# LOCATION BASED SERVICES IN ANDROID

Ch. Radhika Rani[1], A. Praveen Kumar[2], D. Adarsh[2], K. Krishna Mohan[2], K.V.Kiran[2]
[1]Asst. Professor, Dept. of C.S.E., K L University, A.P, India
[2]IV/IV B.Tech Students, Dept. of C.S.E, K L University, A.P, India

***ABSTRACT***
*Location Based Services provide the users a set of services which originate from the geographic location of the user's mobile device. Using these services it is possible for the users to find and locate other persons, vehicles, resources and also to provide location-sensitive services, in addition to tracking their own location. The request for location can originate in the mobile device or in another entity like application provider or network operator. It is possible to automatically trigger Location Based Services when the mobile device is at a particular location. These services can also originate in the user's mobile device itself in order to satisfy location-based requests like finding areas of interest, checking traffic conditions, finding our friends, vehicles, resources, machines and emergency requests. In this paper we will discuss how to implement these location based services in Android after giving a brief introduction to Android and its constituents.*

***KEYWORDS:*** *Android, Location based service, Location Manager, Location Provider, GPS, Geocoding, Overlays.*

## I. INTRODUCTION

Android is a combination of three components:

- Free and open source operating system for mobiles.
- Open source development platform for creating mobile applications.
- Devices, particularly mobile phones that run Android operating system and the applications created for it.

Android SDK was released by Open Handset Alliance in the month of November of the year 2007. Android is actually developed using the kernel of Linux 2.6 and the highlighting features of Android include the following [7]:

- No fees for licensing, distribution and release approval
- GSM,3G EDGE networks for telephony
- IPC message passing
- Background processes and applications
- Shared data stores
- Complete multimedia hardware control
- API's for location based services such as GPS.

The Location Based Service (LBS) applications can help user to find hospitals, school, gas filling station or any other facility of interest indicated by user within certain range [2]. Just like a GPS device its location will also be updated as soon as user changes his/her position. Android can be considered as a unified software package. This software package includes an operating system, middleware and core applications. Android SDK provides some tools and API's which are required to develop Android applications using the programming language of Java. Android platform provides open system architecture along with a powerful debugging environment. It is also characterized by optimized graphics systems, rich media support and a embeddable web browser. It uses a Dalvik virtual machine (DVM) which gives the same priority to all processes and hence optimises the execution of application.GPS, camera, compass and 3d-accelerometer are also supported by Android and they provides some useful APIs for the functions of map and location. Anyone can access, control

and process the Google map which is also free and implement location based services in his mobile device [3].

The skeleton of Android framework and the jargon of Android application development are discussed in section II and section III. The design of location based services in Android is discussed in section IV, the discussion regarding the development of an application which can find route between two geographical locations is mentioned in section V and its corresponding results are mentioned in section VI. After this, section VII gives the conclusion and the section VIII gives the future scope.

## II. SKELETON OF ANDROID

The skeleton of Android framework [9] and its constituents are shown in the following figure:

