**Clemson Tiger Tour Technical Reference Manual (Version 4)**

**Project Name: Clemson Tiger Tour**

**Team Members Hongda Li, Zian Chen**

**Project Title: Clemson Tiger Tour**

**1. Briefly Description:**

This application is aiming to provide the user with a method to take a tour easily in Clemson University Campus. It consists of two Modules. The first module is called the “Camera Hunting” which allows the user to scan around using the camera on the device. The scanned buildings would be displayed on the screen of the device represented by a label. If the user is interested in one of the buildings, he/she can click the label of that building, and then a piece of descriptive information will be displayed to the user. The second module is called the “Tiger Tour” which provides the user with a compass to show the direction and the distance of a destination selected by the user. The user’s location should also be indicated by a blue dot on a campus map. The user could be guided to the destination by the compass.

**2. External Database**

We have six interrelated tables in the external database. These six tables are: Combination, Building, Text, Image, Audio, and Video. In this version, the application may not include any videos or audios. But we expect that this application is extensible and could easily be upgraded to support videos and audios. So in our PHP code, we simply ignore the video and audio tables in this version since we are not going to implement the features of playing videos or audios in out application. But still, we reserve the Video and Audio tables in our external database (no records in the tables).

The Combination table stores the relationships among the other five tables.

The Building table stores the building records, which include the longitude, the latitude and the name of the building.

The Text table stores the URLs of text files each of which includes a piece of textual description.

The Image table stores the URLs of image files each of which is a descriptive picture of a building.

The Audio table stores the URLs of audio files each of which contains a piece of voice information about a certain place such as a building.

The Video table stores the URLs of video files each of which contains a piece of video that is related to a certain place such as a building.

Every time the application needs the building information, it will search the database using the latitude and the longitude of the building. For example, we can select a building with -33.86 as its latitude and 151.20 as its longitude by using the following SQL statement:

*SELECT \**

*FROM BUILDING*

*WHERE BUILDING.LATITUE = -33.86 AND BUILDING.LONGITUDE = 151.20;*

The SQL statement will then return with building\_id, latitude, longitude and the building\_name.

The schema of the external database is shown as Figure-2. Artifacts stored in the external database will be described in Session 4.

