view container, a scroll view container, to navigate among different modules. If some of the modules will not be used for a long period of time, users can simply move them into disabled modules in Settings, this does not affect the use of other modules.

2. Optimal Video Quality

For an accident video footage captured in just a couple of seconds, drivers would want to capture every single pixel as clear as possible, in order to retain a strong proof as an evidence. However, the higher video quality the more space consumed, and the other applications' service would be affected if the video files use up all the storage space. Appy will automatically adjust the best video quality according to the hardware capabilities of the users' phone. Users are able to control the time interval of loop recording, so that can achieve a balance between storage usage and video quality.

3. Tracking and Social Connectivity

As driving holidays are getting popular these days, capture the moment of travelling across countries is another interesting usage of Appy. Driving is not alone anymore with social features implemented. Users can share their video footages and routes with friends or even other drivers, discuss on the experiences encountered on roads and highways. This sharing platform gathers comments from drivers, thus ultimately contributes to the resources base of this application. For instance, users can suggest travel routes to others, if the other drivers find it useful, they can up vote the content as well as the contributor.

Core Functions

The table below briefly describes each module and its functions:

Modules	Functions
Core Framework	• Multi-modules Available • User Customization
Vehicle Camera	• Looped Recording • Read-only Flags • Adjustable Quality and Brightness
Navigation	• Way-finding
Traffic News	• Instant Traffic Snapshot
Social *(future)*	• Drives Tracking • Share Videos/Routes

Project Methodology

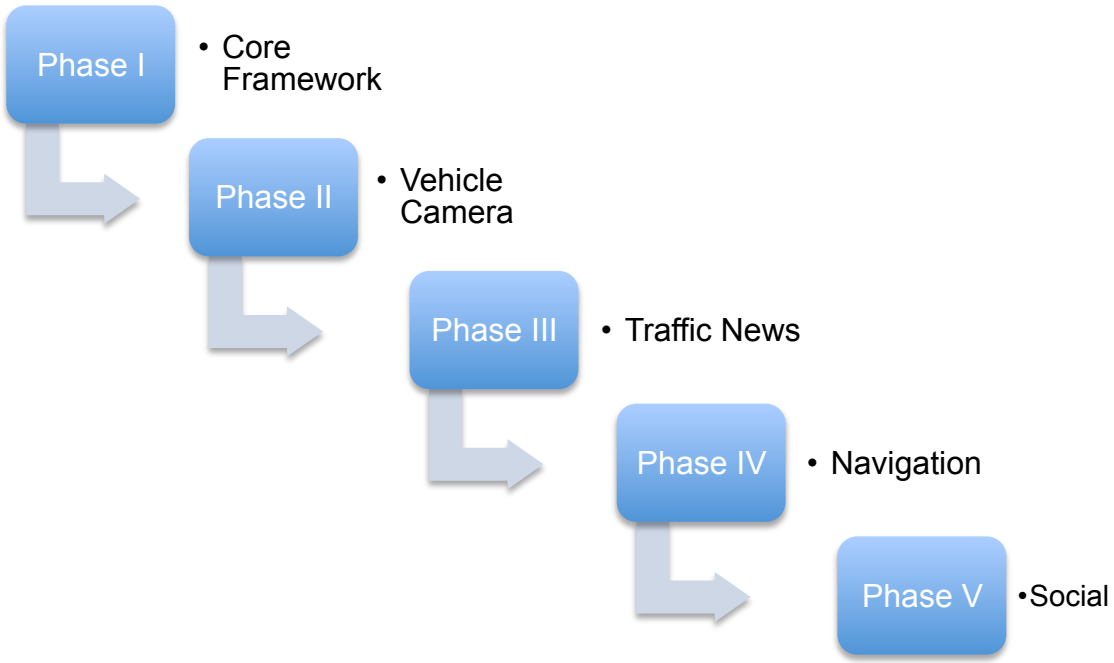
Implementation

The whole project consists of a front-end mobile application as well as a back-end server. For the front-end mobile application, it would include basic vehicle camera functions and value-added functions with modular design in order to enhance the overall user experience. In the smartphone industry, iOS and Android are the two most common operating systems, with the number of applications available in the App Store on par with the Play Store. This project will be built on the iOS platform targeting iOS 8.0 first, due to lesser variations regarding both hardware and software aspects, so that the quality and user experience controls would be easier to manage. The development of the application for Android platform would be discussed in the future development.

For the back-end server, it would be responsible for the content management service of the application, and also as a database for the primary storage of information. Regarding the development platform, Ruby on Rails would be a desirable candidate in terms of its scalability and flexibility, as it offers advantages both in the database management and migration. There will be two environments, development and production respectively, for the deployment of the back-end server. Both environments will be almost the same with regards to the design, structure and functions. However, the development environment will be mainly for internal testing only, while the production environment will be solely for public usage. This practice can stabilize the deployment since the external database is being isolated with the internal one, hence the public users' data will be safe even if there is a critical failure encountered in the testing stages.

Project Management

Since the application is based on a modular design, the whole project will adopt the Phased Methodology in order to design and implement the system. The development cycle will be categorized into five phases, with the earlier phases focusing on the core framework and basic functions such as working as a vehicle recorder while the latter phases would be value-adding add-ons built on top of the core framework. Each module can be tested independently and thus guarantee the quality of the deliverables.



Project Schedule and Milestone

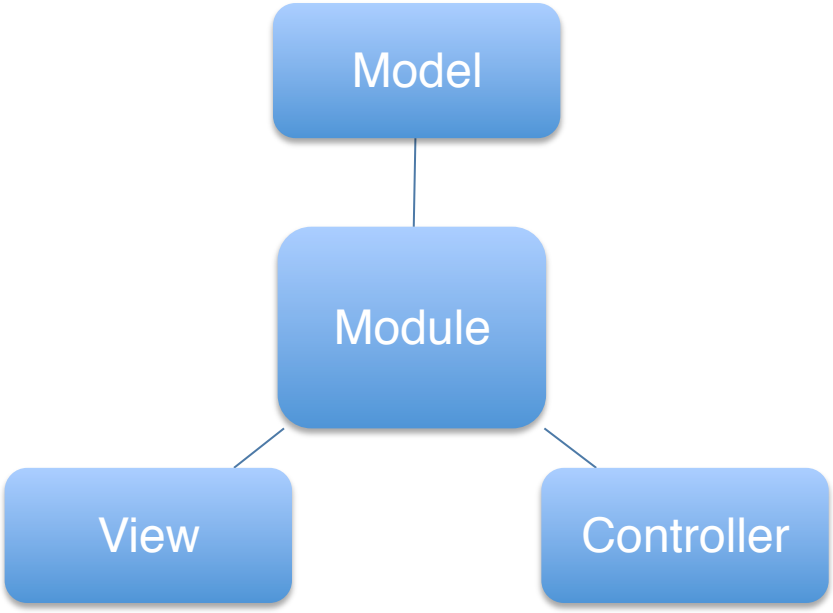
Tasks	Pre-decessor	Expected Start Date	Expected End Date	Duration (days)	Milestone
A: Meeting with supervisor Preliminary requirements gathering	-	Sep 14 2015	Sep 24 2015	10	
B: Finalize scope of project	A	Sep 24 2015	Oct 4 2015	11	Project Plan Project Webpage
C: Phase I Coding and Testing	A	Oct 4 2015	Oct 31 2015	28	
D: Phase II Coding and Testing	C	Nov 1 2015	Nov 30 2015	30	
E: Examination Break	-	Dec 1 2015	Dec 23 2015	23	
F: Phase III Coding and Testing	D	Dec 23 2015	Jan 18 2016	27	
G: Interim Testing and Evaluation	D, F	Jan 18 2016	Jan 24 2016	7	Interim Report Product Prototype
H: Phase IV Coding and Testing	F	Jan 18 2016	Feb 25 2016	39	
I: Final Testing and Evaluation	H, I	Apr 1 2016	Apr 17 2016	17	Final Report Finalized Product
J: Project Exhibition and Presentation Preparation Documentation	J	Apr 18 2016	May 3 2016	16	Project Exhibition Materials
K: Phase V Coding and Testing	H	Feb 25 2016	Mar 31 2016	36	

