目的地検索のための Android 地図アプリケーション作成に関する初歩的研究

島根大学　総合理工学部　数理・情報システム学科

計算機科学講座　田中研究室

s083078　藤原 裕史

目次

第1章　序論

近年、携帯電話にパソコンや携帯情報端末(PDA)の機能を持ち合わせた、スマートフォンが普及して来ている。スマートフォンの持つ機能は Web の閲覧や電子メールの送受信、文章の作成とその閲覧、音楽・動画の再生などがある。また、スマートフォンはインターネットを通じてそのスマートフォンに搭載されている OS に対応したアプリケーションをダウンロードして追加することが可能である。そのため様々なアプリケーションが公開や販売されている。今後もスマートフォンの需要は高まることが予想される。

本研究ではスマートフォンに搭載されている OS のうち、Android に着目した。Android の開発ツールは無償で提供されているため、すぐに Android アプリケーションの開発をはじめることができる。また、Google の Web サービスとの親和性が高いため作り手次第では優れたアプリケーションも開発可能である。この様な点から本研究では、Google マップを用いた、Android 用の地図アプリケーションの開発を目的とした。

第2章　Android

2.1　Android とは

Android とは、Google が中心となって開発を進めている Linux ベースの携帯端末用プラットフォームである。Android プラットフォームには OS だけでなく、ミドルウェア、ユーザーインタフェース、標準アプリケーション(Web ブラウザ、メーラー等)が含まれている。

Android の特徴としては、以下が挙げられる[1]。

- さまざまなメーカーから Android 対応のハードウェアがリリース
  携帯端末用の OS には多額の費用がかかり、それを利用するためのライセンス料も少なくない。それに対し Google は、Android プラットフォームを無償で提供している。さらに、Android のソースコードはオープンソースとして公開されている。

- 世界中へのアプリ配布が容易
  Google は世界規模の携帯アプリケーション市場「Android Market」をオープンしている。これにより日本にいながら、世界中の Android 端末のユーザーにアプリケーションを販売することができる。
  また、開発ツールは無償で提供されているので、誰でもすぐに Android アプリケーションの開発を始めることができる。

- Google の Web サービスとの親和性が高い
  Google は Google マップや Gmail、Youtube などの非常に優れた Web サービスを提供しているが、Android にはそれらを利用するためのコンポーネントが標準装備されている。このことより、アイデアと組み合わせ次第で、より優れたアプリケーションが作れるようになる。

2.2　Android のアーキテクチャ

Android のアーキテクチャは、多くのコンポーネントで構成されている。これらのコンポーネントは「アプリケーション」、「アプリケーションフレームワーク」、「ライブラリ」、「Android ランタイム」、「Linux カーネル」の 5 つのレイヤーに分類される[1]。

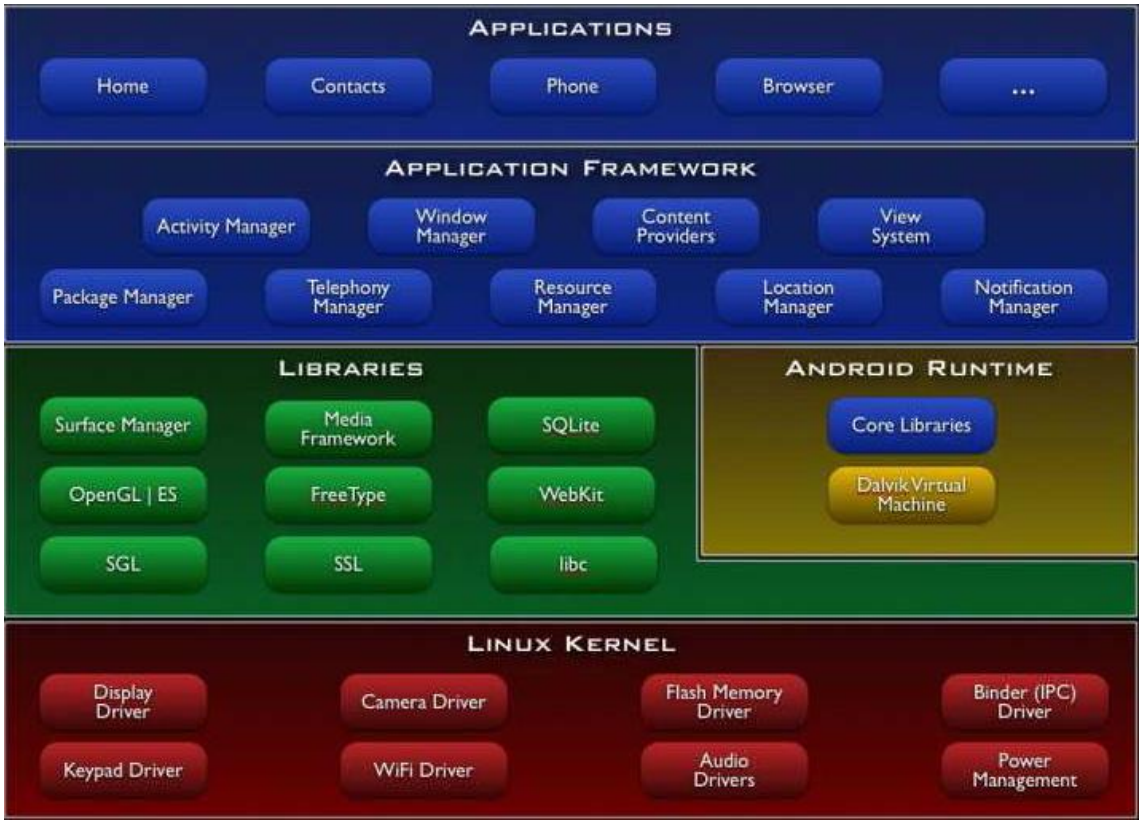

図 2.2.1　Android を構成するコンポーネントとアーキテクチャ[2]

「アプリケーション」は、Android アプリケーションのレイヤーである。Home、Contacts、Phone、Browser などの Android アプリケーションがこのレイヤーのコンポーネントとなる。

「アプリケーションフレームワーク」は Android アプリケーションで利用される API 規定しているレイヤーである。このレイヤーの主要コンポーネントは以下の通りである。

コンポーネント	機能
Activity Manager	アプリケーションのライフサイクルを管理
Window Manger	ウィンドウを管理
Content Providers	アプリケーション間のデータ共有を管理
View System	ユーザーインタフェースを管理
Notification Manager	ステータスバーへのアラート表示を管理
Package Manager	インストールを管理
Telephony Manager	通話機能を管理
Resource Manager	リソースを管理
Location Manager	位置情報を管理

表 2.2.1　アプリケーションフレームワークを構成する主要コンポーネント

「ライブラリ」は、複数のアプリケーションから汎用的に利用される機能をまとめたレイヤーである。アプリケーションからはアプリケーションフレームワークを経由して利用する。このレイヤーの主要コンポーネントは以下の通りである。

コンポーネント	機能
Surface Manager	複数アプリケーション間の 2D/3D グラフィックスを合成するライブラリ
Media Framework	ビデオ形式の再生と記録のライブラリ
SQLite	リレーショナルデータベースのライブラリ
OpenGL/ES	3D グラフィックスエンジン
Free Type	ビットマップとベクターフォントのレンタリングを行うライブラリ
WebKit	ブラウザ表示を行うための HTML レンタリングエンジン
SGL	2D グラフィックスエンジン
SSL	SSL のライブラリ
libc	標準的な C 言語ライブラリ