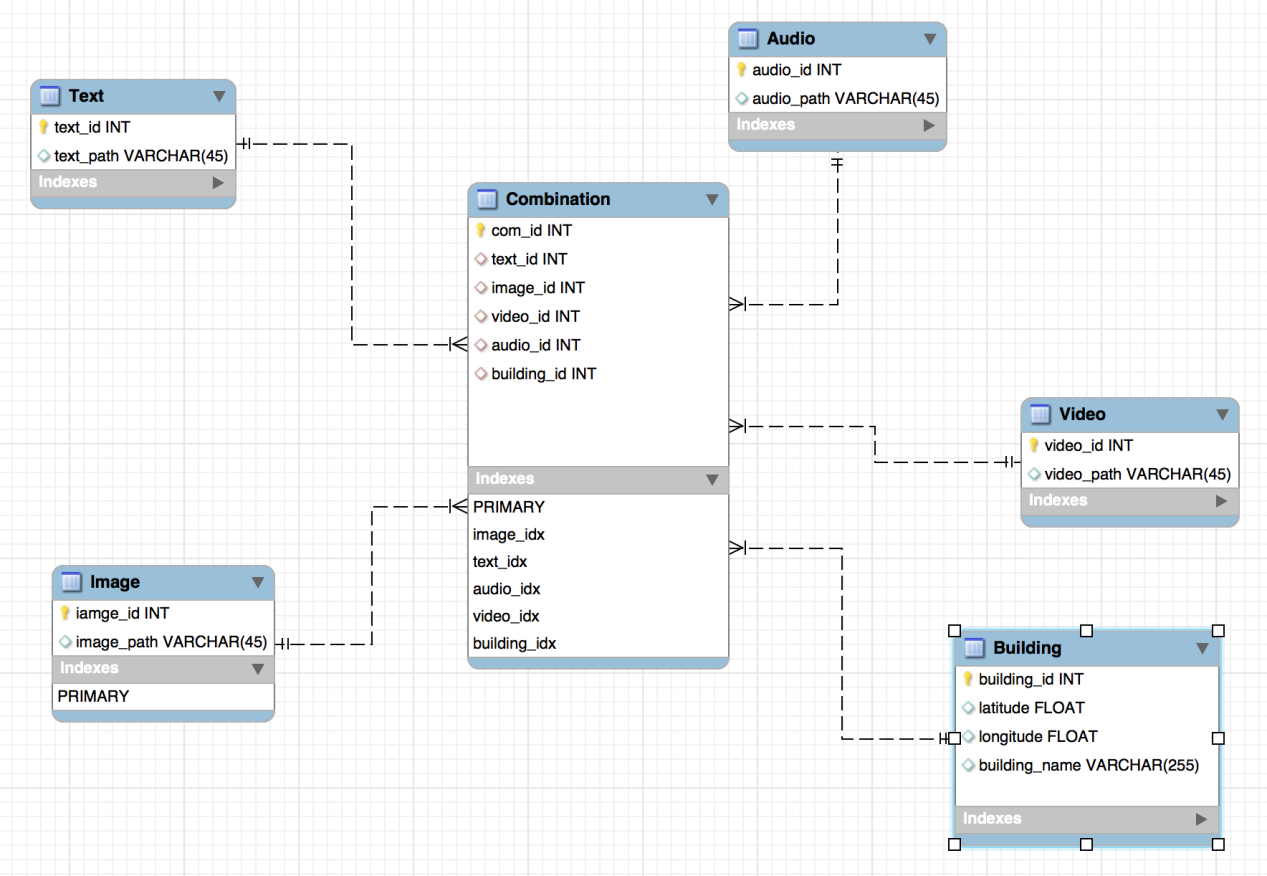


Figure-2 External Database Schema

**3. User Interface**

**1) Welcome View:**

This view shows the welcome page. When the user opens the application, this view will show up firstly. The RootViewController.m implements the welcome view.

There are two functions in the RootViewController.m.

*- (void)sideMenu:(RESideMenu \*)sideMenu willShowMenuViewController:(UIViewController \*)menuViewController;*

*- (void)sideMenu:(RESideMenu \*)sideMenu willHideMenuViewController:(UIViewController \*)menuViewController;*

These two function will be called if the user wants to show or hide the left side-menu.

In the left side-menu, the user can choose to go back to welcome page or to either the Tiger Tour View or the Tiger Eye View. To implement this selection functionality, we write LeftMenuViewController.m. The core function in this file is:

*- (UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath*

This function implements a table view which contains multiple options.Each option is related to a data cell of the table view.

1. **Tour Guide View**

This view contains a Google map that can display the location of the user. Basically, this view has a property named

*GMSMapView \*mapView\_;*

The mapView property is attached to a Google map using the API provided by Google. In order to use Google’s map service, we have to call a number of functions:

*GMSCameraPosition \*camera =*

*[GMSCameraPosition cameraWithLatitude:-33.86*

*longitude:151.20*

*zoom:6];*

The function shown above is creating a camera view so that the Google map could be attached to that camera view.

*mapView\_ = [GMSMapView mapWithFrame:CGRectZero camera:camera];*

The function shown above is attaching the mapView with the camera view we just created.

After those two steps, a Google map will show up on the screen. The center of the view will focus on the position with -33.86 as its latitude and 151.20 as its longitude.

In this view, a compass pointing to the destination will display on the left bottom corner.

**3) Tiger Eye View**

The Tiger Eye view actually is a camera controller. We use cameraController.m to implement this view. The core function of the cameraController.m is:

*-(void)addDevice*

The function shown above creates a variable named “videoDevice”. Then the videoDevice is connected to the real camera device. And we also implement a bunch of functions to control the camera device, such as changing the portrait and landscape of the camera. These functions includes:

*- (void)setPortrait*

*- (void)setLandscapeLeft*

*- (void)setLandscapeRight*

We use coreLocation framework to obtain the user’s current location and heading direction. According to the user’s current location and heading direction, the application can compute the geometrical relationship between the target building and the user’s location. After that the application shows the buildings on the screen with labels if the camera is pointing to the buildings. In order to exploit the coreLocation framework the application must firstly declare a locationManager that takes responsibility for obtaining the location and heading direction.

CLLocationManager \*locationManager;

After that, the application would call several functions to obtain current location and heading direction. There are two types of functions. One type requests the starting location and heading direction update. The other are functions that receiving the updated location and heading direction information from the locationManager.