Implementation Detail

Phase I – Core Framework

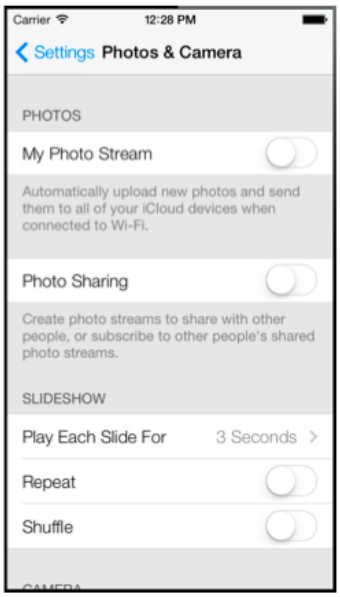
In this phase, the objective is to implement a back-end server as well as design and create the core framework of application itself. For the former part, a server that running Ruby on Rails was set up via the department of Computer Science, HKU. The system and database schema was initialized and ready for phase V development in the near future. On the other hand, UI design was done and implemented as a core framework of the application.

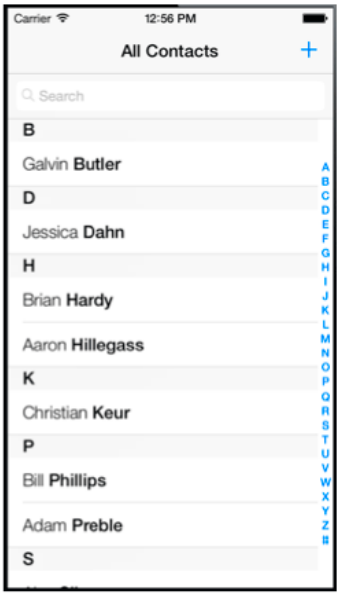
Instead of using traditional Navigation Controller model with several segues pointing to other Model-View-Controller (MVC) instances, Appy adopts separate MVC structure in order to enhance the user experience and maintenance. With isolated MVC instances, users are able to navigate different functions by simple swipes. Moreover, it is more maintainable in terms of testing and maintenance as a result of separate MVC with minimal amount of dependencies. Each MVC instance contains a storyboard instance as view, *.swift* file as view controller and other data formats such as Core Data in iOS as model.



For each MVC instance, each consists of three main domains, Model, View and Controller. Model Objects represent the data source of module, this could be a media file, routes, or even a MVC instance, but not limited to the above. In some complex case of model object, relational database would be implemented by Core Data framework. Model should not have connections with View, in order to ensure a strong encapsulation.

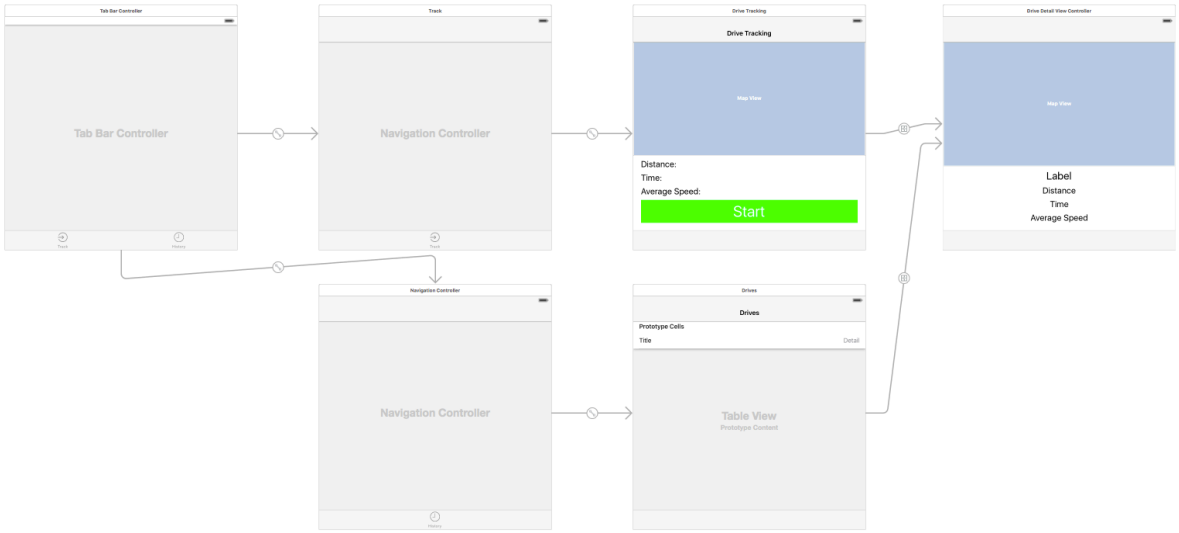
View Objects purely represents the instances that is visible to user. For instance, the image view that contains the image, image itself is regarded as Model but the view that contains it will be regarded as View. A major purpose of view objects is to display data from Model Objects properly, and enable editing by providing input text fields or buttons. In iOS development, view objects are usually reusable and reconfigurable as they are independent from the content actually. This approach could reserve resources and reduce the time of re-generating loads of view objects.





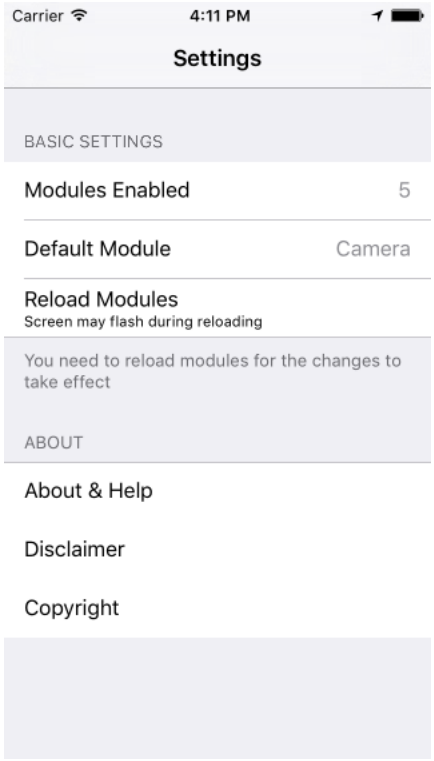
When the user scrolls down the table view, actually there is not any new cells instantiated. Instead, scrolling down the table would make the upper cell become invisible to user, then the system will do recycle job by collecting back the upper cell objects, renew the content to be displayed, add to bottom of table view when it lastly becomes visible to user.

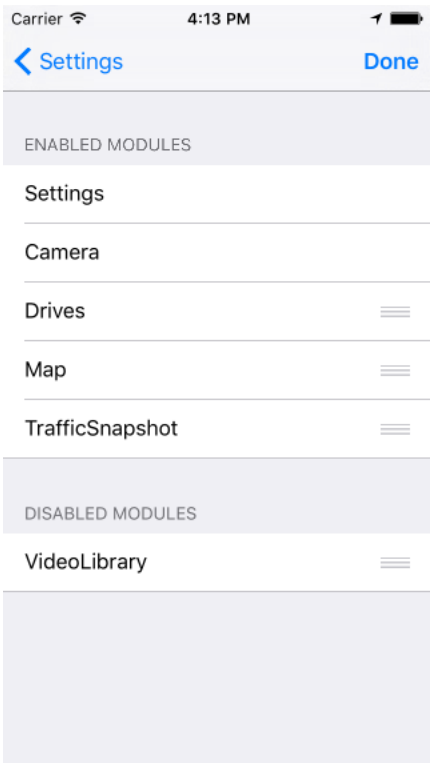
Last but not least, Controller Objects acts as a bridge between Model Objects and View Objects. A controller determine which and how data to be displayed on screen by using what view containers, for example, an instance of UITableViewController determine the number of sections and number of rows to be displayed, also it will control which attributes of data to be shown in a single cell. Another major role of Controller is that to listen user inputs on view objects, within the same example as above, if users select a cell it will notify the controller that a certain cell was selected, then the controller instance will provide feedback by fetching model objects or some logic computations, and ultimately display the results on view objects. Controller itself is not limited to its own MVC structure, it could also direct the user to another MVC structure by sending segues or adding to view hierarchy.

