GEOCOMP

Fabrizio Schmidt

Andrea Rancan

Indice

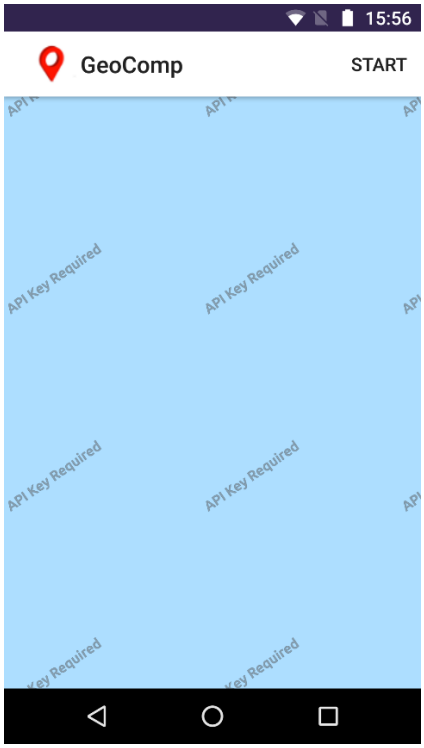
Client_App:

Questa applicazione consiste in una normale app di localizzazione e nella comparazione di vari sistemi GPS messi a confronto.

Il primo molto famoso e diffuso è la localizzazione di Google, il secondo il GPS nativo presente nel cellulare. Il secondo metodo dipenderà dalla qualità dei sensori installati su ogni specifico smartphone che ne fa uso.

La posizione che sarà mostrata sulla mappa non sarà unicamente quella nell'istante dove ci troviamo, ma memorizza uno storico per vedere il percorso effettuato.

L'app si compone di un'unica schermata quasi completamente adoperata per mostrare la cartina geografica.



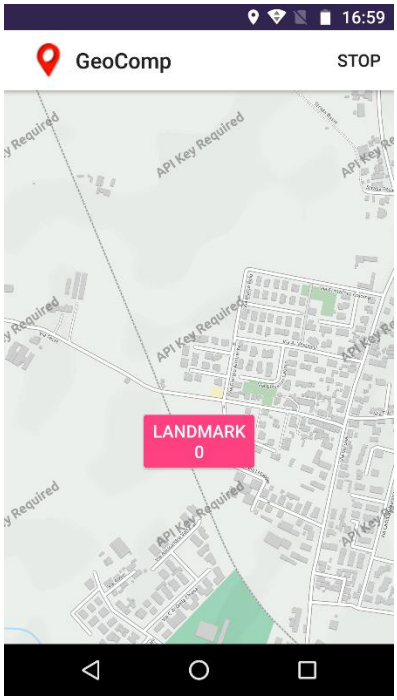
Nel menù sono presenti 2 item denominati rispettivamente "START" e "STOP". "START" compare all'avvio dell'app e quando si termina una sessione di geolocalizzazione premendo su "STOP". "STOP", invece, compare solamente durante una sessione di geolocalizzazione.

L'item "START" permette di avviare una sessione di geolocalizzazione mostrando una finestra di dialogo che permette di scegliere quali tecniche utilizzare nella sessione. Nel caso venga scelta una

tecnica di geolocalizzazione avanzata verrà anche visualizzata un'ulteriore finestra di dialogo per scegliere quale geolocalizzazione base utilizzare.

Nella finestra di dialogo è possibile scegliere le seguenti tecniche:

- "Groundtruth" serve per confrontare le varie geolocalizzazioni con un percorso prefissato a priori. Quando viene avviato il Groundtruth la mappa si posiziona sulla location calcolata dalla geolocalizzazione di Google e sullo schermo compare un tasto denominato "LANDMARK 0" per poter inserire punti di riferimento. Ogni volta che viene inserito un nuovo punto di riferimento il contatore di riferimento viene incrementato per poter distinguere tra di loro i vari punti di riferimento.



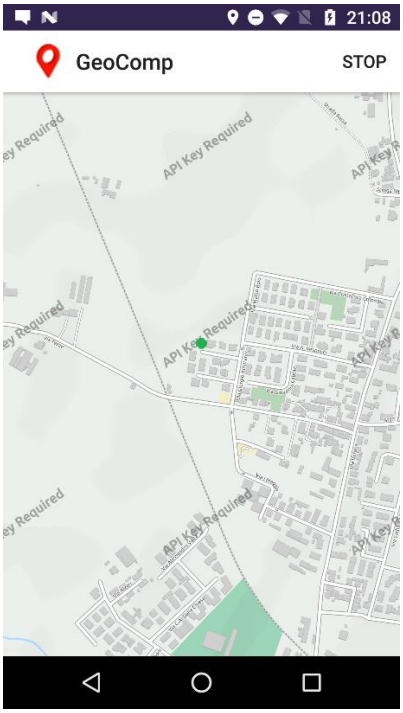
- La geolocalizzazione di Google è identificata con un punto blu.

- La geolocalizzazione nativa è identificata con un punto rosso.