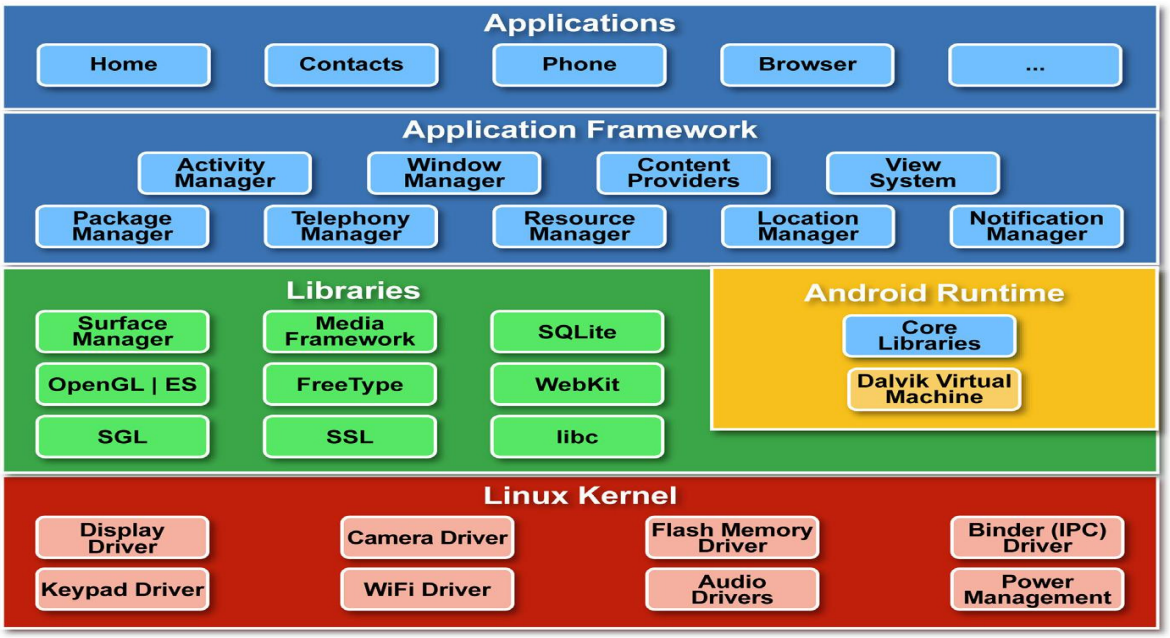


**Figure 1.** Skeleton of Android

### 2.1. Applications Layer

Android ships with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are built using the Java. Each of the application aims at performing a specific task that it is actually intended to do.

### 2.2. Application Framework Layer

The next layer is the application framework**.** This includes the programs that manage the phone's basic functions like resource allocation, telephone applications, switching between processes or programs and keeping track of the phone's physical location. Application developers have full access to Android's application framework. This allows them to take advantage of Android's processing capabilities and support features when building an Android application. We can think of the application framework as a set of basic tools with which a developer can build much more complex tools.

### 2.3. Libraries Layer

The next layer contains the native libraries of Android. These shared libraries are all written in C or C++, compiled for the particular hardware architecture used by the phone and preinstalled by the phone vendor. Some of the core libraries are listed in Fig.1.

### 2.4. Android Runtime Layer

Android Runtime layer includes Dalvik Virtual Machine (DVM) and a set of core java libraries. Every Android app gets its own instance of DVM. Dalvik has been written so that a device can run multiple virtual machines efficiently and it executes files with .dex (dalvik executable format) extension optimised for minimum memory.

### 2.5. Linux Kernel

This layer includes Android's memory management programs, security settings, power management software and several drivers for hardware, file-system access, networking and inter process communication. The kernel also acts as an abstraction layer between hardware and rest of the software stack.

## III. JARGON OF ANDROID APP DEVELOPMENT

The basic components of an Android application include Activity, Broadcast Receiver, Service, and Content Provider. Each of these which when used for any application has to be declared in the AndroidManifest.xml. The user interface of the component is determined by the Views. For the communication among these basic components we use Intents and Intent filters which play crucial role during app development [1].

### 3.1. Activity

An Activity is, fundamentally, an object that has a lifecycle. An Activity is a chunk of code that does some work; if necessary, that work can include displaying a UI to the user. It doesn't have to, though - some Activities never display UIs. Typically, we will designate one of our application's Activities as the entry point to our application [4].

### 3.2. Broadcast Receiver

Broadcast Receiver is yet another type of component that can receive and respond to any broadcast announcements.

### 3.3. Service

A Service is a body of code that runs in the background. It can run in its own process, or in the context of another application's process, depending on its needs. Other components "bind" to a Service and invoke methods on it via remote procedure calls. An example of a Service is a media player; even when the user quits the media-selection UI, she probably still intends for her music to keep playing. A Service keeps the music going even when the UI has completed.

### 3.4. Content Provider

Content Provider is a data storehouse that provides access to data on the device; the classic example is the Content Provider that's used to access the user's list of contacts. Our application can access data that other applications have exposed via a Content Provider, and we can also define our own Content Providers to expose data of our own.

### 3.5. Intents

Intent is a simple message object that represents an "intention" to do something. For example, if our application wants to display a web page, it expresses its "Intent" to view the URI by creating an Intent instance and handing it off to the system. The system locates some other piece of code (in this case, the Browser) that knows how to handle that Intent, and runs it. Intents can also be used to broadcast interesting events (such as a notification) system-wide. There are two types of intents namely implicit and explicit intents. Implicit intents has no specified component where as Explicit intents do specify the component.