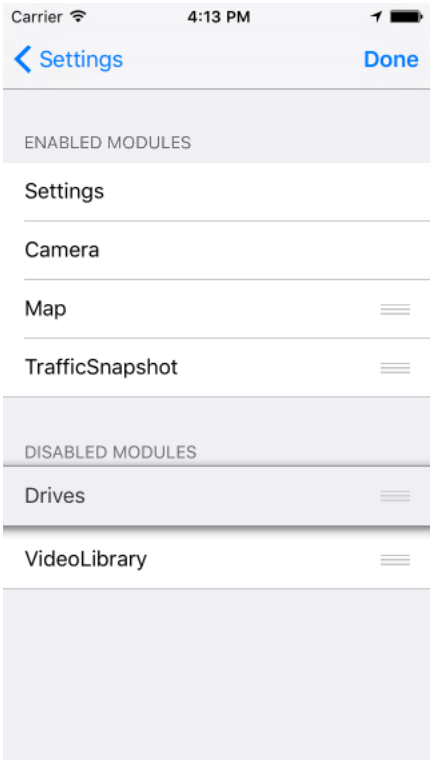


As the figure shows, there are arrows representing segues between MVC instances.

Separate MVC structure on top of core framework delivers a more organized way of modules; users can choose which to stay and which to hide according to preferences, as well as performance/battery consideration. In Settings page, Appy implemented a swipe to select gesture to have a greater extent in terms of user experience. The demonstrations are as follows:







Phase II - Camera

Stepping into second phase of deployment, the main function in this phase is to implement the camera recording function. In order to make use of built-in camera devices of Apple devices, there are two classes available in Swift 2.0 to enable the vision: UIImagePickerController and AVCaptureSession. UIImagePickerController manages user interfaces for taking images and movies, and for choosing the media from user library (i.e. Camera Roll in Apple devices). Comparatively, AVCaptureSession is just an interface to coordinate the flow of data from AV input sources to outputs, without the media picking function and built-in camera function like HDR and flash control. General users may be more familiar with former interface as it is the same as iOS built-in camera app, but Appy camera is implemented according to the AVCaptureSession as it suggests a much more powerful camera interface than UIImagePickerController.

Initializing AVCaptureSession

In AVCaptureSession, input devices available to the class are front and rear camera, microphones, etc. In order to perform a real-time image taking or video shooting, after creating an instance of AVCaptureSession the application requires handling various input devices and adding them into the current session. After that, configure appropriate output format and preset to manage the quality as well as output format. The quality of video will be adjusted automatically according to the preferred space set by the user, to achieve this several video preset are used as follows:

sessionPreset Properties

AVCaptureSessionPreset3840x2160	Suitable for extremely definition video output
AVCaptureSessionPresetHigh	Suitable for high definition video output
AVCaptureSessionPresetMedium	Suitable for sharing over WiFi
AVCaptureSessionPresetLow	Suitable for sharing over legacy cellular network
AVCaptureSessionPresetPhoto	Suitable for high resolution photo output

Finishing Set Up Session

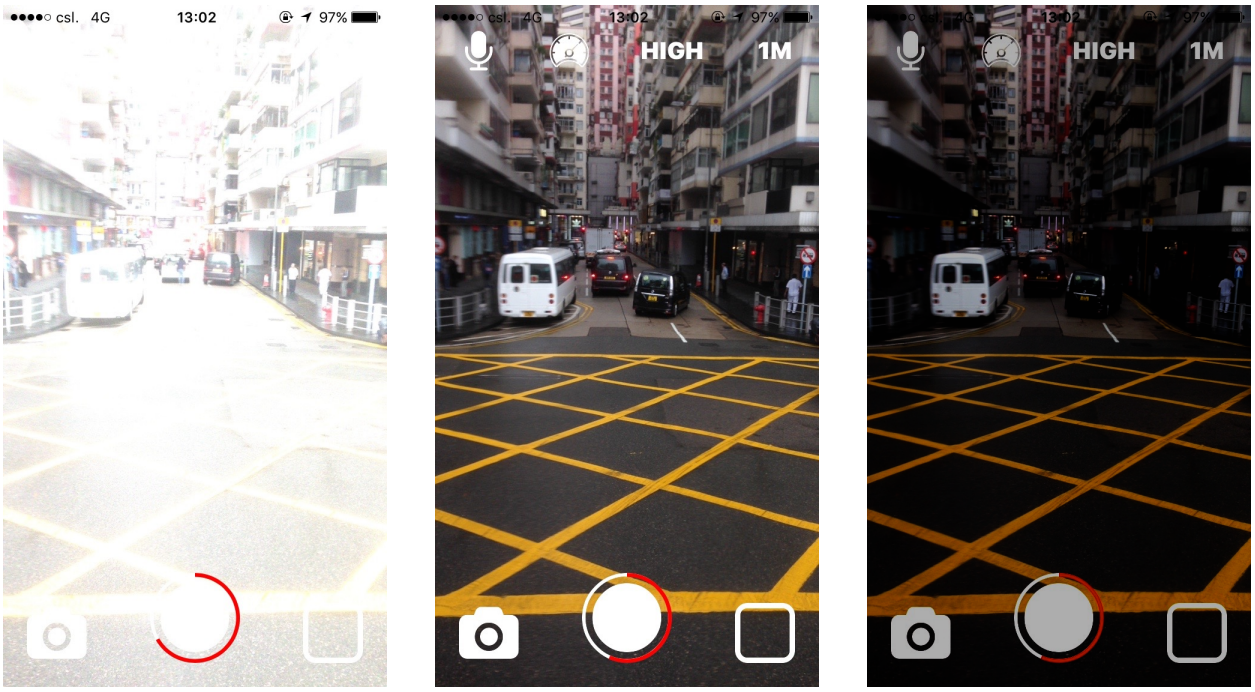
After setting up all the inputs, outpus and session presets, it is time to begin the AVCaptureSession to start the flow of data. It is absolutely a good feature if the user can see what the camera sees, so it is also required to set up a preview layer on top of the view hierarchy after setting up the AVCaptureSession.

Screenshots of camera preview layer

Users can directly know what the camera input is and make further adjustment according to the result. Moreover, having a preview layer can also suggest the camera is working properly without any error while recording. Since users cannot put much focus on the application during driving, the preview layer acts as a monitor that shows what would be included in the video footage.

Brightness Control

More importantly, there is a huge contrast of brightness in daytime and night. What if the users do not want to solely rely on the automatic settings of built-in camera? Appy introduces a pan gesture recognizer on top of preview layer to act as a brightness slider. So simply swipe upward or downward the brightness of camera can then be adjusted, simplify the actions that may require few more steps. In order to achieve this, firstly it has to add a UIPanGestureRecognizer object to listen the vertical gesture on layer, and then multiply a certain scale factor to adjust the brightness value.



