FIND MY CAR

Final Project Report

Angel Ortega

Version 1.0

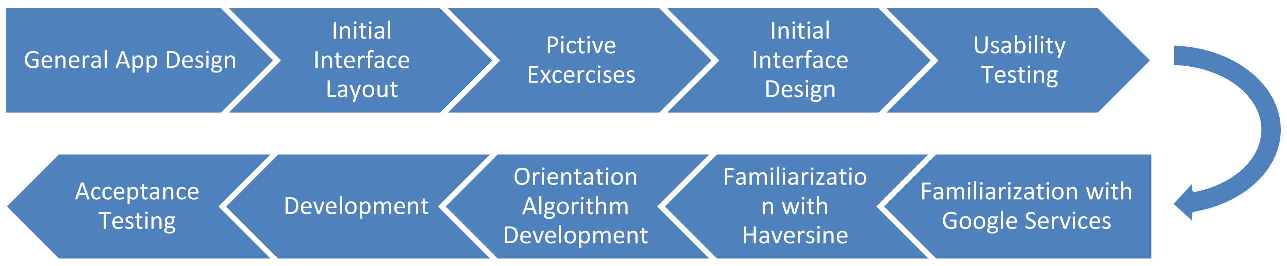
5/28/2016

Table of Contents

1. Project Overview

Find My Car is the approach to solve a recurring problem. Find My Car resulted from the inability to efficiently locate a vehicle in a large or unfamiliar parking space. Current apps in the market all had something in common: too hard to use. The overdesigned aspect of current apps made them inefficient for the task of locating a car in an easy and efficient manner. Currently, any user could actually locate their car using google maps, but the task requires at least five clicks. The interface also creates an information overload for the user. The user does not need to be rerouted every s/he take a wrong turn, and Google maps does not store parking aisle information or sidewalk information. With this in mind, I set out to create a simple yet effective app that would solve the problem without overwhelming the user with information or graphics.