### 3.6. AndroidManifest.xml

The AndroidManifest.xml file is the control file that tells the system what to do with all the top-level components (specifically activities, services, intent receivers, and content providers described below) we have created. For instance, this is the "glue" that actually specifies which Intents our Activities receive.

### 3.7. Views

A View is an object that knows how to draw itself to the screen. Android user interfaces are comprised of trees of Views. If we want to perform some custom graphical technique (as we might if we're writing a game, or building some unusual new user interface widget) then we would create a View.

### 3.8. Notification

A Notification is a small icon that appears in the status bar. Users can interact with this icon to receive information. The most well-known notifications are SMS messages, call history, and voicemail, but applications can create their own. Notifications are the strongly-preferred mechanism for alerting the

user of something that needs their attention.

## IV. DESIGN OF LOCATION BASED SERVICES (LBS)

Location based services is an umbrella term is used to describe the different technologies used to find the current location of a device. The main LBS elements include:

- Location Manager
- Location Provider


**Figure 2.** Address search interface

Location Manager acts as a hook to LBS. Each of the Location providers represent a different location finding technology used to find the current location of a device[7]. Using Location Manger we can Perform following tasks:

- Findind current location
- Tracking the movement
- Setting proximity alerts for specified locations
- Checking all the available location providers

We can use the Location Controls in DDMS perspective in Eclipse to perform location changes directly into emulator's GPS Location Provider either manually or by using KML (or GPX) tabs in

Location Controls. In the Manual tab, we can send geo coordinates (latitude and longitude) manually. GPX stands for GPS Exchange format. Most GPS systems record tracking files using GPX. KML stands for Keyhole Markup Language. It is used extensively online to define geographic information. We can write KML file by our hand or generate it using Google Earth to find directions between two locations.

The most common location providers are GPS provider and Network provider. They can be accessed by using the static string constants mentioned below that can return the corresponding provider name:

- LocationManager.GPS_PROVIDER
- LocationManager.NETWORK_PROVIDER

To access a specific provider we can use getProvider() method as shown in the following code snippet:

```
String providerName = LocationManager.GPS_PROVIDER;
LocationProvider gpsProvider;
gpsProvider = locationManager.getProvider(providerName);
```

To see all the available location providers we use getProviders( ) method as shown in the following code snippet:

```
boolean e = true;
List<String> providers = locationManager.getProviders(e);
```

To select best provider we can set criteria regarding cost, power consumption, altitude, speed, accuracy, direction, we can use get Best Provider( ) method as shown in following code snippet :

```
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_COARSE);
criteria.setPowerRequirement(Criteria.POWER_LOW);
criteria.setAltitudeRequired(false);
criteria.setBearingRequired(false);
criteria.setSpeedRequired(false);
criteria.setCostAllowed(true);
String bp = locationManager.getBestProvider(criteria, true);
```

### 4.1. Finding our Location

First we need to get a debugging certificate from google using MD5 fingerprint which can be generated using key tool. MD5 fingerprint is unique for any user. The Map API key thus generated is used for display of Google map in the emulator. The purpose of location-based services is to find the physical location of the device. Access to the location-based services is handled by the Location Manager system Service. To access the Location Manager, request an instance of the LOCATION_SERVICE using the get System Service() method, as shown in the following code snippet:

```
String ss = Context.LOCATION_SERVICE;
LocationManager lm;
lm = (LocationManager)getSystemService(ss);
```

Before we can use the Location Manager we need to add one or more uses-permission tags to our manifest to support access to the LBS hardware. We can have fine and coarse permissions for GPS and Network providers respectively. Permission will have coarse permission granted implicitly. We can find the last location fix determined by a particular Location Provider using the getLastKnownLocation method, passing in the name of the Location Provider. The following examplefinds the last location fix taken by the GPS provider:

```
String provider = LocationManager.GPS_PROVIDER;
Location location = locationManager.getLastKnownLocation(provider);
```

Before using this method , we need to request updates using location manager at least once. We use request Updates method for this. It takes the location provider, minimum time, minimum distance and Location Listener object as arguments.

```
lm.requestUpdates( provider, time,distance,listener);
```