表 2.2.2　ライブラリを構成する主要コンポーネント

「Android ランタイム」は、Android の Java 仮想マシンのレイヤーである。

Android では、プログラミング言語として Java 言語を利用する。そして、Android はサン・マイクロシステムズの Java ME の仮想マシンでなく、独自の仮想マシンである Dalvik 仮想マシンが採用されている。ソースコードを Dalvik バイトコードと呼ばれる中間言語に変換し、実行時に Dalvik 仮想マシンによって、ベースとなる OS である Linux で実行可能な形式に変換して実行する方法をとっている。

このレイヤーの主要コンポーネントは以下の通りである。

コンポーネント	機能
Core Libraries	Java 言語に準拠したコアライブラリ機能
Dalvik 仮想マシン	.dex フォーマットのバイトコードを実行する仮想マシン

表 2.2.3　Android ランタイムを構成する主要コンポーネント

「Linux カーネル」は、Android が Linux ベースの OS のため、ハードウェアに一番近いレイヤーとなっている。

2.3　Android アプリケーションの構成要素

Android アプリケーションは、「アクティビティ」、「ブロードキャストレシーバー」、「サービス」、「コンテンツプロバイダ」の 4 つの要素から成り立つ[1]。

本研究では、4 つの要素のうち「アクティビティ」のみを用いてアプリケーションの作成を行っている。

2.3.1　アクティビティ

アクティビティは、ユーザーインタフェースの提供やイベント処理など、ユーザーとアプリケーション間のやり取りを仲介するオブジェクトである。アプリケーションが自ら保持するアクティビティだけでなく、他のアプリケーションが保持するアクティビティを利用することも可能である。他のアプリケーションを起動するだけでなく、他のアプリケーションの任意の画面(アクティビティ)を開き、その結果を元のアプリケーションで受け取ることができる。

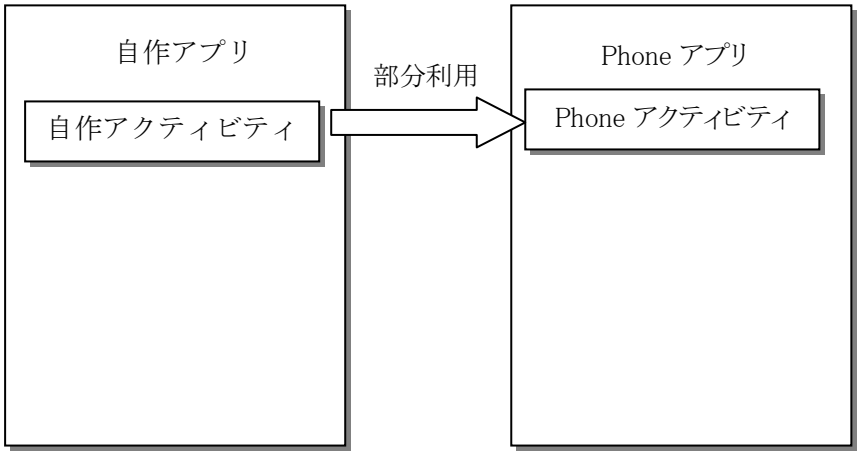

図 2.3.1　アクティビティから他のアプリのアクティビティを呼び出す

2.3.2　ブロードキャストレシーバー

ブロードキャストレシーバーは、他のアプリケーションや Android OS から投げられた(ブロードキャストされた)インテントを受信し、各種処理を行うオブジェクトである。

インテントとは、アクティビティやサービスがやりとりを行う時に利用するメッセージオブジェクトのことである。

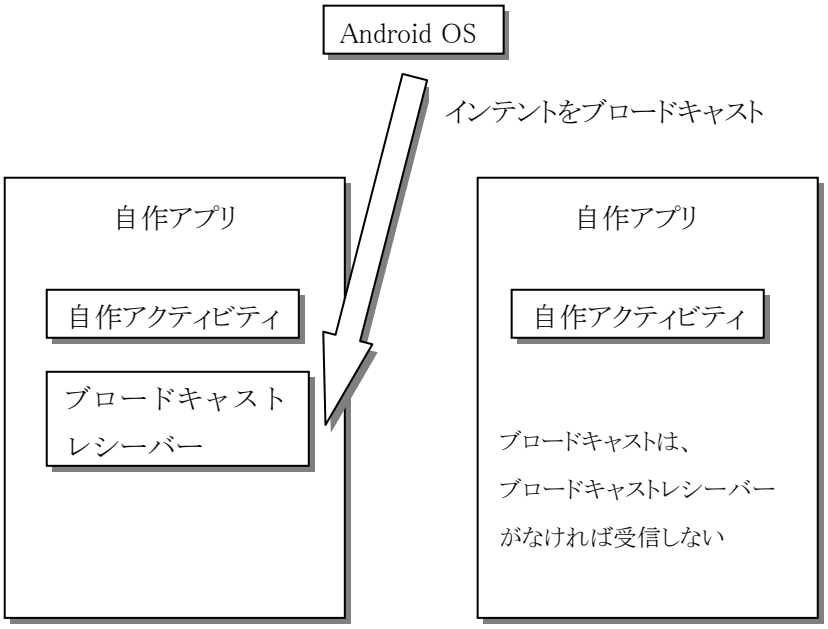

図 2.3.2　ブロードキャストレシーバーを実装することで、メッセージオブジェクトを受信する

2.3.3　サービス

サービスは、画面表示とは独立してバックグラウンドで処理を実行し続けることが可能なオブジェクトである。画面を持たないので、ユーザーとのやりとりはノンティフィケーションやトーストといった特殊なインタフェースを通じて行う。

2.3.4　コンテンツプロバイダ

コンテンツプロバイダは、データベースのデータを他のアプリケーションに提供するオブジェクトである。アプリケーション間のデータのやりとりは、インテントのパラメータでも行えるが、大量のデータをやりとりする時はコンテンツプロバイダを使う方が便利である。

2.4　開発環境

Android アプリケーションを開発するために開発環境を整える必要がある。

Android アプリケーションは Java 言語で開発するため、Java Development Kit (JDK) 6.0[3]をダウンロードおよびインストールを行った。

次に、Eclipse 3.6(Helios)[4]をダウンロードし、インストールを行った。Eclipse は Java 言語に対応したオープンソースの統合開発環境である。そして、Eclipse で Android アプリケーションを開発するために Eclipse のメニューから Android Development Tools (ADT)[5]プラグインをインストールし、Android 用のプロジェクトの作成、ビルド、エミュレータの起動を行えるようにした。

また、Android アプリケーションの開発キットである Android SDK[6]をダウンロードし、Eclipse からインストールを行った。

2.5　アプリケーション開発の手順

Android 上で動作するアプリケーションを開発するための手順を簡単に説明する[7]。

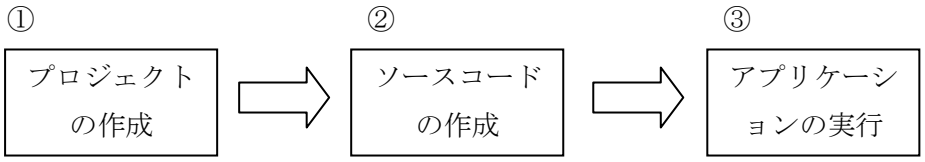

図 2.5.1　アプリケーション開発の手順

① プロジェクトの作成

Androidアプリケーションは、Eclipse上でAndroid用のプロジェクトを使用して開発を行う。プロジェクトはアプリケーションの単位で作成する。Eclipse の機能を使って、新規に作成することもでき、独自に雛形となるプロジェクトを作成して、Eclipse 上に読み込んで使用することもできる。