2. Approach



3. Software Design

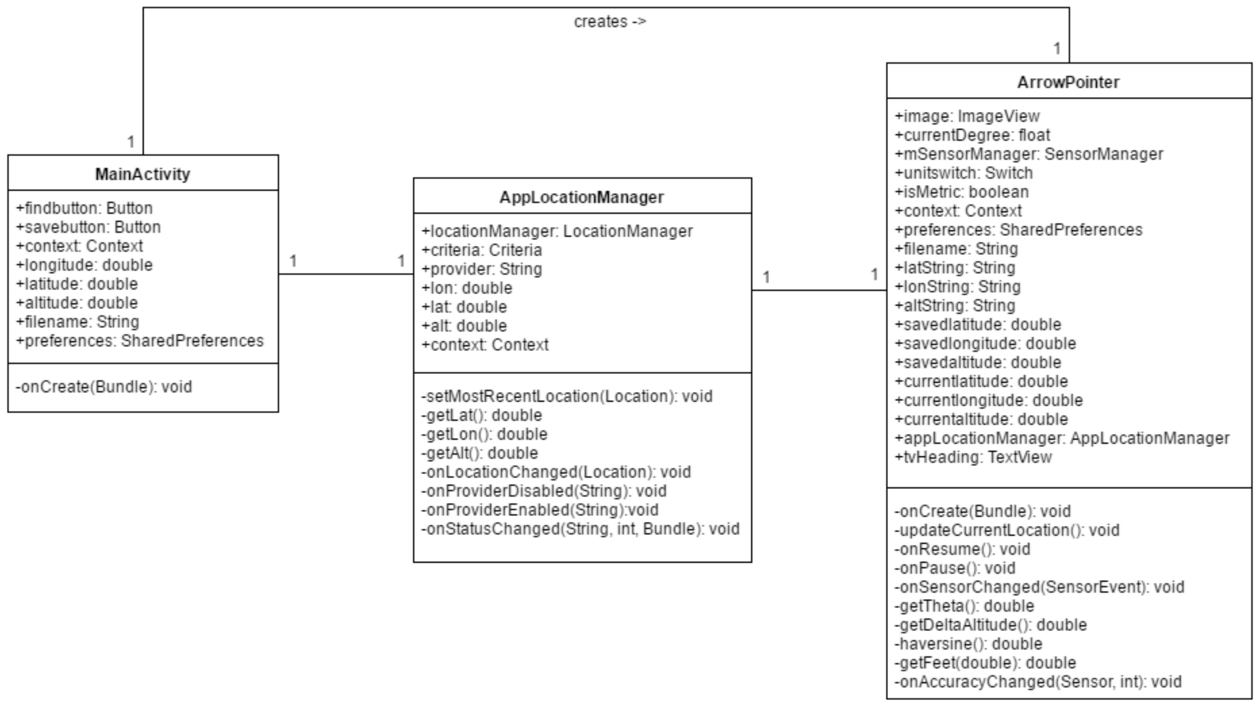


Figure 1: Class Diagram for FindMyCar Application

Figure 1 shows the class diagram for the FindMyCar Application. I needed a main page for the initial user interaction with the app. MainActivity serves as the main page for this interaction. MainActivity uses AppLocationManager to know current location in case user decides to save current location. MainActivity also creates ArrowPointer once user decides to find car.

The ArrowPointer class utilizes an ImageView to use a png file I created as a compass marker. The onSensorChanged method then rotates the marker to the general direction of the magnetic north plus the angle (theta) calculated between the tangent of the saved and current location and the magnetic north. The ArrowPointer class utilizes SharedPreferences to obtain saved location and AppLocationManager class to obtain current location values (latitude, longitude, and altitude). ArrowPointer class also displays an approximate distance between saved location and current location utilizing the haversine function. The ArrowPointer class also has the ability to display this distance in both meters and feet, at the user's discretion.

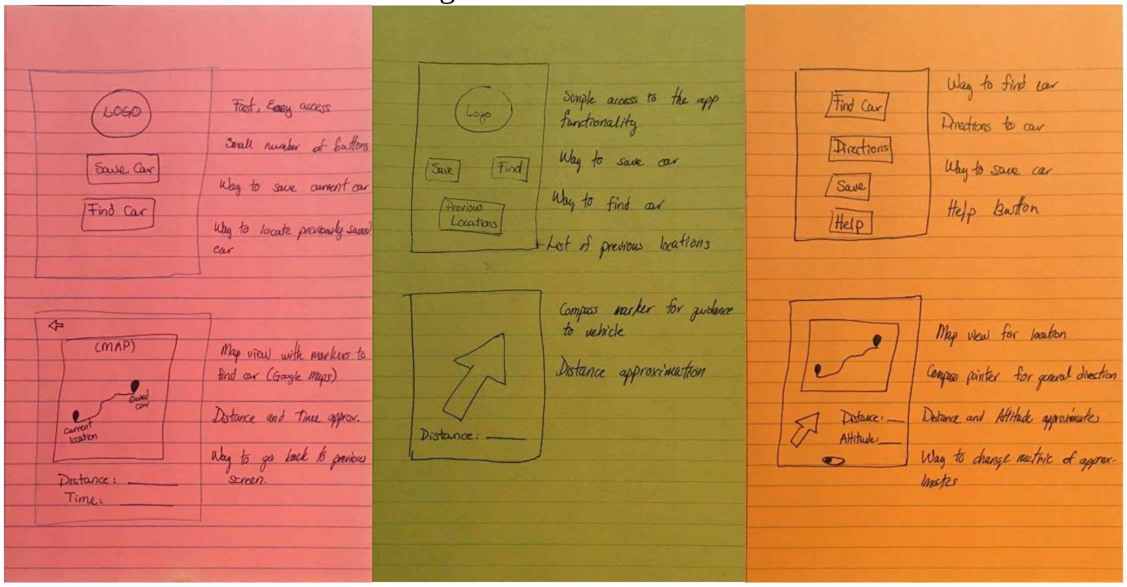
The AppLocationManager obtains and transmits GPS location values. The class utilizes android APIs to access current location. The class then separates the Location object into latitude, longitude, and altitude double values. These values can then be accessed in real-time by the other classes in the system.