The functions that request starting the location and heading direction update are:

- (void) startUpdatingLocation

- (void) startUpdatingHeading

The functions that receiving the updated location and heading direction information from the locationManager are:

- (void)locationManager:(CLLocationManager \*)manager

didUpdateToLocation:(CLLocation \*)newLocation

fromLocation:(CLLocation \*)oldLocation

- (void)locationManager:(CLLocationManager \*)manager

didUpdateHeading:(CLHeading \*)newHeading

heading direction information is available. Then the application obtains the new location and new heading direction by:

newLocation.coordinate.latitude;

newLocation.coordinate.longitude;

newHeading.trueHeading;

Now the application successfully obtains the user’s current location and heading direction. Upon the user moving the device, these functions will be called and thus the location and heading direction information are updated.

After the application obtains the location and heading information, functions will be called to compute the position of the buildings nearby. We implement two functions to compute the direction and distance of the buildings respectively. These two functions are:

-(double)GetDistance:(double)lat1 firstLon:(double)lng1

secondLat:(double )lat2 secondLon:(double )lng2;

-(float)GetAngle:(double)lat1 firstLon:(double )lng1

secondLat:(double)lat2 secondLon:(double)lng2 angle:(float)ang;

The first function returns the distance of the position given by (lat2, lng2) as its latitude and longitude. The second function returns the direction angle of the position given by (lat2, lng2) as its latitude and longitude.

**4. Methodology of Direction Detection**

Figure-3 shows the method to detect the direction of a building. Assume that the user stands at P0, and the destination locates at P1. The latitude and the longitude of P0 is notated as (, ), and the latitude and the longitude of P1 is notated as (, ). We make a point, P2 with its latitude and longitude as (, ). P2 locates at the east of P0, and the south of P1, so P2 has the same latitude with P0 and the same longitude with P1.

Our objective is to compute the angle. We note that approximates θ, since P0, P1 and P2 are very close to each other on the surface of the earth. So we compute instead.

is one of the angles of triangle , so we use formulas[1]:

th.png

a.png

b.png

c.png

where r is the radius of the earth, to compute .