② ソースコードの作成

Android プロジェクトを新規に作成した場合、プロジェクトの中には HellowWorld が表示されるアプリケーションの雛形が含まれる。つまり、Eclipse で作成されるプロジェクトは、画面を使用するタイプのアプリケーションとなる。

③ アプリケーションの実行

アプリケーションが完成したら、Eclipse の実行/デバッグダイアログから指示することで、ADTが以下の作業を内部的に行う。

1. Java プログラムのコンパイル
2. Java バイトコード(.class) を Android バイトコード(.dex) に変換
3. Android アプリケーションをパッケージング(.apk ファイルの作成)
4. Android エミュレータの起動
5. アプリケーションを Android エミュレータにインストール
6. アプリケーションを Android エミュレータ上で起動

上記1～6の作業が正常に完了した場合、Androidエミュレータ上でアプリケーションが動作する。

第3章　アプリケーションの設計

3.1　アプリケーションの概要

本研究で作成したアプリケーションは、Google マップ上に現在地と目的地を画像イメージで表示し、また、現在地と目的地の 2 点間の直線距離を求め、表示する地図アプリケーションを作成した。

アプリケーションの処理を図 3.1.1 で示す。

アプリケーションを実行すると、GPS でアプリケーションを実行している位置情報を取得する。位置情報を取得すると、取得した位置情報から現在地として設定し、Google マップ上に画像イメージを描画する。

現在地を描画したら次に目的地を設定する。アプリケーション起動時には目的地は設定されていないため、アプリケーション上のテキストボックスに住所または地名を入力して、検索ボタンを押す。検索ボタンを押すと目的地として設定し、Google マップ上に画像イメージを描画する。また、入力した地名の候補が複数あった場合は、リストでその候補を表示してそのリストから目的地を選べるようになっている。

現在地と目的地を描画すると、現在地と目的地の 2 点を結ぶ直線を描画し、その直線距離を現在地の画像イメージ付近に描画する。

移動しながらアプリケーションを実行する場合は、位置情報が変化するため再び現在地の位置情報を取得し、現在地の再設定と再描画する。目的地は、目的地を変更しない限り変わらないので再描画は行わない。現在地を再描画したら、直線と直線距離も再描画する。

また、アプリケーションの設計にはhttp://codezine.jp/article/detail/4878[8]を参考とした。

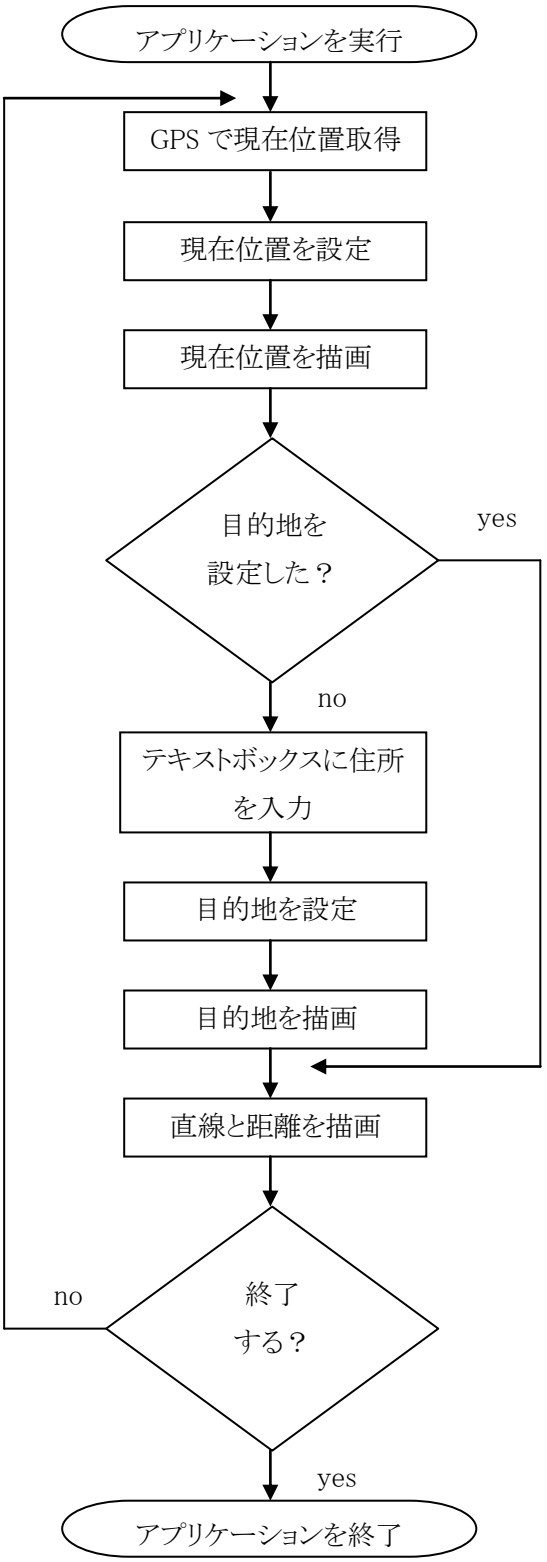

図 3.1.1　アプリケーションの処理の流れ

3.2　Google マップの表示

Google マップをアプリケーションで表示するには、MapView が必要となる。MapView は Google マップを表示するコンポーネントである。MapView を利用するためには、Android Maps API Key を取得する必要がある。Android Maps API とは Google マップを表示するコンポーネントの MapView を Android 上で実現するための API である。

3.2.1　Android Maps API Key の取得

Android Maps API Key の取得には、まず Google アカウントが必要であるため、Google アカウントを取得した。そして、JDK の keytool コマンドで証明書を作り、Sign Up for the Android Maps API のサイトでフィンガープリント(MD5)を入力し、Android Maps API Key を取得した。

3.2.2　MapView のレイアウトの設定

Google マップの表示をアプリケーションで行うために、レイアウトファイルにリスト 3.2.1 のように MapView の定義を追加した。

```
<com.google.android.maps.MapView
  android:id="@+id/mapV"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:clickable="true"
  android:apiKey="APIKey"/>
```

リスト 3.2.1　MapView のレイアウトファイルの設定

Map View の属性は以下のようになっている。

属性	機能
android:id	プログラムからビューを参照するための ID。
android:layout_width	ビューの幅を設定する。
android:layout_height	ビューの高さを設定する。
android:clickable	“true”を設定することで画面のスクロールが可能となる。
android:apiKey	取得した Android Maps API Key を設定する。

表 3.2.1　Map View の属性

3.2.3　AndroidManifest.xml の設定

Android Maps API を利用するために AndroidManifest.xml にパーミッションを設定した。

Map View や Map Activity のクラスは com.google.android.maps のパッケージ属している。これらのクラスは Android のライブラリとは別に Google が提供しているライブラリであるため、com.google.android.maps パッケージのライブラリを利用する宣言をしなければならない。

また、Map View は地図の画像をインターネットから取得するため、アプリケーションがインターネットに接続する許可を設定する必要がある。リスト 3.2.2 が Android Maps API を利用するための AndroidManifest.xml の設定の中身である。

```
・・・・・・略・・・・・・
<!-- Googleマップのライブラリを利用する宣言 -->
    <uses-library android:name="com.google.android.maps" />
</application>

<!-- インターネットへのアクセス権限 -->
<uses-permission android:name="android.permission.INTERNET" />
```