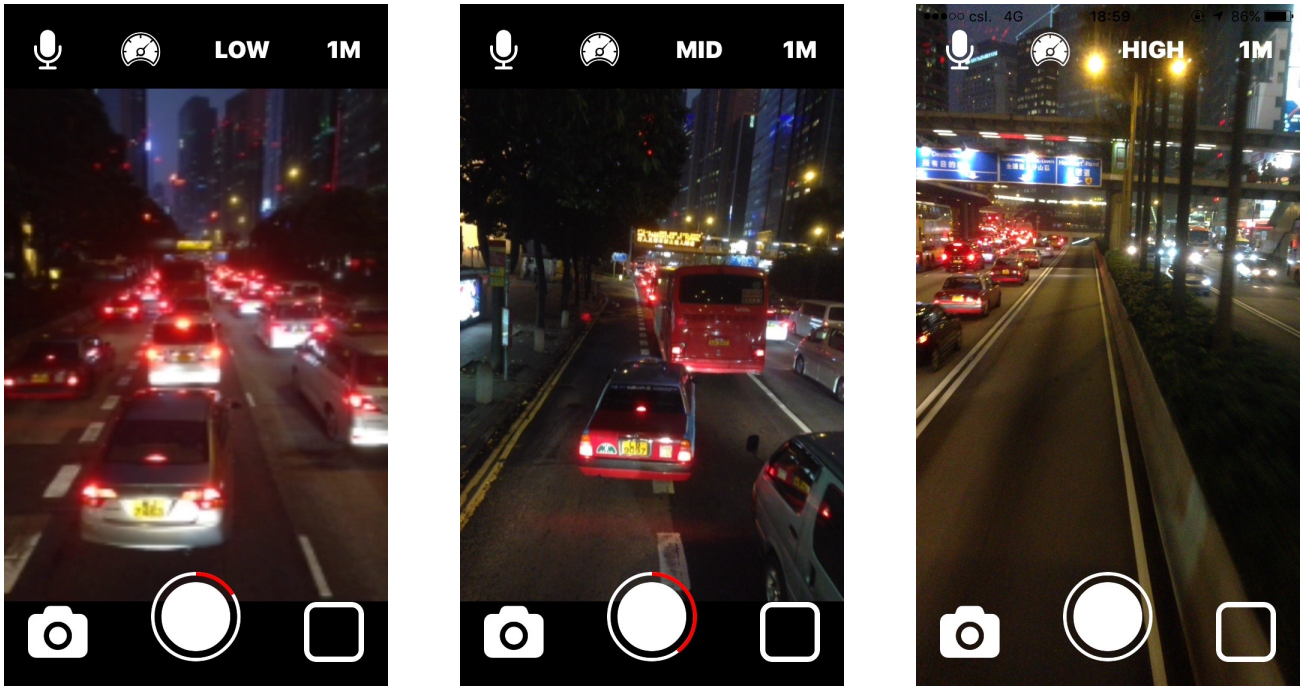
Screenshots of different brightness values

To change brightness control back to automatic mode, just double tap on the screen and the brightness value will be automatically updated. Adding gesture recognizers to listen what user inputs would offer a better interactive between the application and users, also suggest a user-friendly control manner in terms of user experience.

Camera Quality Control

Appy default sets the session preset to high, though users can toggle between various presets mentioned before. As the trend of iPhone storage capacity getting larger to enormous nowadays, this setting may not affect every user. However, imagine that someone using his/her legacy iPhone 4s or 5 to be a vehicle recorder, lowering the quality setting would definitely ease the resources pressure as well as storage consumption.

In iOS 9, Apple announced the latest 4K camera featured in latest model of iPhones, Appy also suggest 4K recording if the device support such high resolution. Users can capture every spectacular moment during driving when setting the ultra high quality, even though the battery and storage consumption will ramp up a lot compared to normal 1920x1080 full HD recording.



Screenshots of switching quality on iPhone 5

Capture Session Delegate

Apple iOS uses delegate method to send notifications of certain actions have been done or is going to be done. AVCaptureSession also has its own delegate class to do listening jobs on capturing events. For instance, the implementation of loop recordings require the delegate method to tell whether a video has been successfully written to persistent storage or not, and when to start next recording, etc.
Since writing files, especially video files has large file size, it is regarded as risky operations due to I/O performance. Listening to delegate methods can minimize the risk of data lost or system crash as a result of waiting completion of previous I/O operation.

Video Library Associated

In the AVCaptureSession implemented in Appy, the footages are directly write to persistent storage under the application sandbox rather than saving them to Camera Roll, i.e. the default media manager built-in by iOS. Having this approach because Appy may further process the footages to route tracked, or social platform to share with others. In this case, it is necessary to build a video library on scratches for reviewing the footages.

The video library consists of three MVC structure and an entity in Core Data. They are a UITableViewController as the video library, a UIViewController as the detail view of video and a UITableViewCell as the cell to represent a video.

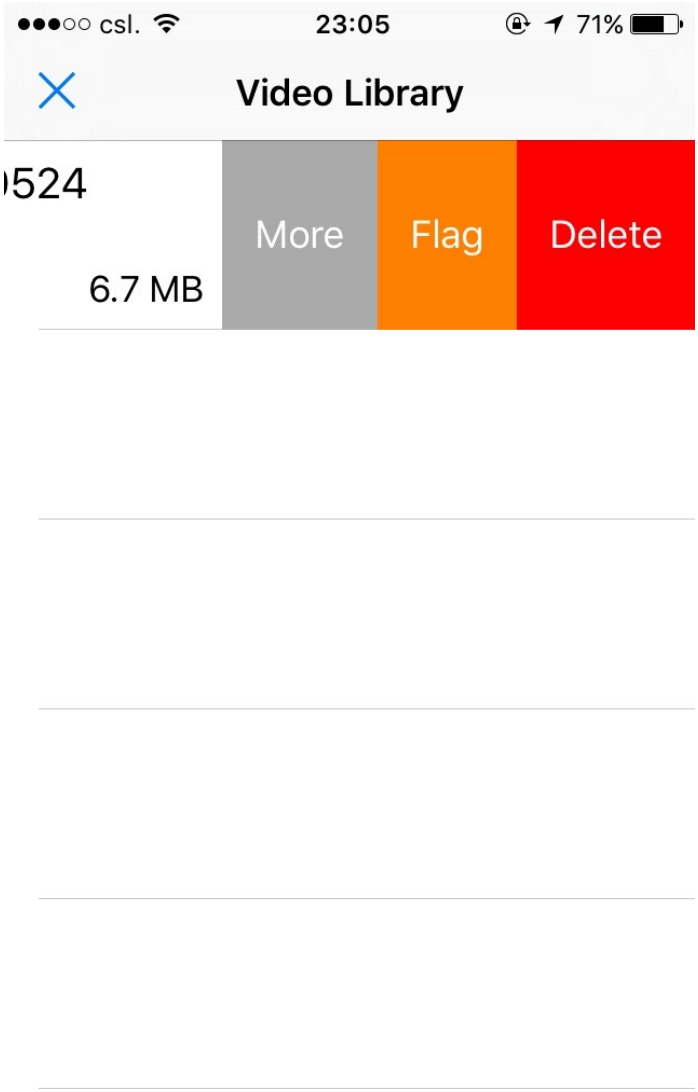
Video Library – UITableViewController

In this table view controller, the data source and delegate are integrated to one single file for easy maintenance. The data source is fetched from Core Data using the AppDelegate, and return an array of file names as identifier. Originally this was done by returning the file list of directory, this is a less complex manner to manage a list of video that stores under same directory. However, the old approach will miss the functions of having relationship among other entities and the multi-attributed

content of certain video. Therefore, in the latter part of implementation, the approach was migrated to Core Data and mount up relationship with drives tracked.

Video Library Cell – UITableViewCell

The standard table cells only support text display and a small thumbnail, but this is not big enough to see the detail of video. Thus, the table used a custom cell rather than system defaults, this is due to the support of larger thumbnail and more than two text labels available for each cell, and hence, there is another UITableViewCell class manages it.



Each cell has added three edit actions to perform actions like flag, show detail and delete, if a video is being flagged then the delete function is hidden to protect user accidentally delete it.

Video Detail View Controller - UIViewController

When the users press “More”, it will perform segue to the video detail view controller. This controller is the last piece of puzzles of entire video library implementation, it shows the video in a full screen manner and allows user to rename the video as they wish.

Phase III – Realtime Traffic Snapshots

In this phase, the main functionality is to provide the real time traffic situation in Hong Kong. At the beginning, this idea was related to RSS feeding from Traffic Department of Hong Kong, and then parse the content to a UISplitViewController. This approach would enable user to input their own RSS feed to retrieve different source of traffic news. However, RSS feed only support text-only news generally, populate a table with text-only would not be a user-friendly way to do so. Based on such concerns, the idea was switched to obtain real-time snapshot provided by Traffic Department of Hong Kong.