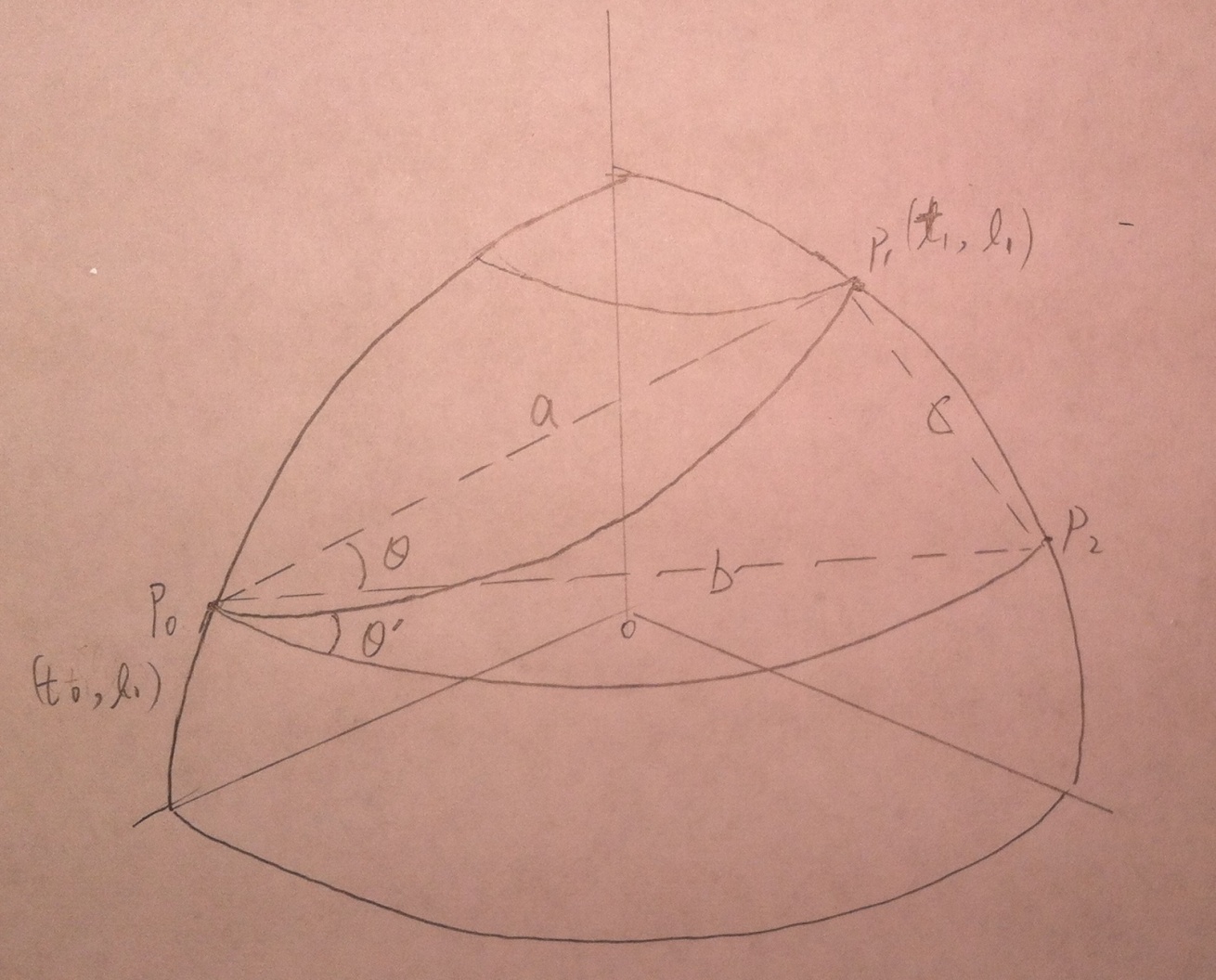


Figure-3 Direction Computation Method

In addition, we can also obtain the heading direction of the device by using the coreLocation framework. The next step is to figure out the offset angle of the building based on the center of the camera screen so that we can put the label in the right place of the screen. The offset angle is notated as . As is shown in Figure-4,



where is the angle between the device heading direction and the true North and is the angle between the building direction and the true East. Once we obtain the offset angle , we can compute the x coordinate of the label displayed on the device screen:



where w is the width of the screen (i.e. 480 pixels). The y coordinate of the label is set randomly.