リスト 3.2.2　AndroidManifest.xml の設定

3.2.4　プログラムの作成

アプリケーション内で Google マップを表示させるためのプログラムの例をリスト 3.2.3 で示す。

```
public class MyMapActivity extends MapActivity{
        MapView mapV = null;
        MapController mc = null;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //xmlからレイアウトを作成
        setContentView(R.layout.main);
        mapV = (MapView)findViewById(R.id.mapV);
        mc = mapV.getController();
        mc.setZoom(18);
        mapV.setBuiltInZoomControls(true);
……略……
    @Override
    protected boolean isRouteDisplayed() {
       return false;
    }
```

リスト 3.2.3　Google マップを表示するプログラム

MapView を配置した画面を表示するアクティビティは MapActivity を継承して作成する。そして、アプリケーション起動時に呼び出される onCreate メソッド内で MapView のレイアウトを設定した xml ファイルからレイアウトを作成する。MapController クラスの setZoom メソッドでマップの拡大や縮小のサイズを指定している。setBuiltInZoomControls メソッドでズームコントロールの表示を行っている。また、isRouteDisplayed メソッドは MapActivity を継承する場合、実装する必要がある。

3.3　現在地の表示

現在地を Google マップ上に表示するためには、まず現在位置を取得しなくてはならない。

Android には位置情報を取り扱う機能がロケーション API として提供されている。提供されている機能は、人工衛星の電波により詳細な位置を取得できる「GPS(Global Positioning System)」と通信を行う基地局の位置情報から位置を取得する「ネットワーク」の 2 つがある。本研究では、GPS を用いて現在位置を取得することにした。

3.3.1　AndroidManifest.xml の設定

Android Maps API と同様にロケーション API で GPS を利用できるように AndroidManifest.xml にパーミッションを設定した。実際の設定例をリスト 3.3.1 に示す。

```
<!-- 位置情報のアクセス権限 -->
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<!-- エミュレータをDDMSのLocationControlsで操作する権限 -->
<uses-permission
android:name="android.permission.ACCESS_MOCK_LOCATION"/>
```

リスト 3.3.1　AndroidManifest.xml の設定

「android.permission.ACCESS_FINE_LOCATION」が GPS で位置情報を取得するためのハードウェアへのアクセス権限である。

また、「android.permission.ACCESS_MOCK_LOCATION」は、エミュレータによる位置情報取得を行う権限である。本研究では、アプリケーションの実行を実機ではなくエミュレータ上で行ったため、この権限が必要となった。

3.3.2　現在位置の取得

アプリケーションから現在位置を取得するには、LocationListener というリスナの実装と位置情報を管理する LocationManager に登録する必要がある。

LocationListener の実装はリスト 3.3.1、LocationManager への登録はリスト 3.3.2 でそれぞれ実装例を示す。

LocationListener では位置情報が変更されたときに LocationListener 内の onLocationChanged メソッドが呼ばれる。onLocationChanged メソッドでは、緯度と経度の値が Location クラスのインスタンス location に格納され、引数として受け取り、リスト 3.3.3 で示す setCurrent メソッドに変数 location を渡す。そして、setCurrent メソッドで現在位置の設定を行う。

```
private class locationListener implements LocationListener{
          @Override
          public void onLocationChanged(Location location) {
              double d = 0;
              setCurrent(location);
```

リスト 3.3.1　LocationListener の実装

LocationListener を実装したら、作成した LocationListener を LocationManager に登録する。位置情報を管理する LocationManager は getSystemService メソッドを呼び出して取得する。LocatonManager を取得したら、LocationManager クラスの requestLocationUpdates メソッドで登録して位置情報の通知を開始する。requestLocationUpdates メソッドの引数には、プロバイダーの種類、通知のための最小時間間隔と最小距離間隔、そして LocationListener が入る。プロバイダーは先に述べたように、GPS を用いるため、GPS_PROVIDER としている。時間間隔と距離間隔は頻繁に位置情報を取得できるようにしている。

また、LocationListener を一度でも行うと、アプリケーションが背面に行っている間も位置情報が通知される。そのため端末のバッテリーを無駄に消費してしまう。これを防ぐために LocationListener の登録は onResume メソッドで行っている。onResume メソッドはアプリケーションが前面に来たときに呼び出されるメソッドである。そして、アプリケーションが背面に行くときに呼び出される onPause メソッド内で removeUpdates メソッドを呼び出し、LocationListener を解除している。例えば、アプリケーションを実行中に通話画面に変わったら onPause メソッドで LocationListener を解除し、通話画面からアプリケーションの画面に戻ったら onResume メソッドで LocationListener を登録するような形になっている。

```
private LocationManager lm = null;
protected locationListener locationListener = null;

@Override
public void onResume(){
// 位置情報管理クラスの取得
        lm = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
        locationListener = new locationListener();
        // 最終地点を取得
        Location location =
        lm.getLastKnownLocation(LocationManager.GPS_PROVIDER);
        setCurrent(location);
        lm.requestLocationUpdates(LocationManager.GPS_PROVIDER,0,0,loca
        tionListener);
        super.onResume();
}
@Override
public void onPause(){
        lm.removeUpdates(locationListener);
        lm = null;
        locationListener = null;
        super.onPause();
}
```

リスト3.3.2　LocationListenerの登録

3.3.3　現在地の設定と描画

現在地の設定するためにsetCurrentメソッドを実装した。setCurrentメソッドは現在地が変更されたことをLocationListenerが受け取ると呼び出される。リスト3.3.3が実装例である。

setCurrentメソッドでは、取得した位置情報を地図上に反映させるためにLocationクラスのインスタンスが保持していた緯度と経度をGeoPointクラスのインスタンスに変換する。GeoPointクラスでは緯度と経度を10の6乗倍、つまりIE6倍したものを保持する。そのため緯度×IE6、経度×IE6という処理を行う。GeoPointに変換したら、取得した位置情報の場所が地図の中心に来るようにする。

また、setCurrentメソッドでは地図上に現在地の場所を画像イメージで示すために、currentLocationOverlayクラスのメソッドを呼び出している。CurrentLocationOverlayクラスはオーバーレイという領域を設けて、地図上に任意の画像等を描画することを可能にするOverlayクラスを継承したクラスである。

currentLocationOverlayクラスではGeoPointの値を元に地図上に画像を描画する。どのように描画するかは、currentlocationOverlayクラスのdrawメソッドで実装する。リスト3.3.4が実装例である。

```
  protected void setCurrent(Location location){
        if(location == null){
            return;
        }
        // LocationからGeoPointへ変換
        Double latitude = location.getLatitude() * 1E6;
        Double longitude = location.getLongitude() * 1E6;
        GeoPoint geoPoint = new GeoPoint(latitude.intValue(),
         longitude.intValue());

        //地図の中央に設定
            mc.setCenter(geoPoint);

        // 目標値の座標にイメージを描画
        if( currentLocationOverlay == null){
            // 初回はOverlayを作成
            Bitmap bmp =
                BitmapFactory.decodeResource(getResources(),
                R.drawable.current);
            currentLocationOverlay = new CurrentLocationOverlay(bmp);
            currentLocationOverlay.setGeoPoint(geoPoint);
            // Overlayの追加
            mapV.getOverlays().add(currentLocationOverlay);
        }
        else{
            // 2回目以降はOverlayを再利用
            currentLocationOverlay.setGeoPoint(geoPoint);
            // 無効にして再描画
            mapV.invalidate();
        }
}
```