Data Fetching

There are real-time snapshots from 183 camera located in various region in Hong Kong, they are available to public in the official website of Traffic Department, as well as Data-One government database. The data fetching process consists of three steps:

1. XML Parsing

To start with, Appy needs to identify the latest URL of image and camera location. Such information could be accessed from an XML configuration file on Data-One

database, http://data.one.gov.hk/code/td/imagelist.xml, which includes all the camera identifier and camera location.

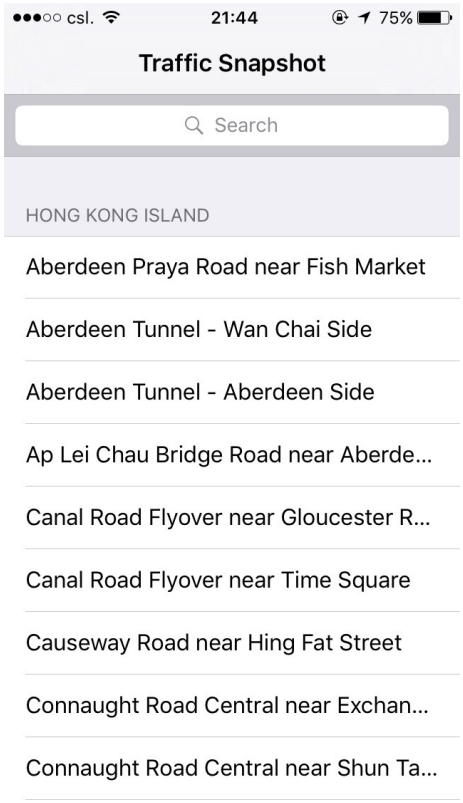
<image>
<key>H421F</key>
<english-region>Hong Kong Island</english-region>
<chinese-region>香港島</chinese-region>
<english-description>Aberdeen Tunnel - Aberdeen Side</english-description>
<chinese-description>香港仔隧道香港仔入口</chinese-description>
</image>
Sample of Camera Infromation

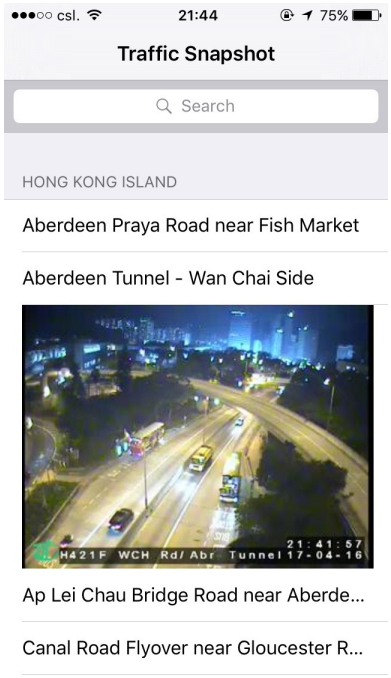
This action requires the NSXMLParser class and its delegate class to fetch XML file into useful array of information.
After the parse completed, the information will be stored as an array of dictionaries.

2. Table View Populating

Here comes to step 2, after fetching the XML configuration file now it has a dictionary of data specifies the camera identifier and location, thus populating a table view is feasible using such information.

The table view is multi-sections according to the number of region from data source, and each section will display the camera location within that particular region.





Delegate methods listen to the tap action on cell, if the cell is being tapped the image fetching process would undergo. Moreover, the row height property of cell would increase in order to fit the image fetched.

UISearchController enables the indexed searching within the table, the search bar is hidden under the navigation controller. When user want to search a specific camera, type the query and the table view will automatically update the matched results.

3. Image Fetching

The cell that pressed would pass the identifier of that camera to image fetching method, this method take camera identifier as an argument and return a UIImage to caller. Noted that the internet connections may not be stable using cellular network, this method will be ran using Grand Central Dispatch (G.C.D.) context-switching approach to do multi-tasking.

The main queue is responsible for view reaction and to ensure the application is being responsive all the time. According to Apple User Experience Guideline, time-consuming task shall not use the main queue otherwise the application responsiveness would dramatically decrease.

Fetching image would be regarded as User-Initiated action as it is also vital for ensuring a good user experience. After fetching the UIImage, the UI update method will be call back on the main queue and therefore the image would be shown on screen.

Phase IV – Navigation

In this Navigation Phase, the deliverable is a way-finding kit for driver. Several applications offering similar functions are already exist in App Store, for instance, Google Map, AutoNavi, or even the built-in Apple Map. All of them provide very good navigation and detail of map, but Appy also offer some new functions to drivers seeking for the best path.

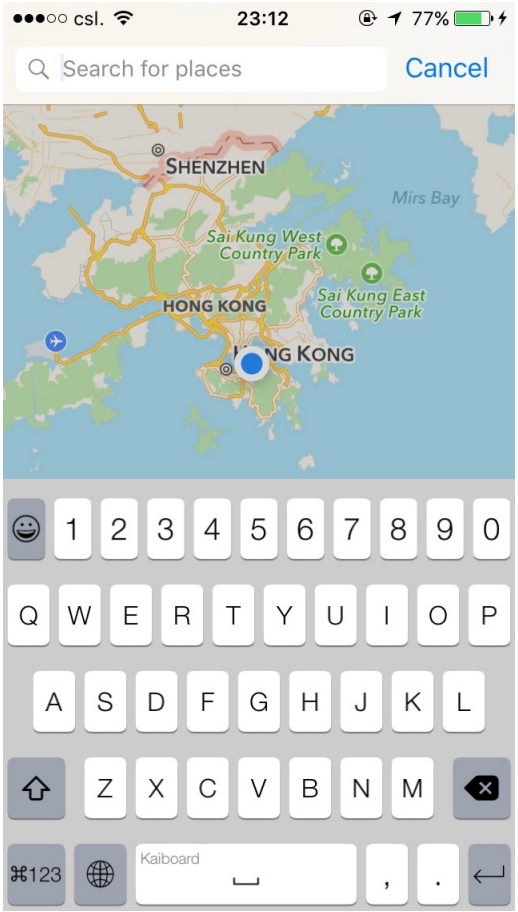
The implementation of this part uses the MapKit framework provided by Apple, the reason of not choosing third-party map framework is that the inconsistency between user interface and bridging methods. Similarly, there are two MVC structure were used to populate a map view and a searching table.

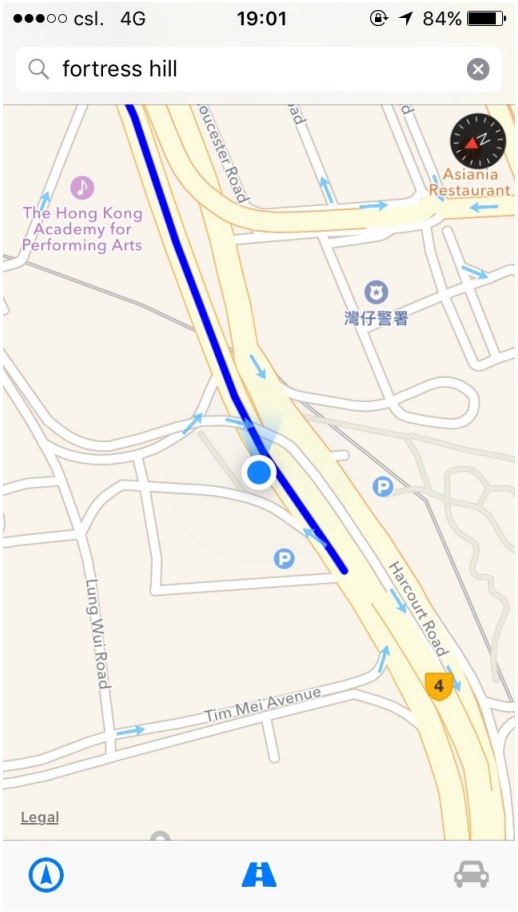
MapViewController – UIViewController import MapKit

This is a basic view controller that contains a MapView on top of it, and confront to the MKMapViewDelegate protocol to listen for map view notifications.