4. User Interface Design

The User Interface Design included five steps: initial draft, pictive exercise, UI design, usability testing, and UI modification. The design of the interface follows Schneiderman's Eight Golden Rules of Interface Design. The rules were obtained from *Designing the User Interface* by Ben Schneiderman.

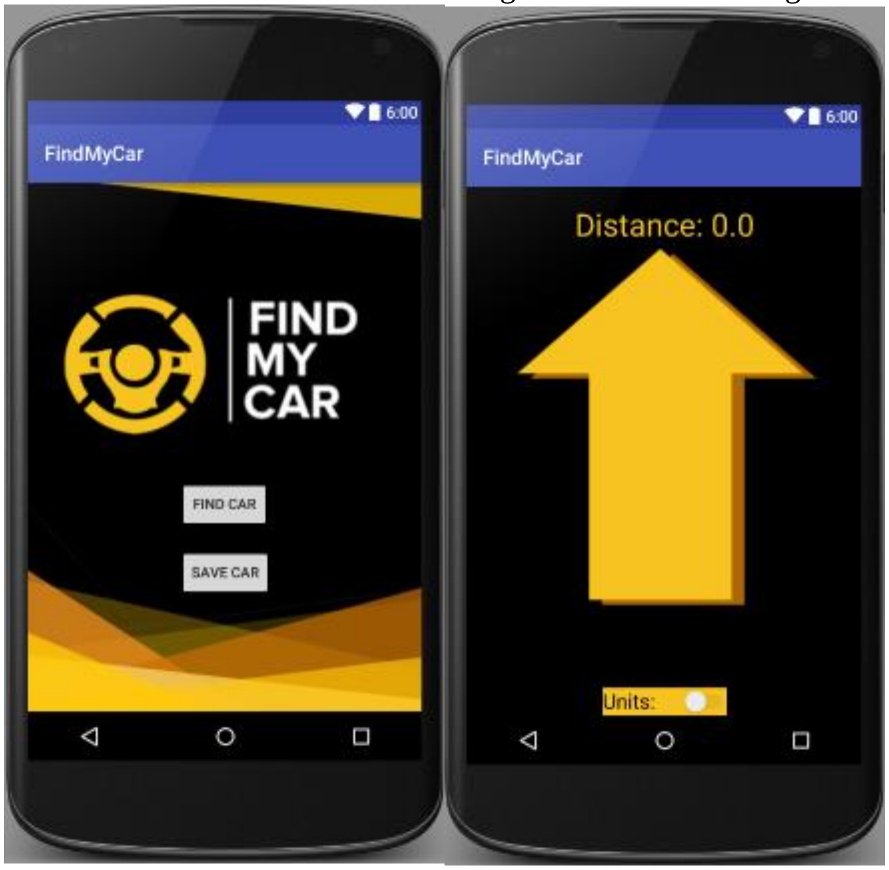
Initial Draft: The initial draft of the user interface had the goal of reducing short-term memory load, lower the time of interaction between user and app, and create an easy-to-use experience for the user.

Pictive Exercise: I had three students who drove to describe an interface they would be more likely to use for an application like FindMyCar. I followed principles learned in Human Computer Interaction to conduct the exercise. Figures 3-5 show the results of the exercise.