リスト3.3.3　setCurrentの実装

```
public class CurrentLocationOverlay extends Overlay {
        // 描画するイメージ
        private Bitmap bmp;
        // 描画する座標
        private GeoPoint geoPoint;

        public CurrentLocationOverlay(Bitmap bmp) {
            this.bmp = bmp;
        }

        public GeoPoint getGeoPoint() {
            return geoPoint;
        }

        public void setGeoPoint(GeoPoint geoPoint) {
            this.geoPoint = geoPoint;
        }
        @Override
        public void draw(Canvas canvas, MapView mapView, boolean shadow)
            {
            // shadow = true,falseで2回呼び出される
            if( !shadow){
                // Mapの表示位置と座標から画面の描画位置を算出
                Projection pro = mapView.getProjection();
                Point p = pro.toPixels(geoPoint,null);

                // 中心がくるように調整
                int centerX = p.x - bmp.getWidth() / 2;
                int centerY = p.y - bmp.getHeight() / 2;

                // 画像の描画
                canvas.drawBitmap(bmp, centerX, centerY, null);
            }
        }
    }
```

リスト3.3.4　CurrentLocationOverlayの実装

3.4　目的地の表示

地図上に目的地を表示させるために、Geocoderクラスを目的地の設定に利用している。Geocoderは緯度、経度、住所などの情報を保持しているAddressクラスのリストを取得するために利用ものである。

目的地の地名または住所をアプリケーションから取得するために、EditTextを実装した。EditTextはテキストを入力するためのウィジェットである。また、目的地設定のためにsetDestメソッドを実装した。

3.4.1　EditText のレイアウト設定

Geocoderを実装して、地名や住所から目的地を設定するにはアプリケーションにテキストを入力するためのユーザーインタフェースが必要となる。EditTextを利用してテキストを取得するためにレイアウトを設定した（リスト3.4.1）。

```
<EditText
        android:id="@+id/editText"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
android:layout_weight="2.12">
  </EditText>

  <!-- 送信用のボタン -->
   <Button
        android:id="@+id/button1"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="@string/button1"/>
```

リスト3.4.1　EditTextのレイアウト設定

3.4.2　目的地の設定と描画

Geocoderを利用して地名や住所から緯度、経度を取得するプログラムは、リスト3.4.2_1～3.4.2_3に示している。アプリケーションから入力された住所や地名はGeocoderクラスのgetFromlocationNameメソッドでAddressクラスのリストに変換され、このAddressのリストから位置情報を取得する。取得した位置情報はsetDestメソッドに渡し、目的地として設定する。setDestメソッドの実装例はリスト3.4.3に示す。setDestメソッドは、現在地を設定するためのsetCurrentメソッドとほぼ同様の処理を行っている。また、目的地の候補が複数あった場合はダイアログから選択して選べるように、AddressSelectionListenerを実装した。(リスト3.4.4)

目的地の描画は、現在地の描画と同様にOverlayクラスを継承したDestinationOverlayを実装し、描画している。(リスト3.4.5)

```
EditText editText=(EditText)findViewById(R.id.editText);
Geocoder geocoder =
      new Geocoder(MyMapActivity.this,Locale.getDefault());
 try{
      List<Address> addressList =
      geocoder.getFromLocationName(editText.getText().toString(),10);
      Address address = null;

      //目的地の候補がなかった場合
     if(addressList.isEmpty()){
      String message =
              getResources().getString(R.string.cannot_get_address);
      Toast toast = Toast.makeText(MyMapActivity.this, message + "[" +
              editText.getText().toString() + "]", Toast.LENGTH_SHORT);
      toast.show();
     }

     //目的地の候補が１つだった場合リストからその情報を取得
     else if(addressList.size() == 1){
              address = addressList.get(0);
     }
```

リスト3.4.2_1 Geocoderを利用した緯度、経度の取得(1)

```
else{
 // 複数のアドレスが見つかった場合はダイアログで対象を選択する
 AlertDialog.Builder alertDialogBuilder = new
       AlertDialog.Builder(MyMapActivity.this);
 alertDialogBuilder.setTitle(R.string.select_target);
 // ダイアログ表示用の文字列の生成
 List<String> strAddressList = new ArrayList<String>();
 for( Address element: addressList){
     int maxAddressLineIdx = element.getMaxAddressLineIndex();
     strAddressList.add(element.getAddressLine( maxAddressLineIdx));
 }
  // ダイアログに表示するリストの生成
 AddressSelectionListener listener =
       new AddressSelectionListener( addressList);
 ListView listView = new ListView(MyMapActivity.this);
 ArrayAdapter<String> listAdaptor =
       new ArrayAdapter<String>(MyMapActivity.this,
              android.R.layout.simple_list_item_1, strAddressList);
 listView.setAdapter(listAdaptor);
 listView.setOnItemClickListener( listener);
 // ダイアログにリストを生成
 alertDialogBuilder.setView(listView);
 // ダイアログの表示
 AlertDialog dialog = alertDialogBuilder.create();
 listener.setDialog(dialog);
 dialog.show();
}
```

リスト3.4.2_2　Geocoderを利用した緯度、経度の取得(2)

```
//目的地の候補が１つだった場合目的地に設定
if( address != null){
  setDest(address);
  double d = 0;
  //現在地が描画されているならば
  if(currentLocationOverlay != null){
    //直線が描画されていなかったら
    if(lineOverlay == null){
       GeoPoint geoPointFrom = currentLocationOverlay.getGeoPoint();
       GeoPoint geoPointTo = destLocationOverlay.getGeoPoint();
       hubeny = new Hubeny(geoPointFrom,geoPointTo);
       d = hubeny.getDistance();
       lineOverlay = new LineOverlay(geoPointFrom,geoPointTo);
       lineOverlay.setDistance(d);
       mapV.getOverlays().add(lineOverlay);
   }
   else{
        GeoPoint geoPointFrom = currentLocationOverlay.getGeoPoint();
        GeoPoint geoPointTo = destLocationOverlay.getGeoPoint();
        hubeny.setStartPoint(geoPointFrom);
        hubeny.setEndPoint(geoPointTo);
        d = hubeny.getDistance();
        lineOverlay.setGeoPoint(geoPointFrom,geoPointTo);
        lineOverlay.setDistance(d);
        mapV.invalidate();
     }
   }
  }
//例外処理
}catch(IOException e){
   Toast toast = Toast.makeText(MyMapActivity.this,
        R.string.cannot_get_address, Toast.LENGTH_SHORT);
   toast.show();
 }
```

リスト3.4.2_3　Geocoderを利用した緯度、経度の取得(3)

```
protected void setDest(Address address){
        // AddressからGeoPointに変換
        int latitude = (int)(address.getLatitude() * 1E6);
        int longitude = (int)(address.getLongitude() * 1E6);
        GeoPoint distPoint = new GeoPoint(latitude,longitude);

        // 目的地をマップの中心にする。
        mc.animateTo(distPoint);
        // ズームをデフォルト設定
        mc.setZoom(18);
 // 目標値の座標にイメージを描画
 if(destLocationOverlay == null){
  // 初回はOverlayを作成
  Bitmap bmp =
        BitmapFactory.decodeResource(getResources(),R.drawable.pushpin)
        ;
  destLocationOverlay = new DestinationOverlay(bmp);
  destLocationOverlay.setGeoPoint(distPoint);
  // Overlayの追加
  mapV.getOverlays().add(destLocationOverlay);
 }
  else{
   // 2回目以降はOverlayを再利用
   destLocationOverlay.setGeoPoint(distPoint);
   // 無効にして再描画
   mapV.invalidate();
  }
}
```