The map view also listens to a gesuture recognizer LongPressGestureRecognizer to enable user to add its own waypoint without searching for an exact place.

On top of that, it also confront to a self-defined protocol HandleMapSearch. This will active when user select a location from the populated search result table view.

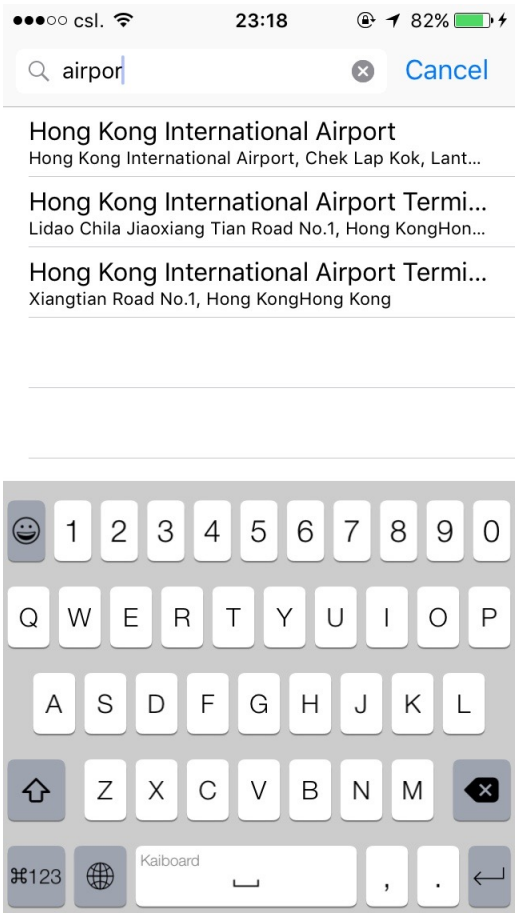


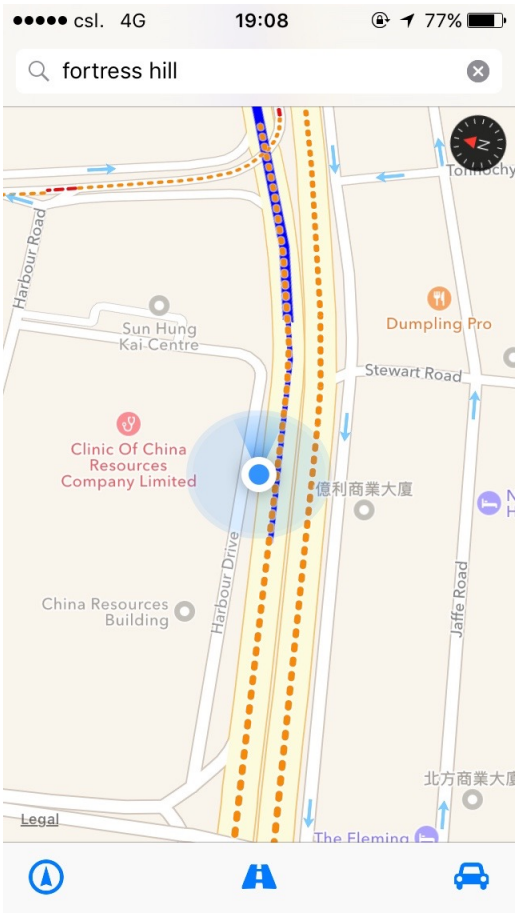


When there is a waypoint selected, regardless from long pressing the map or select from search result table, it will create an annotation on map view and display the corresponding information regarding the location selected. The most important part of this phase, is to calculate the best route for drivers, so there is a small button with a vehicle icon. Trivially, pressing that car button will process the route calculation method, when there is a possible result it will then automatically draw on the map, and set the GPS tracking stick to the user current location with heading support.

But it could be understand that sometimes driver may accidentally run into the wrong way, and all of the settings have to be redo once again. But using Appy navigation module this would not be a problem at all. Appy will automatically draw a new route for users if they drive too far from the highlighted route, drivers so that can remain

focus on their driving without worrying to set up a new location direction from current user location. On top of that, it also confront to a self-defined protocol HandleMapSearch. This will active when user selects a location from the populated search result table view.





With the latest iOS 9, the map can also reflect traffic situation on map.

Phase V – Social Connectivity

This is the last phase of the entire project, however, unfortunately this part is yet to be finished in near future. Social connectivity is a hot trend in the internet world, everyone can interact with other users via internet on social platform, this social connectivity also enhance loads of media sharing among the huge internet world. The idea was that Appy allows user to share their route and footage via the backend server and some online video sharing platform, then users can comment and share their drive like a blog.

Therefore, in this stage the implementation has to make route tracking possible beforehand. The term Tracking was introduced with the rise popularity of health products recently, like Apple Watch allows user to track their daily health issue. Imagine that having a wearable for land vehicle, it tracks the drive and compute the average speed and speed analysis to user, providing more information other than just a point to point route.

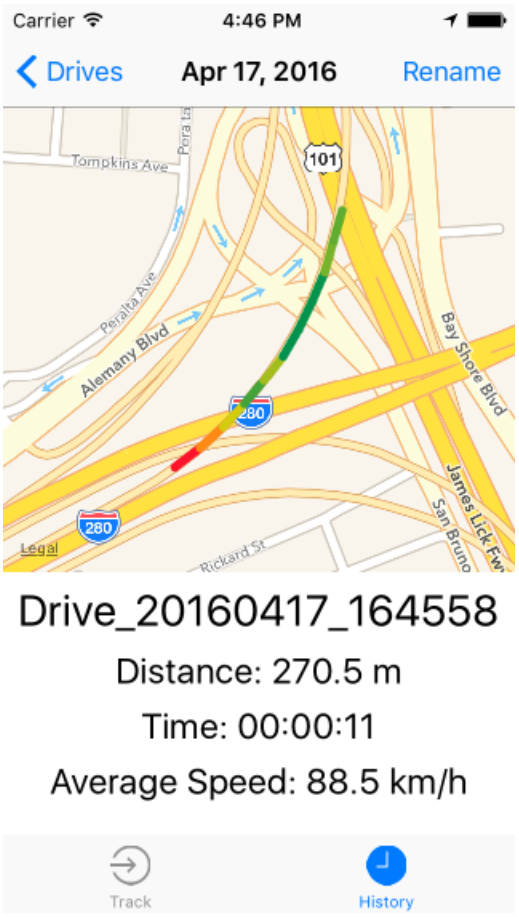
The tracking development consists of four separate classes, and of course an entity stored in Core Data. There are three MVC structures, a new tracking page, a library represents a list of tracking and a detailed view of route respectively. Lastly, there is one class to perform speed analysis and render a speed graph on map.

AddDriveViewController – UIViewController

This class is used to start tracking from current user location, keep tracking activities until the users press stop tracking. The location information is retrieved from Core Location framework, a CLLocationManager instance to provide real-time location. Moreover, the location is filtered by horizontal accuracy in order to obtain the most accurate information under an acceptable tolerance error.

When there is a new location data processed, the tracking page will calculate the distance travelled and log the timestamp of each location data, once all the data recorded to Core Data database, the analysis part will take part of each location data to see the speed distribution of that drive.

For the speed analysis, it will uses three color to represent the speed categories, with green is the fastest, yellow the middle and red the slowest. By having an analysis of different routes, users can find out which path is more likely to have traffic jam while the others don't.



Design Pattern

One of the drawback of separated MVC structures is that some core services may be instantiated twice or more in order to fulfil different needs from MVC structure. For instance, the Core Location service instance CLLoactionManager helps to retrieve the user current location from GPS and A-GPS. In Appy, both the tracking part and navigation part requires the use of location data, that would possibly mess up by creating two location manager instances, and report to different controller independently. As a result, there is drop of performance and waste of system resources if objects are duplicated, and also it is difficult for developers to trace back errors, resulting in a time-consuming maintenance required.