We can also stop receiving updates using the remove Updates method which takes the location provider as argument.

```
lm.removeUpdates(provider);
```

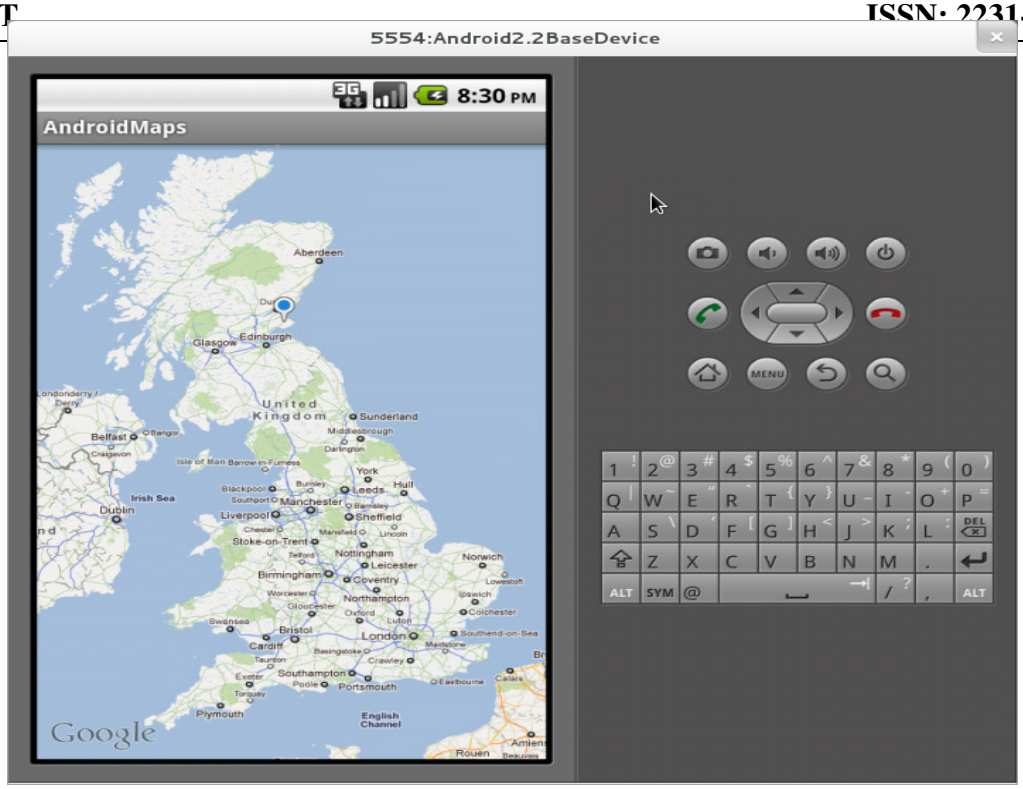


**Figure 3.** Finding our location

## 4.2. Proximity Alerts

It's often useful to have our applications react when a user moves toward, or away from, a specific location. Proximity alerts let our applications set triggers that are fired when a user moves within or beyond a set distance from a geographic location. To set a proximity alert for a given coverage area, select the center point (using longitude and latitude values), a radius around that point, and an expiry time-out for the alert. The alert will fire if the device crosses over that boundary, both when it moves from outside to within the radius, and when it moves
from inside to beyond it. When triggered, proximity alerts fire Intents, most commonly broadcast Intents. To specify the Intent to fire, we use a Pending Intent, a class that wraps an Intent in a kind of method pointer, as shown in the following code snippet:

```
Intent in = new Intent(MY_ACTION);
PendingIntent pi = PendingIntent.getBroadcast(this, -1, in, 0);
```