リスト3.4.3　setDestの実装

```
public class AddressSelectionListener implements OnItemClickListener{
       private List<Address> listAddress = null;
       private AlertDialog dialog = null;
       private Address address = null;
       private AddressSelectionListener( List<Address> listAddress){
          this.listAddress = listAddress;
       }
       public AlertDialog getDialog() {
          return dialog;
       }
       public void setDialog(AlertDialog dialog) {
          this.dialog = dialog;
       }
       public void onItemClick(AdapterView<?> parent,View view,int
              position,long id) {
              double d = 0;
       // 選択された住所を目的地として設定
              address = listAddress.get(position);
              setDest(address);
              if(lineOverlay == null){
                     GeoPoint geoPointFrom =
                            currentLocationOverlay.getGeoPoint();
                     GeoPoint geoPointTo =
                            destLocationOverlay.getGeoPoint();
              hubeny = new Hubeny(geoPointFrom,geoPointTo);
              d = hubeny.getDistance();
              lineOverlay = new LineOverlay(geoPointFrom,geoPointTo);
              lineOverlay.setDistance(d);
              mapV.getOverlays().add(lineOverlay);
              }
              else{
                     GeoPoint geoPointFrom =
                     currentLocationOverlay.getGeoPoint();
                     GeoPoint geoPointTo =
                     destLocationOverlay.getGeoPoint();
                     hubeny.setStartPoint(geoPointFrom);
```

```
                    hubeny.setEndPoint(geoPointTo);
                    d = hubeny.getDistance();
                    lineOverlay.setGeoPoint(geoPointFrom,geoPointTo);
                    lineOverlay.setDistance(d);
                    mapV.invalidate();
                }
            // ダイアログを閉じる
            dialog.dismiss();
        }
}
```

リスト3.4.4　AdressSelectionListenerの実装

```
public class DestinationOverlay extends Overlay {
	// 描画するイメージ
	private Bitmap bmp;
	// 描画する座標
	private GeoPoint geoPoint;
	public DestinationOverlay(Bitmap bmp) {
		this.bmp = bmp;
	}
	public GeoPoint getGeoPoint() {
		return geoPoint;
	}
	public void setGeoPoint(GeoPoint geoPoint) {
		this.geoPoint = geoPoint;
	}

	@Override
	public void draw(Canvas canvas, MapView mapView, boolean shadow){
		// shadow = true,falseで2回呼び出される
		if(!shadow){
			// Mapの表示位置と座標から画面の描画位置を算出
			Projection pro = mapView.getProjection();
			Point p = pro.toPixels(geoPoint, null);

			// 描画(目的地がピンの先に来るようにy座標を調整)
			int centerX = p.x;
			int centerY = p.y - bmp.getHeight();
			// 画像の描画
			canvas.drawBitmap(bmp, centerX, centerY, null);
		}
	}
}
```

リスト3.4.5 DestnationOverlayの実装

3.5　直線と直線距離の表示

直線と直線距離の表示には、現在地と目的地を表示するために用いたOverlayを使用し、LineOverlayというOverlayを実装して現在地と目的地を結んだ直線とその直線距離を描画した。

また、直線距離の算出にはヒュベニの公式[9]を用いた。ヒュベニの公式は緯度、経度から2点間の距離を求める公式である。

3.5.1　直線距離の算出

ヒュベニの公式を用いた直線距離の算出をするためにHubenyというクラスを実装した。また、Hubenyクラスを実装するにあたってhttp://magpad.jugem.jp/?eid=121 [10]のソースコードを用いている。プログラムの実装例はリスト3.5.1に示す。

HubenyクラスではコンストラクタがGeoPoint型の現在地の位置情報と目的地の位置情報を受け取り、計算し、double型で計算結果を返している。

```
public class Hubeny{

        /*
         * DEG表記の例
         * 新宿駅の場合
         * 緯度：35.689325302714984
         * 経度：139.70051229000092
         * getDistance()で使用している公式は
         * 楕円体の原子，諸公式および定数について（国土地理院測地部）
         *
http://vldb.gsi.go.jp/sokuchi/surveycalc/algorithm/ellipse/ellipse.htm
         * を利用
         */
          //出発地点（DEG表記）
        private double latStart;
        private double lonStart;

        //到着地点（DEG表記）
        private double latEnd;
        private double lonEnd;

        //長半径(WGS84)
        private final double a = 6378137D;

        //扁平率(WGS84)
        private final double f = 1D / 298.257222101D;

        //日本測地系の場合
        //private double a = 6377397.155D;
        //private double f = 1D / 299.152813D;

        public Hubeny(GeoPoint pointStart, GeoPoint pointEnd) {
            setStartPoint(pointStart);
            setEndPoint(pointEnd);
        }
```

```
public void setStartPoint(GeoPoint point) {
    //GeoPoint型の緯度経度を通常のDEG表記に変換する(100万分の1倍する＝1E6
      で除算)
    latStart = point.getLatitudeE6() / 1E6;
    lonStart = point.getLongitudeE6() / 1E6;
}

public void setEndPoint(GeoPoint point) {
    //GeoPoint型の緯度経度を通常のDEG表記に変換する(100万分の1倍する＝1E6
      で除算)
    latEnd = point.getLatitudeE6() / 1E6;
    lonEnd = point.getLongitudeE6() / 1E6;
}

public double getDistance() {
    //緯度経度をラジアンに変換
    double radLatStart = latStart * Math.PI/180D;
    double radLonStart = lonStart * Math.PI/180D;
    double radLatEnd = latEnd * Math.PI/180D;
    double radLonEnd = lonEnd * Math.PI/180D;

    //二点間の平均緯度（ラジアン）
    double avgLat = (radLatStart + radLatEnd) / 2D;

    //扁平率の逆数
    double F = 1D / f;

    //第一離心率
    double e = (Math.sqrt(2 * F - 1)) / F;

    double W = Math.sqrt(1 - Math.pow(e, 2) *
                        Math.pow(Math.sin(avgLat), 2));
```

```
        //子午線曲率半径
        double M = (a*(1 - Math.pow(e, 2))) / Math.pow(W, 3);

        //卯酉線曲率半径
        double N = a / W;

        //2点間の緯度差(ラジアン)
        double dLat = radLatStart - radLatEnd;

        //2点間の経度差(ラジアン)
        double dLon = radLonStart - radLonEnd;

        //2点間の距離（メートル）
        double d = Math.sqrt(Math.pow(M*dLat, 2) + Math.pow(N *
                            Math.cos(avgLat) * dLon, 2));

        return d;
    }
}
```

リスト3.5.1 Hubenyの実装

3.5.2　直線と直線距離の描画

直線と直線距離の描画にはOverlayクラスを継承したLineOverlayクラスを実装した。LineOverlayクラスでは、GeoPoint型の現在地の位置情報と目的地の位置情報をコンストラクタまたはsetGeoPointメソッドが受け取る(リスト3.5.2)。また、Hubenyクラスで計算した直線距離をsetDistanceメソッドが受け取り、直線距離を描画できるようにdouble型からstring型に変換している(リスト3.5.3)。直線と直線距離の描画の実装の詳細はdrawメソッド内に定義する(リスト3.5.4)。

```
public class LineOverlay extends Overlay {

        private GeoPoint geoPointFrom = null;
        private GeoPoint geoPointTo = null;
        private String distance = null;

        //コンストラクタ
        public LineOverlay(GeoPoint from, GeoPoint to) {
            geoPointFrom = from;
            geoPointTo   = to;
        }

        //座標の設定
        public void setGeoPoint(GeoPoint from, GeoPoint to){
            geoPointFrom = from;
            geoPointTo   = to;
        }
```