Avoid such situation from happening, the development of Appy adopted a popular design pattern, Singleton, as the solution to duplicate instances. Singleton pattern restricts the number of instance of a class to one object only, which is a global variable among whole project. It usually uses lazy declaration to create the shared instance only when it is needed, other classes or instances are not allowed to instantiate object from that class.

Based on this design pattern, the class itself needs to instantiate itself as a global class variable, and makes its initiator become private function so that other classes cannot instantiate instances from it. Moreover, the class requires to set up public APIs for that single shared instance to be called. The CLLocationManager and NSFileManager classes in Appy are implemented in Singleton pattern, as the read and write operations to sandbox and current location data are usually required among all the MVC structures.

Source Control

Source control, or version control, is a system that records changes to a set of files. In this case, the entire project is under monitoring by version control system by using Git protocol. Having source control in the project definitely eases the burden of maintenance, especially when attempting to add new features to existing classes. If one of the newly added function critically crashed the entire project, rather than doing

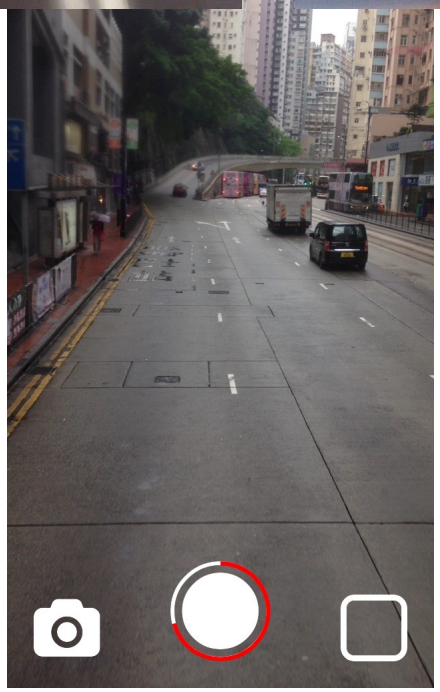
undo for thousands time, a source control enabled project can revert to stable version swiftly. Detailed historical information on files can act as an automatic backup of project with versioning supported, also making branching far away from troublesome.

Product Testing

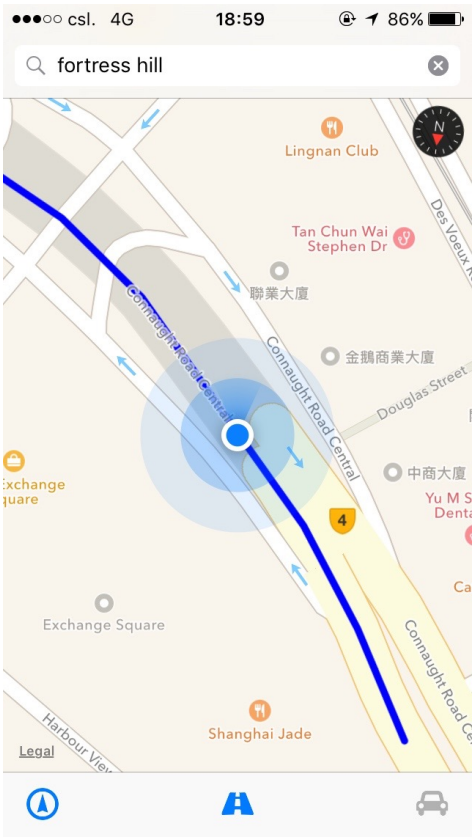
Within the final testing and evaluation period, there were several tests performed to test the functionality of Appy modules.

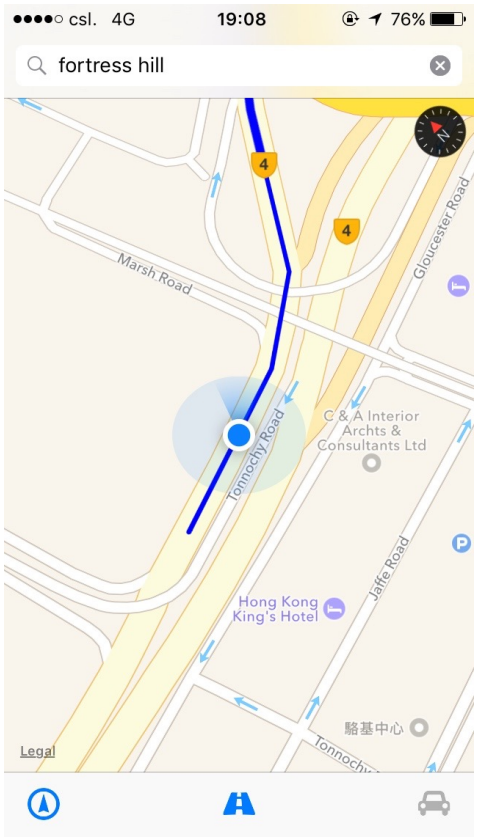
Test 1 was executed on 15th April 2016 on iPhone 5 iOS 9.3.1 (13E238)