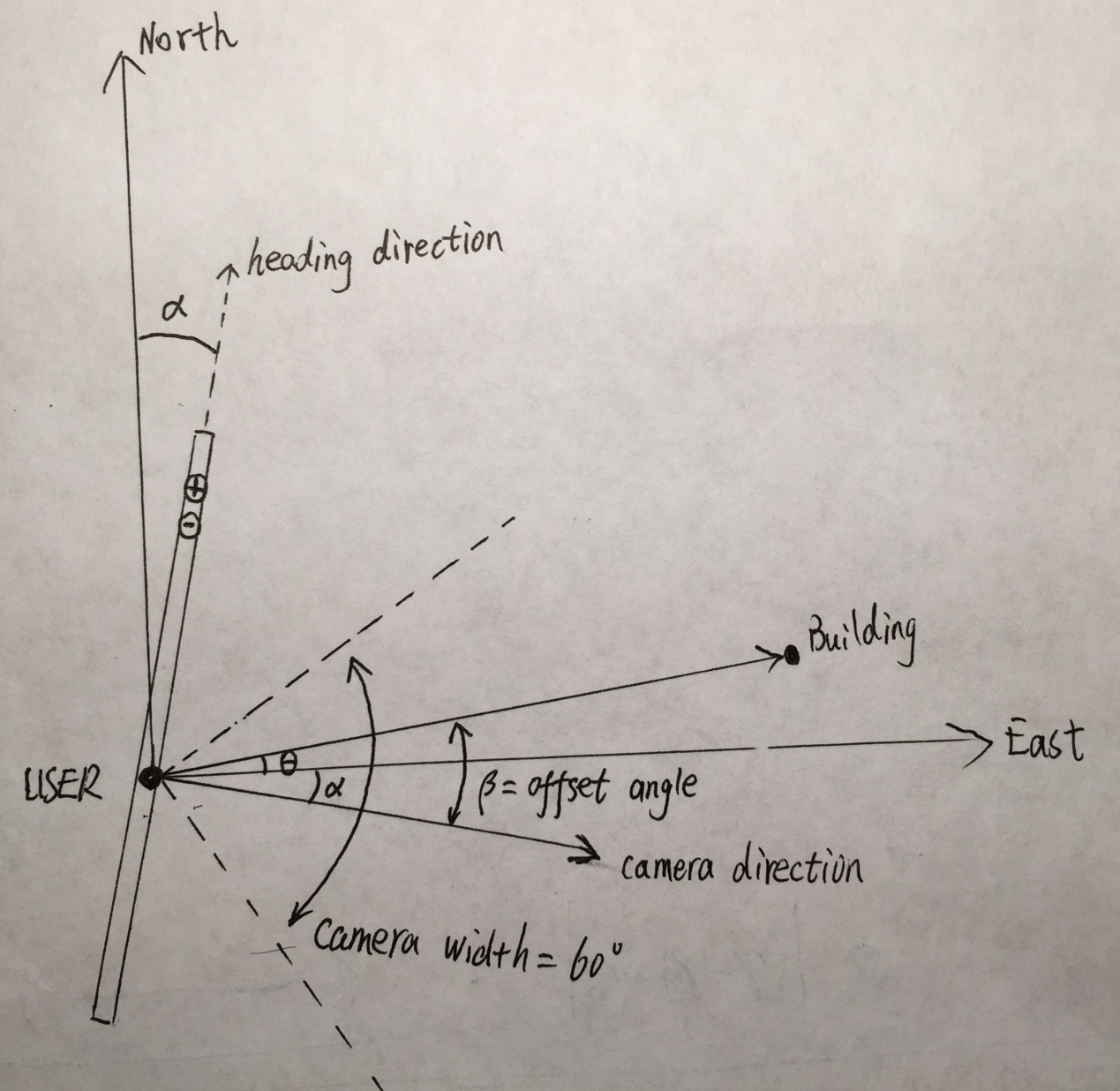


Figure-4 Computation of Offset Angle

The distance between the building and the user's location is computed by the formula[2]:

dist.png

where r is the radius of the earth, (, ) is the latitude and longitude of the user position and (, ) is the latitude and longitude of the building position.

**5. Artifacts Stored in Database**

We will not use an internal database since we can put all the records in an external database. The schema of the external database is shown in Figure-2. We have populated some initial data to the external database.

We store buildings’ related pictures and texts in a remote file system and put the URLs into the table records. We store the building’s latitude and longitude as float numbers in the table records.

**6. Features of Mobile Device**

In this version, we have made a big progress in our features. Most importantly, the application can obtain the user’s current location and heading direction in real time, which plays the key role in the “Camera Hunting” module.

1) In this version, we implement the GPS feature. The user can now obtain his/her current location via the application.

2) We implement the Camera feature. The application now can run on a real iPhone machine and display the Camera view.

3) We successfully employ the corLocation framework which allows the application obtain the heading direction of the device. Thus the screen shows the heading direction in real time.

4) The application can now display the location of a building on the device screen.

5) When the label is clicked, information page of the related building will show up on the screen.

6) The application can use a compass to show the direction of the destination so that the user can follow the guide to go to the destination.

**7. External Database Access**

We are going to use PHP files deployed on the department server to access the external database deployed on buffet.cs.clemson.edu. We also use phpMyAdmin as a database browser to manage our MySQL database on buffet.cs.clemson.edu.

We use PHP files to select records from the external database. The Select.php is shown in Figure-5.

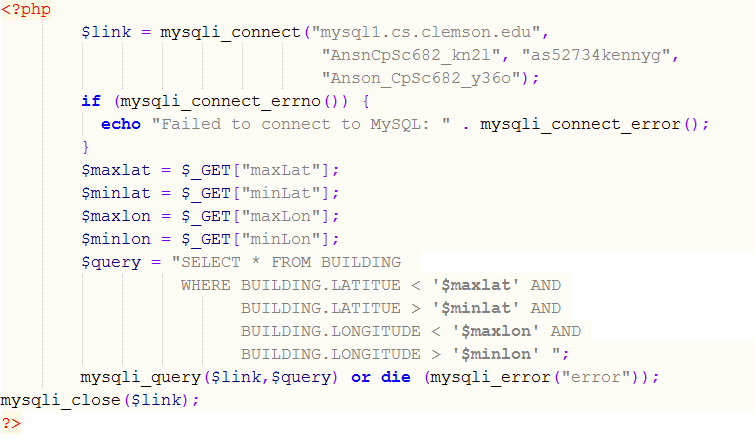


Figure-5 Access external database via PHP script

By using this PHP script the application can obtain building records whose latitudes are less than maxLat and greater than minLat, while longitudes are less than maxLon and greater than minLon. So the application can fetch several building records within a small area and then decide which building should be displayed on the screen using the method introduced in session 4.

**8. Authoring tools**

We are going to install the application on an iPhone and then project the iPhone screen on a projector.

**9. References**

[1] Law of cosines, http://en.wikipedia.org/wiki/Law\_of\_cosines .

[2] Geographical distance, http://en.wikipedia.org/wiki/Geographical\_distance