リスト3.5.2　コンストラクタとsetGeoPointメソッドの実装

```
public void setDistance(double d){
        // フォーマット後の結果が入る変数
        distance ="";
                try {
                // 10進数Formatオブジェクト
                DecimalFormat df = new DecimalFormat();

                // パターンを設定
                df.applyPattern("0");

                // 小数点以下の桁数を指定
                // この引数にどちらも同じ値を入れると、
                // 固定小数値になります。
                df.setMaximumFractionDigits(1);
                df.setMinimumFractionDigits(1);

                // double変数をDoubleオブジェクトとして作成
                Double objnum = new Double(d/1000);

                // フォーマット
                distance = df.format(objnum);

                }catch(Exception e){
                        // エラーをキャッチ
                        distance = null;
                }
}
```

リスト3.5.3　setDistanceメソッドの実装

```
public void draw(Canvas canvas, MapView mapView, boolean isShadow) {
          super.draw(canvas, mapView, isShadow);
          //Projection#toPixels()で変換した場合の結果格納用
          Point pxPointFrom = new Point();
          Point pxPointTo   = new Point();

          //GePointからPointに変換
          Projection projection = mapView.getProjection();
          projection.toPixels(geoPointFrom, pxPointFrom);
          projection.toPixels(geoPointTo, pxPointTo);
          Paint paint = new Paint();
          Paint paint2 = new Paint();

          //Lineの両端を丸く
          paint.setStrokeCap(Paint.Cap.ROUND);

          //Lineのつなぎ目を丸く
          paint.setStrokeJoin(Paint.Join.ROUND);
          paint2.setTextSize(16);
          paint2.setColor(Color.RED);
          paint2.setStrokeWidth(2);
          paint.setStyle(Paint.Style.STROKE);
          paint.setColor(Color.RED);
          paint.setStrokeWidth(4);
          paint.setAlpha(64);
          paint.setAntiAlias(true);

   //drawメソッド内で、Canvas#drawLine()を使って始点から終点に線を引く
  canvas.drawLine(pxPointFrom.x,pxPointFrom.y,pxPointTo.x,pxPointTo.y,
       paint);
  //距離を表示する
  canvas.drawText(distance+"km",pxPointFrom.x,pxPointFrom.y,paint2);
 }
}
```

リスト3.5.4　drawメソッドの実装

第4章　実行結果

以下が本研究で作成したアプリケーションの実行結果である。作成したアプリケーションは、Android SDK に付属しているエミュレータで実行している。そのため現在位置を GPS で取得する代わりに、Dalvik Debug Monitor Service(DDMS)というデバッガの Emulator Control ビューから手動でエミュレータに緯度、経度を設定している(図1)。

この実行結果では現在地は島根大学の位置情報を取得している。図 2 の実行結果 1 は、目的地を松江駅に設定している。図 3 の実行結果 2 は目的地を出雲市駅に設定している。また、目的地を検索したとき、候補が複数あった場合は図 4 のように候補が表示されるのでその候補から選ぶ。