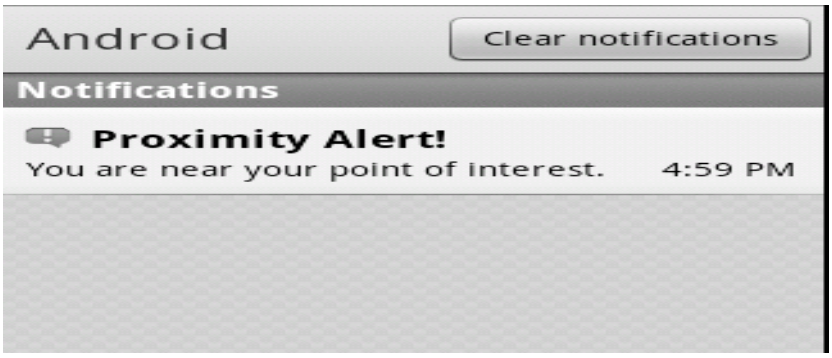


**Figure 4.** Proximity Alert

If TREASURE_PROXIMITY_ALERT is static string then to start listening for proximity alerts, we can register receiver as follows:

```
IntentFilter filter = new IntentFilter(TREASURE_PROXIMITY_ALERT);
registerReceiver(new ProximityIntentReceiver(), filter);
```

### 4.3. Geocoding and Reverse Geocoding

Geocoding lets us translate between street addresses and longitude/latitude map coordinates. This cangive us a recognizable context for the locations and coordinates used in location-based services and map-based activities. The geocoding lookups are done on the server, so our applications will require us to include an Internet uses-permission in our manifest. The Geocoder class provides access to two geocoding functions [5]:

- Forward Geocoding
- Reverse Geocoding

Forward Geocoding converts the address into latitude and longitude. Reverse Geocoding converts latitude and longitude to corresponding address.
First we create a Geocoder object using which we need to perform geocoding and reverse geocoding

```
Geocoder geocoder = new Geocoder(getApplicationContext(),Locale.getDefault());
```

For geocoding we use getFromLocationName method of Geocoder class. The following is the sample code snippet :

```
List<Address> locations = null;
Geocoder gc = new Geocoder(this, Locale.getDefault());
try {
        locations   = gc.getFromLocationName(streetAddress, 10);
    } catch (IOException e) {}
```

For reverse geocoding we use getFromLocation method of Geocoder class. The following is the sample code snippet:

```
List<Address> addresses = null;
Geocoder gc = new Geocoder(this, Locale.getDefault());
try {
        addresses   = gc.getFromLocation(latitude, longitude, 10);
} catch (IOException e) {}
```

### 4.4. Map based Activities

To use maps in our applications we need to extend MapActivity. The layout for the new class must then include a MapView to display a Google Maps interface element. The Android maps library is not a standard Android package; as an optional API, it must be explicitly included in the application manifest before it can be used. Add the library to our manifest using a uses-library tag within the application node, as shown in the following XML snippet [8]:

```
<uses-library android:name="com.google.android.maps"/>
```

To see map tiles in our Map View we need to add a <uses-permission> tag to our application manifest for INTERNET. MapView controls can be used only within an Activity that extends MapActivity. Override the onCreate method to lay out the screen that includes a MapView, and override isRouteDisplayed to return true if the Activity will be displaying routing information (such as traffic directions).By default the Map View will show the standard street map. In addition, we can choose to display a satellite view, StreetView, and expected traffic, as shown in the following code snippet:

```
mapView.setSatellite(true);
mapView.setStreetView(true);
mapView.setTraffic(true);
```

We can also query the Map View to find the current and maximum available zoom levels, as well as the center point and currently visible longitude and latitude span (in decimal degrees).

```
int m= mapView.getMaxZoomLevel();
GeoPoint center = mapView.getMapCenter();
int latS = mapView.getLatitudeSpan();
int longS = mapView.getLongitudeSpan();
```

We can also optionally display the standard map zoom controls using the setBuiltInZoomControls method.

```
mapView.setBuiltInZoomControls(true);
```

We can use the Map Controller to pan and zoom a MapView. We can get a reference to a MapView's controller using getController method .

```
MapController mapController = myMapView.getController();
```

We can re-center and zoom the Map View using the setCenter and setZoom methods available on the Map View'sMapController.

```
mapController.setCenter(geopointobject);
mapController.setZoom(1);
```

### 4.5. Creating and using Overlays

Overlays enable us to add annotations and click handling to MapViews. Each Overlay lets us draw 2D primitives, including text, lines, images, and shapes, directly onto a canvas, which is then overlaid onto a Map View [6].

We can add several Overlays onto a single map. All the Overlays assigned to a Map View are added as layers, with newer layers potentially obscuring older ones. User clicks are passed through the stack until they are either handled by an Overlay or registered as clicks on the Map View itself.