Figures 3-5: Pictive Exercises conducted for Interface Design

UI Design: Using all three, I designed the User Interface of the app. The interface consists of two activities. The first activity contains two buttons, one for saving car (current location) and one for finding car (last saved location). Toast messages inform user of changes in the state of the application like current location, or when a location is saved. The second activity consists of a compass marker that points in the general direction of the last saved. The second activity also displays an approximate distance to the saved location from the current location. By default, the distance approximation is displayed in meters, but a switch allows the user to alternate between meters and feet. Both views utilize the same color palette. I designed the marker and the logo utilizing Microsoft Publisher. The User Interface I designed can be seen in Figures 6-7.



Figures 6-7: User Interface Design of FingMyCar.

Usability Testing: Usability testing of the interface design concluded high affordance of interaction items, low short-term memory load, high ease of use, and overall good uniformity.

UI Modification: The usability testing exposed some issues with text size that were easily fixed before final implementation of the interface.

5. Implementation

As it can be seen by the class diagram, the app utilizes a total of three classes. Each class has a particular task. The MainActivity serves as the first interaction for the user. The main activity manages user input in the form of two buttons. The main activity seems simple, but it also utilizes the AppLocationManager class to know the current location and SharedPreferences to persist this information if the user indicates s/he wants to save this location.

The main activity then calls the ArrowPointer class to manage the guidance to the last saved location. The ArrowPointer class also uses the SharedPreferences to know saved location and it also uses the AppLocationManager to find out current location. The ArrowPointer class handles all the algorithmic information regarding path and direction approximation. This class not only handles the display of the marker, but also the calculation of degree and distance, and the management of rotation animation of the marker. The ArrowPointer class uses android sensor APIs to handle marker rotation animation. I followed TechRepublic's tutorial on creating a compass to begin the implementation of this class [1].

The AppLocationManager uses android location APIs to retrieve GPS location. This class retrieves current location in the form of a location object and retrieves specific data from this object in the form of doubles (latitude, longitude, and altitude). The class has interface classes to handle the retrieval of this information by the other classes. I saw some discussion boards on StackOverflow to determine the best approach for retrieving current location from GPS [2]. Google location services proved too complicated to work with. In the end I decided to use Android.Location APIs [3]

Some of the neat features of the app is its ability to display approximations in both meters and feet. The ability for users to save a location by the click of one button also sets this app apart from some of its existing counterparts. The use of a compass marker also allows for expandability to wearable technology like smart watches. At first sight, the app might not seem very complex, which was the goal of creating a simple user interface, but in reality, the app handles a lot of data and displays data in real time. Every time the device detects a change in the motion sensors, the data being displayed (orientation of marker and distance approximation) is updated. The app also takes into account the Earth's curvature through the use of the Haversine function [4] instead of the use of triangular approximation. I also utilized triangular algebra to approximate the angle between the tangent of the saved and the current location and the magnetic north as detected by the device. This approach can be seen in Figure 8.