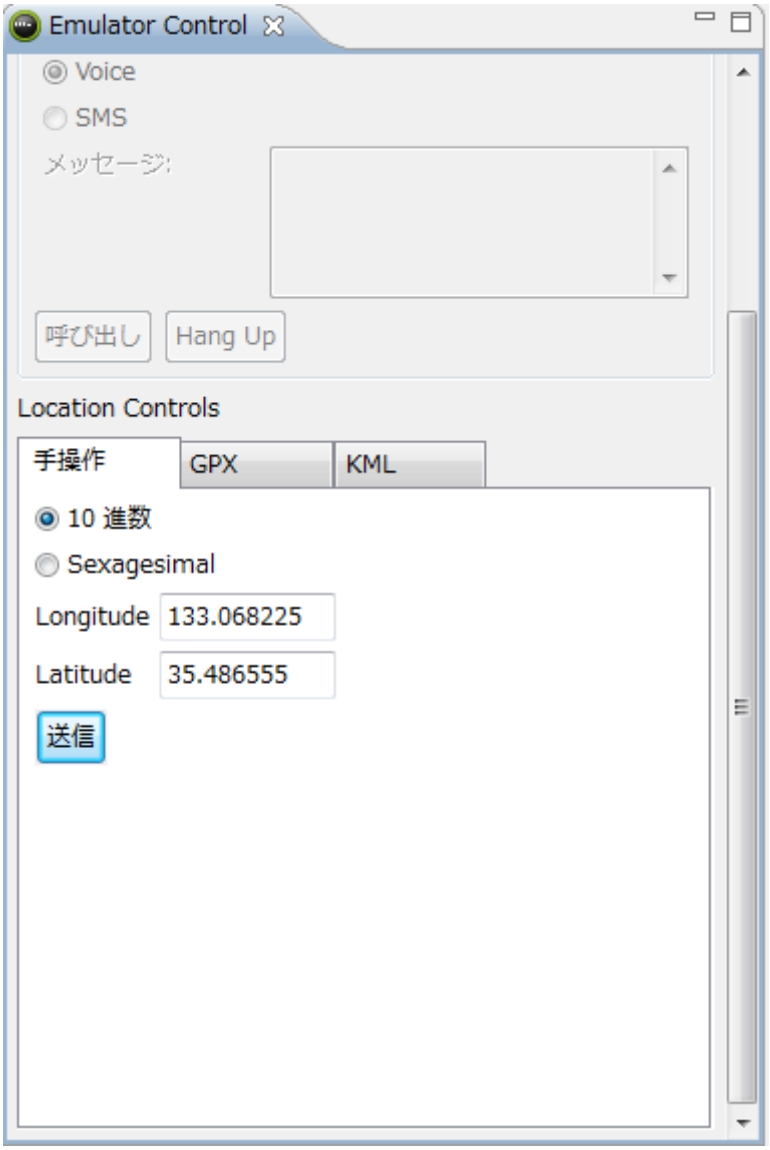

図 1　Emulator Control

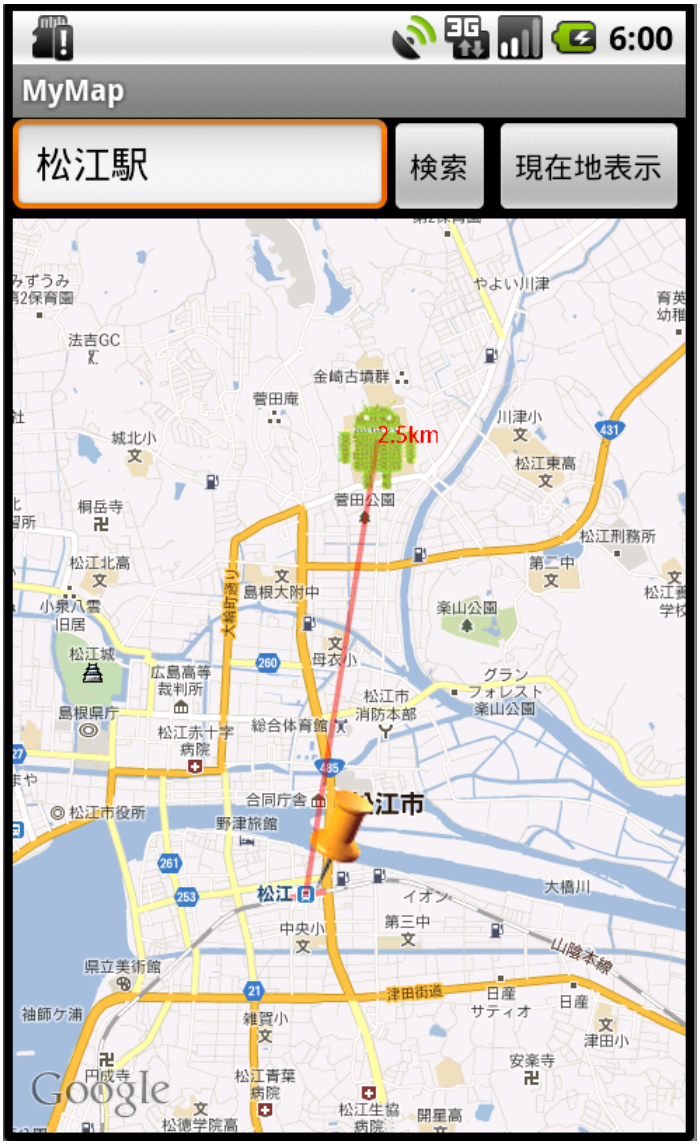
6:00
MyMap
松江駅
検索
現在地表示
2.5km
菅田庵
菅田公園
金崎古墳群
やよい川津
川津小
松江東高
城北小
桐岳寺
松江北高
島根大附中
第二中
松江刑務所
楽山公園
小泉八雲旧居
松江城
広島高等裁判所
島根県庁
松江赤十字病院
総合体育館
松江市消防本部
グランフォレスト楽山公園
松江市役所
合同庁舎
野津旅館
松江
イオン
大橋川
中央小
第三中
県立美術館
津田街道
日産サティオ
雑賀小
津田小
袖師ケ浦
安楽寺
松江青葉病院
松江生協病院
開星高
Google

図 2　実行結果 1


6:55
MyMap
出雲市駅
検索
現在地表示
31.6km
境港市
朝日山
米子鬼太郎空港
中海
安来市
宍道湖
鼻高山
出雲空港
出雲市
神西湖
斐伊川
雲南市
船通山
鬼の舌震
大万木山
猿政山
王貴峠
比婆山
道後山
Google

図 3　実行結果 2

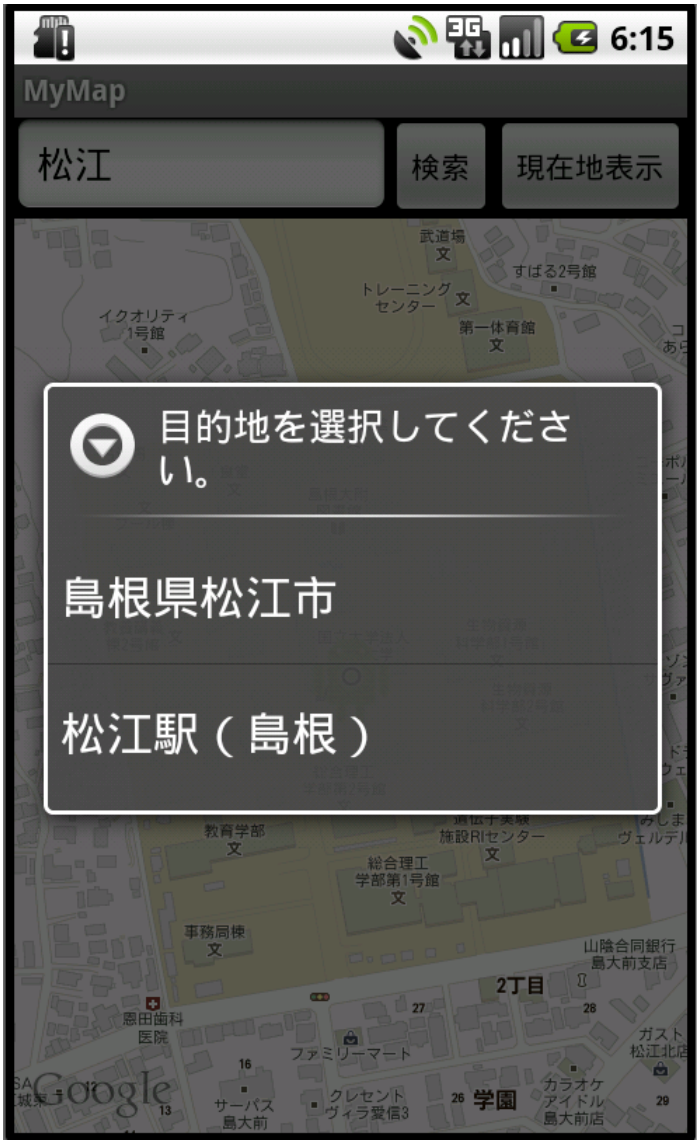
6:15
MyMap
松江
検索
現在地表示
目的地を選択してください。
島根県松江市
松江駅（島根）
武道場
すばる2号館
トレーニングセンター
イクオリティ1号館
第一体育館
教育学部
総合理工学部第1号館
事務局棟
2丁目
恩田歯科医院
ファミリーマート
サーパス島大前
クレセントヴィラ愛信3
学園
カラオケアイドル島大前店
山陰合同銀行島大前支店
Google

図 4　候補の選択

第5章　終論

本研究では、Android 端末用の Google マップを用いた地図アプリケーションを作成した。Android アプリケーションの開発は Java で開発する。アプリケーションの作成を通して、Java のプログラムの経験があれば、様々な API を用いて実用的なアプリケーションを作成できることを確認できた。

本研究で作成したアプリケーションの機能は、既存のアプリケーションにも存在する機能である。そのため、独自の機能の考案とその実装、また、機能の充実を図ることが課題として挙げられる。実装例としては、今回作成したアプリケーションでは現在地と目的地を直線で結んでいるが、直線ではなく道に沿った線で経路の表示と、その距離を表示できればさらに使いやすいアプリケーションになると思われる。また、近年起きている自然災害に対応して、避難経路などを表示できるようにすることも必要である。

謝辞

本研究を進めるにあたり、終始熱心に御指導頂きました田中章司郎教授には心より御礼申し上げます。また、同じ研究室の皆様には数々の御協力と御助言を頂きましたこと、深く感謝致します。

なお、本論文、本研究で作成したプログラム及び、データ、並びに関連する発表資料などの全ての知的財産権を本研究の指導教官の田中章司郎教授に譲渡致します。

参考文献

[1] 布留川英一、“Android2.1 プログラミングバイブル”、ソシム、2010

[2] http://blog.flatlabs.net/20110522_104427/

[3] http://java.sun.com/javase/ja/6/download.html

[4] http://www.eclipse.org/downloads/

[5] https://dl-ssl.google.com/android/eclipse/

[6] http://developer.android.com/sdk/index.html

[7] 江川崇、藤井大助、麻野耕一、藤田泰介、山田暁通、山田敏夫、佐野徹郎、竹端進、“Google Android プログラミング入門”、ASCII、2009

[8] http://codezine.jp/article/detail/4878

[9] http://yamadarake.web.fc2.com/trdi/2009/report000001.html

[10] http://magpad.jugem.jp/?eid=121