Each Overlay is a canvas with a transparent background that is layered onto a Map View and used to handle map touch events. To add a new Overlay, we create a new class that extends Overlay. We override the draw method to draw the annotations we want to add, and override onTap to react to user clicks (generally made when the usertaps an annotation added by this Overlay).


**Figure 5.** Overlays on Maps

## V. DISCUSSION

In Android, a wide variety of applications can be developed in the field of Location Based Services. One such application is "Finding the route between two locations". For preparing this application we first we need three activities namely

- Activity requesting source address from user.
- Activity requesting destination address from user.
- Map Activity which should display the actual route between the two locations specified in above activities.

For preparing the user interface of the activities requesting source and destination addresses, we need to use the corresponding XML files of the activities. Once the user interface of these two activities is ready we need to design the user interface of the Map Activity. As this activity should hold the Google Map, the procedure illustrated in the above section should be followed for the access to Google Map API. Then we should create two explicit intents (which are generated when an event is performed by user like clicking a button) one for communication between the source (passing the source address to destination activity) and destination and other for communication between destination and Map Activity (passing the source and destination addresses to the Map Activity).The overall stepwise procedure for developing this application is mentioned below

- Preparing map resource and internet access for it
- Acquiring KML route file from google
- Drawing the Path using the following small procedure

  i) Building URL from source to destination

  ii) Connecting to URL and creating DocumentBuilder for parsing the KML file

  iii) Splitting each point in the path and drawing each line on the map
- Drawing points and lines on the Map

## VI. RESULTS

The application thus developed using the above procedure helps to find the route between the two locations. The first activity which takes the source address from the user is shown in the following figure.

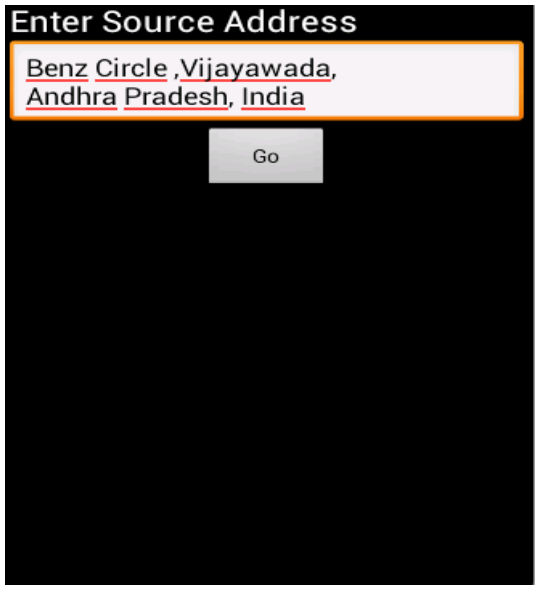


**Figure 6**. Source Activity

For instance if the source address is given as “Benz Circle, Vijayawada, Andhra Pradesh, India”, after clicking the “Go” button explicit intent loads the second activity which requests the destination address to be entered by the user. The second activity is shown in the following figure

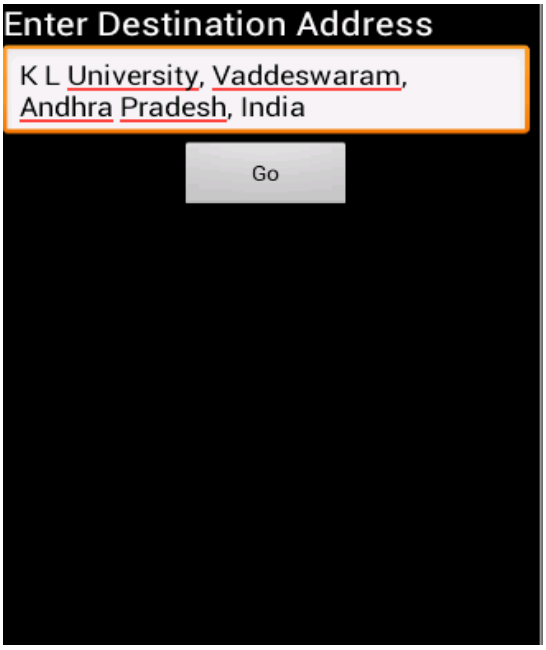


**Figure 7.** Destination Activity

For instance if the destination address is given as "K L University, Vaddeswaram, Andhra Pradesh, India", after clicking the "Go" button explicit intent loads the MapActivity which shows the route between these two locations. The MapActivity thus loaded is shown in the following figure.