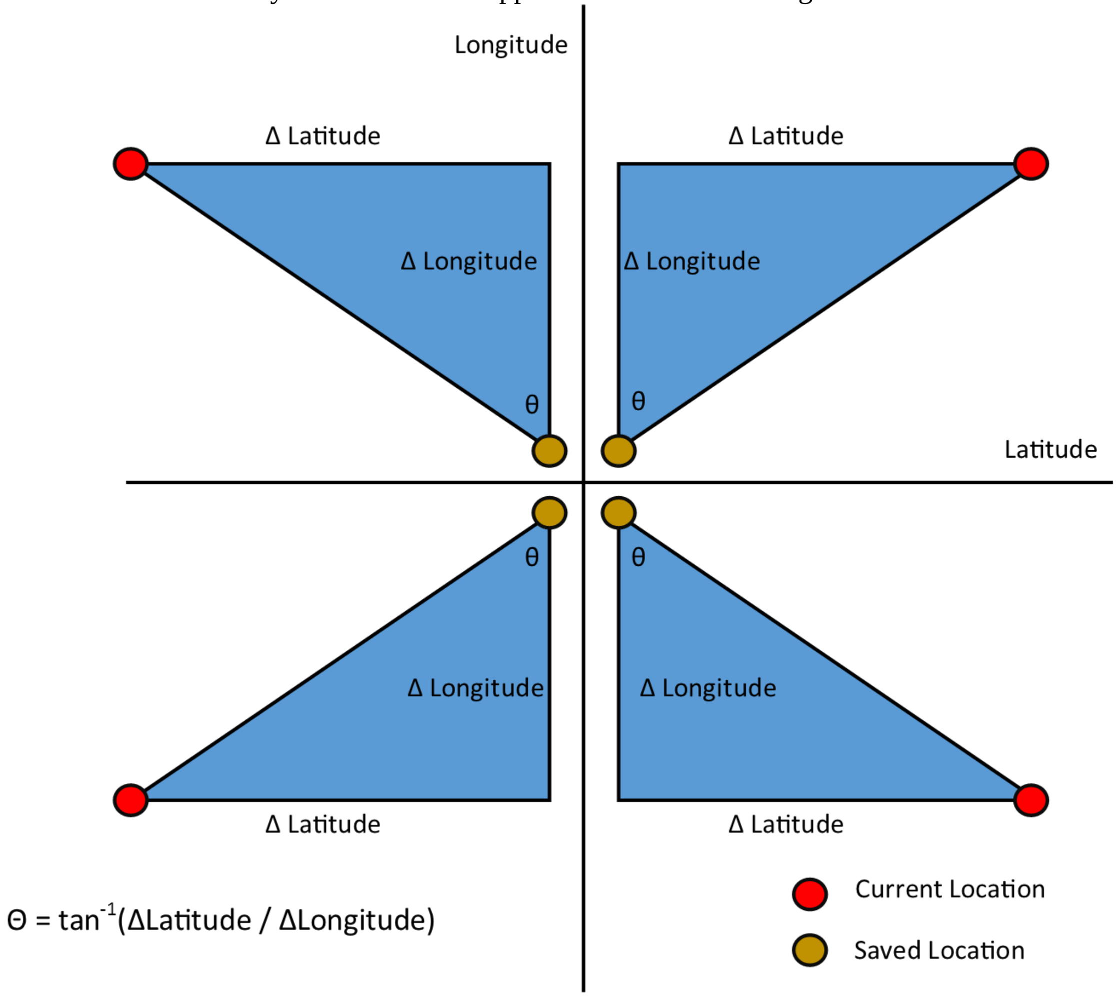


Figure 8: Triangle diagram of degree calculation.

6. Lessons Learned

Depending on the purpose of the application, sometimes less is more. I am referring to the overload of UI elements in the interface. At first, my app was so complex that I was just creating another complex location application. I think the power in this app is its seamlessness. The user does not know how much work the app is doing, the user just knows that the app is constantly modifying itself depending on phone orientation and location. The perception of the user should be separate from the internal working of the app. This yields in a good design for the interface. The user should not worry about how an app is doing something, just that it is doing it correctly. User input, especially for mainstream applications, is crucial to the development of a good app. The input I received during the pictive exercise and the usability testing was insightful and very useful when determining the final design of the user interface.

Because of the nature of mobile applications, you should be aware of limitations caused by the hardware. For example, I began testing in an RCA Tablet. I did not know that this tablet did not have a gyroscope, and thus, I was unable to test the marker rotation in that device. Location accuracy and availability is also dependent on the hardware. The app development should comply with the target device specifications.

7. References

[1] http://www.techrepublic.com/article/pro-tip-create-your-own-magnetic-compass-using-androids-internal-sensors/

[2] http://stackoverflow.com/questions/17591147/how-to-get-current-location-in-android

[3] http://developer.android.com/reference/android/location/package-summary.html

[4] https://rosettacode.org/wiki/Haversine_formula

8. End of Document

End of Document