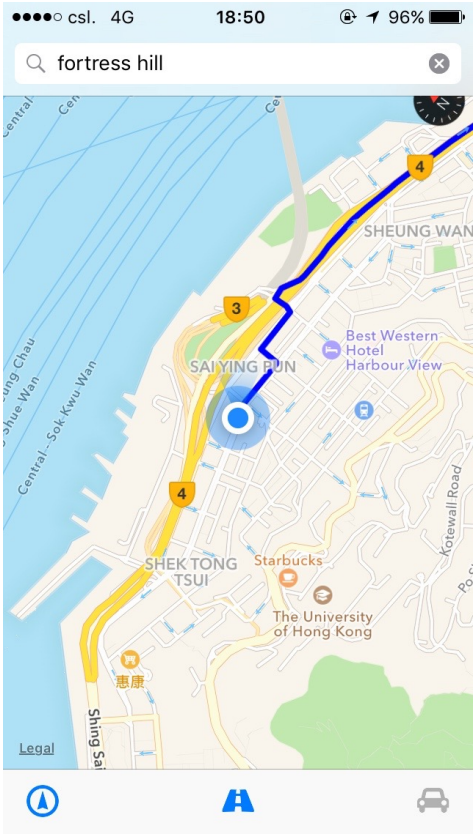
Camera Module



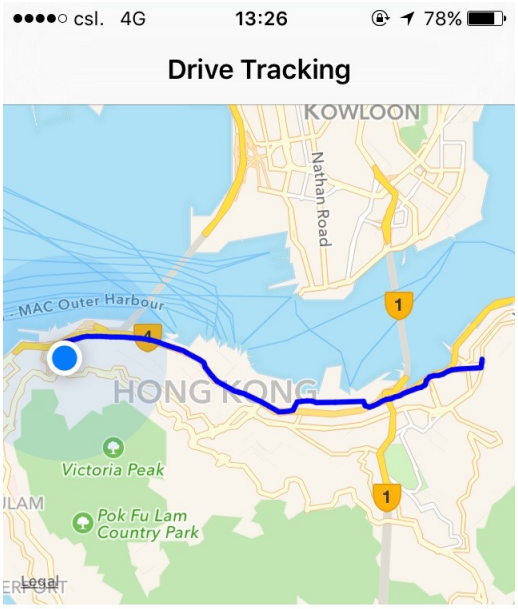
Navigation Module







Drive Tracking Module

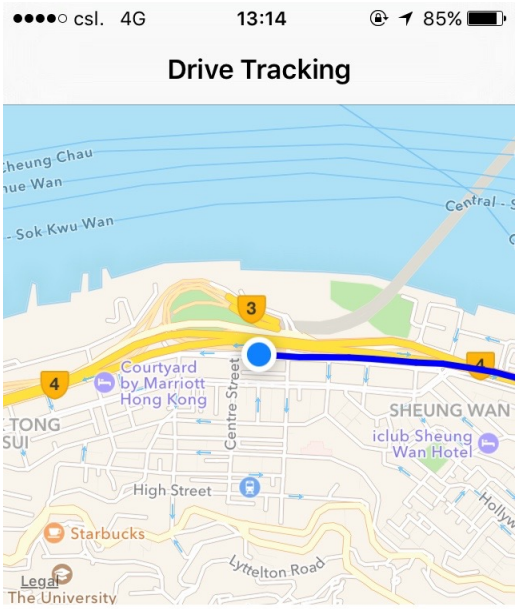


Time: 00:19:09

Distance: 6407.83199056871

Average Speed: 20.076758194

Start

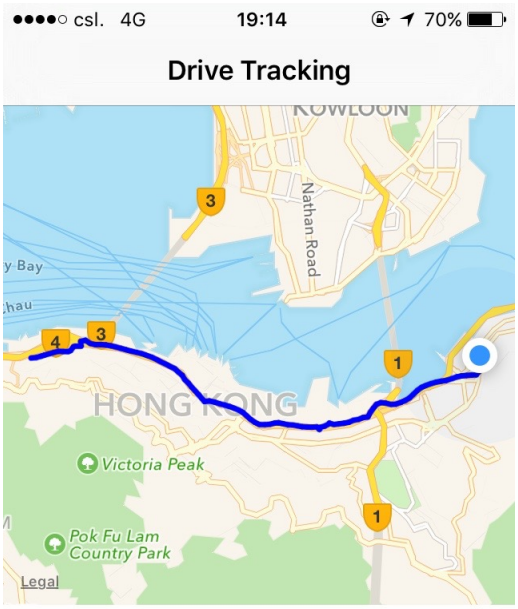


Time: 00:14:24

Distance: 5758.17405223404

Average Speed: 23.992391884

Stop

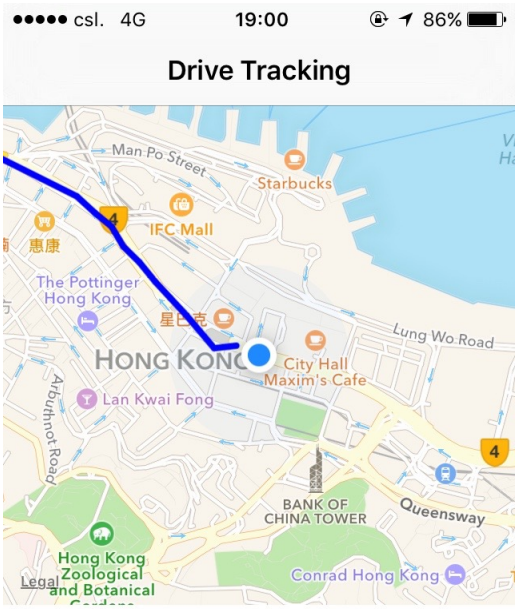


Time: 00:22:58

Distance: 6.65 km

Average Speed: 17.5 km/h

Start



Time: 00:09:18

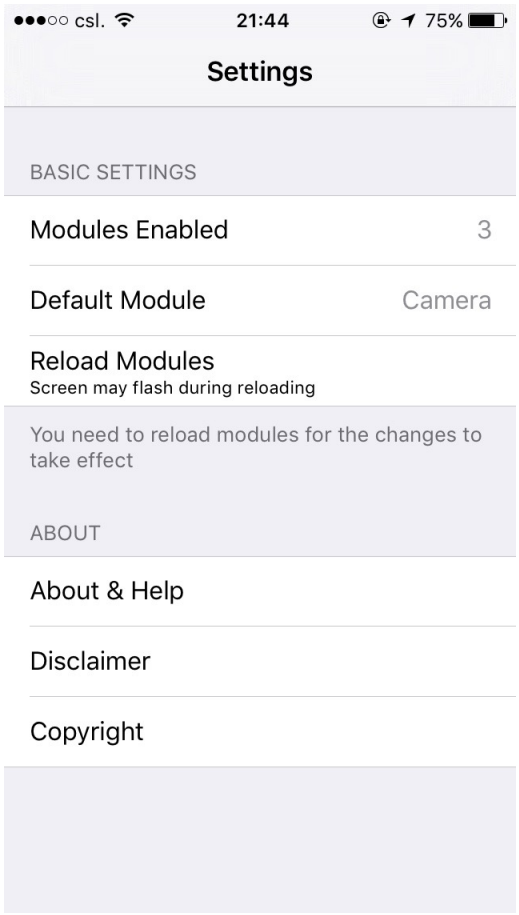
Distance: 2.64 km

Average Speed: 17.2 km/h

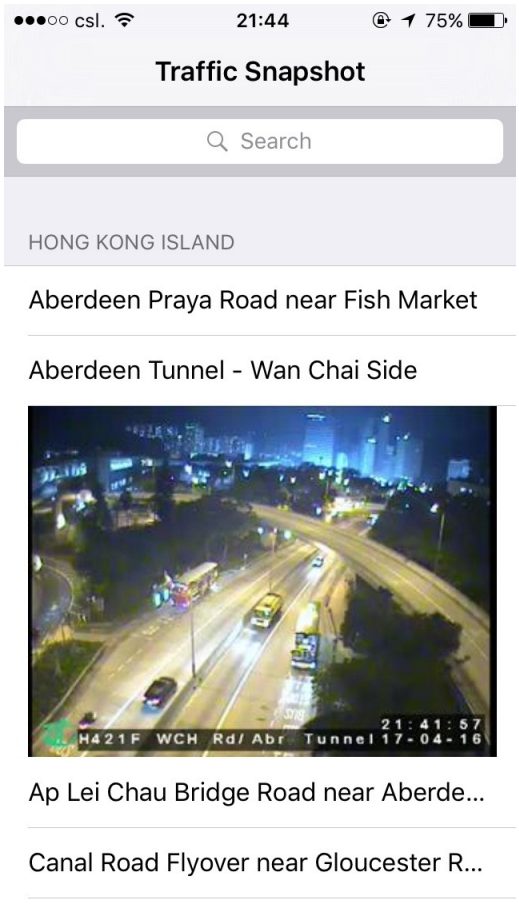
Stop

Test 2 was executed on 17th April 2016 on iPhone 5 iOS 9.3.1(13E238)

Settings Module



Traffic Snapshot Module



Future Development

In the future, Appy development would focus on the remaining part of Phase V that utilize the social capability of a smartphone. The remaining part can be divided into two parts, first part is the back-end API that allows user update their route via a .gpx file and JSON, while the second part is the front-end user interface and implementation.

For the former part, since handling loads of media data would incur a large demand to system resources, the back-end server itself is not able to handle huge media files at this stage. Instead, processing the locations detail and a URL of video uploaded to other platform would be an easier approach to do so. Locations data can be pass by JSON, plist or .gpx file while the video URL can be represented by String. Storing these light-weighted data is more practical regarding the maintenance and stability.

On the other hand, the front-end application needs to add few more MVC structure to enable social connectivity feature. Data could be parse by JSON and store within the Core Data Entities, while media file stay in the internal directory of the application sandbox. In the future, Appy aims to connect with other social platform such as Facebook and twitter by connecting public API from service supplier.

After finishing all five stages, the software development cycle moves to a testing and evaluation period, mainly focusing on fine tuning and bugs fix. The back-end server can upgrade to become a remote logger which gathers user anonymously error reporting. Moreover, as iOS SDK and Swift language updates rapidly the application also needs to be stay updated to catch the latest features.

Conclusion

Drivers would think vehicle recorder is only a device that capture the moments of car accidents, but apart from obtaining an evidence footage there are more features can be added on top of a plain vehicle recorder. This thought leads to the theory behind Appy, a multi-functional vehicle recorder application on iOS platform.

Designing application for drivers is unlikely to be same as an ordinary phone application, a handy user experience is the number one priority since no one would want the drives being distracted during driving. To enable various functionality working simultaneously within one application, Appy adopts a structural Model-View-Controller design pattern to break down a complex system into small, simple subsystem. As a result, a customizable user interface can be achieved by gathering many subsystems into one, with the objective to fulfill more users needs.