**Figure 8.** MapActivity displaying route

In the above figure the blue circle indicates the source and the green circle indicates the destination and the green curved line indicates the route between these locations.

## VII. CONCLUSION

Location Based Services are those services which provide both information and entertainment and are accessible with mobile devices through the mobile network. They utilize the ability to make use of the geographical position of the mobile device. They can utilize multiple technologies such as the GPS satellite network, cellular networks, Wi-Fi networks and other technologies. Context awareness is the excellent feature for the LBS. Location Based Services can be used in a variety of aspects like vehicle tracking, monitoring driving habits, locating our employees, finding the route between two places or to find the route to a specified location from current position etc. All of these need the use of GPS along with some tracking programs. Android provides a very nice platform for developing LBS applications. It provides separate methods and classes for each and every separate entity involved in

the development of the application. Location Based Services thus help the users in a variety of aspects and thus has a greater scope of development in the near future.

# VIII. FUTURE SCOPE

After performing a detailed survey it is observed that obtaining user location from a mobile device can be complicated. There are several reasons why a location reading (regardless of the source) can contain errors and be inaccurate. Some sources of error in the user location include:

## 8.1. Multitude of Location Sources

GPS, Cell-ID, and Wi-Fi can each provide a clue to user's location. Determining which to use and trust is a matter of trade-offs in accuracy, speed, and battery-efficiency.

## 8.2. User movement

Because the user location changes, you must account for movement by re-estimating user location every so often.

## 8.3. Varying accuracy

Location estimates coming from each location source are not consistent in their accuracy. A location obtained 10 seconds ago from one source might be more accurate than the newest location from another or same source. These problems can make it difficult to obtain a reliable user location reading. This document provides information to help you meet these challenges to obtain a reliable location reading. It also provides ideas that you can use in your application to provide the user with an accurate and responsive geo-location experience.

The future work is to eliminate these problems and provide an efficient way to find out the location of user accurately.

# ACKNOWLEDGEMENTS

We thank our principal Prof. K. Rajsekhar Rao for providing guidance in understanding the concepts. We also want to thank our beloved head of the computer science department, Prof. S. Venkateswarulu for providing valuable information regarding the various categories of the applications that can be developed in Android.

# REFERENCES

[1] Chris Haseman, Android Essentials, firstPress, 9_17, 2008
[2] Cláudio Bettini,Sushil Jajodia,Pierangela Samarati, X.Sean Wang, Privacy in Location-Based applications, Springer,1_2,2009
[3] Georg Gartner, Felix Ortag, Advances in Location-Based Services, Springer, 4_5, 2011
[4] James Steele, Nelson To,Shan Conder,Lauren Darcey,The Android Developer's Collection, Addison-Wesley Professional, 30A_40A, 2012.
[5] Jerome(J. F.) DiMarzio, Android - A Programmer's guide, Tata McGraw-Hill Education, 203_237,2010
[6] Mark L. Murphy, The busy coder's guide to Android development, CommonsWare, 287_292,2008
[7] Reto Meier, Professional Android 2 Application Development,Wrox, 247_264, 2010
[8] [Online] chengalva.com
[9] [Online] developer.android.com

**Authors**

**Ch. Radhika Rani** is a Lecturer of CSE Dept. in K L University. She is very good at programming and is well versed in subjects like Microprocessors, Embedded Systems, Digital Logic Design

**A. Praveen Kumar** is a student studying final year of B. tech in K L University with CSE as his specialization. He has built some interesting apps in Android.

**D. Adarsh** is a student studying final year of B. tech in K L University with CSE as his specialization. He has keen interest in the area of Android app development.

**K. Krishna Mohan** is a student studying final year of B .tech in K L University with CSE as his specialization. He developed many Android applications.

**K. Venkata Kiran** is a student studying final year of B. tech in K L University with CSE as his specialization. He has built some interesting apps in Android.