La comparazione tra le 2 tecniche di base è soggettiva poiché non esiste la posizione corretta su cui fare riferimento, dipenderà dall'utilizzatore che conosce la sua posizione e può determinare quale dei due metodi funziona meglio.

Sono presenti inoltre tecniche di geolocalizzazione avanzate:

- Il Particle Filtering è identificato con un punto verde.



L'item "STOP" permette di interrompere le tecniche di geolocalizzazione attivate in precedenza lasciando memorizzato sul display il percorso mentre il tasto del Grountruth viene rimosso.

OpenStreetMap:

OSM mette a disposizione gratuitamente l'utilizzo di vari tipi di mappe precise e ben georeferenziate motivo per cui utilizzate, per poterle usare in un'applicazione Android bisogna fare determinati passaggi che ora saranno illustrati.

Per poter utilizzare le mappe e varie API messe a disposizione bisogna aggiungere la dipendenza a OSM, cercando nelle risorse dell'app *Grandle Scripts -> build.grandle(module: app)* e aggiungere il seguente comando:

```
dependencies {

    implementation 'org.osmdroid:osmdroid-android:5.4.1:release@aar'
}
```

Ora è possibile usare tag, metodi e mappe come si spiegherà in seguito.

Per poter memorizzare uno storico del percorso non solo dei dati ma anche visivo per la regolazione di zoom ed altre opzioni è stato necessario importare le seguenti librerie di OSM .

Ne esistono moltissime altre ma per quello che è stato realizzato, sono state sufficienti queste:

```
import org.osmdroid.api.IMapController;
```

Permette di applicare i controlli di zoom alla mappa.

```
import org.osmdroid.tileprovider.tilesource.TileSourceFactory;
```

Mette a disposizione la visualizzazione e l'utilizzo di diversi tipi di tiles.

```
import org.osmdroid.util.GeoPoint;
```

Mette a disposizione un oggetto per contenere le coordinate geografiche applicabili sulle mappe di OSM.

```
import org.osmdroid.views.MapView;
```

Permette di usufruire di oggetti mappa.

```
import org.osmdroid.views.overlay.Marker;
```

Mette a disposizione svariati costrutti e oggetti per l'utilizzo dei markers.

API Google:

Come per OSM anche per le API di Google bisogna aggiungere le dipendenze andando sempre nel file *Grandle Scripts -> build.grandle(module: app)* inserendo:

```
dependencies {
    implementation 'com.google.android.gms:play-services-location:15.0.1'
}
```

In questo modo avremo a disposizione i metodi per la localizzazione aggiungendo le librerie, ne sono servite molte in questo caso quindi saranno spiegate solo le più importanti:

```
import com.google.android.gms.location.LocationServices;
```

Mette a disposizione tutti i metodi e oggetti per memorizzare e usare i dati di longitudine e latitudine.

```
import com.google.android.gms.location.LocationRequest;
```

Permette di inoltrare richieste di localizzazione potendo definirne l'accuratezza.

```
import com.google.android.gms.location.LocationResult;
```

Permette di ricevere i dati di geolocalizzazione combinando segnale GPS e triangolazione delle reti.

Sviluppo:

L'intera applicazione per la maggior parte è strutturata su questi file che verranno spiegati nelle parti più importanti:

- **ClientServer.java:** si occupa di creare la connessione con il server, di mandare i dati e riceverne;
- **ActivityMain.java:** contiene il codice vero e proprio per l'esecuzione dell'app;
- **Content_main.xml:** contiene gli elementi visivi, ovvero l'interfaccia grafica dell'app;
- **AndroidMainfest.xm:l** contiene i vari permessi necessari per far funzionare l'app;
- **Activity_main_actions.xml:** contiene il menù di interfaccia, ovvero i pulsanti per l'esecuzione delle varie azioni;

ClientServer.java

Questa è la classe fondamentale per far eseguire le tecniche avanzate; essa si occupa di instaurare la connessione con il server e di mandargli periodicamente i dati, è stata pensata in modo da permettere all'applicazione di lavorare comunque durante il lasso di tempo per definire la connessione, inviare e ricevere i dati.

Quindi è stata implementata come thread, ovvero per essere eseguita in parallelo durante lo svolgimento della classe principale.

```
try
{

    result = apiInstance.compute(lat.toString(), lon.toString(),alt.toString());

} catch (ApiException e)
{
    System.err.println("Exception when calling CalculatorApi#compute");
    e.printStackTrace();
}
```

Sostanzialmente aspetta un risultato dal server, result, mandandogli latitudine, longitudine, altitudine.

apiInstance è un'istanza della classe CalculatoriApi.

ActivityMain.java:

ActivityMain è la classe dell'app che permette di visualizzare l'interfaccia grafica nel suo complesso(mappa, bottoni, item, finestre di dialogo ecc.), eseguire le tecniche di geolocalizzazione di base, salvare tipo, tempo e coordinate di ogni tecnica di geolocalizzazione su file di testo. Le tecniche avanzate hanno 2 righe di riferimento: una è SEND per l'invio delle informazioni riguardanti la tecnica di base utilizzata al server, l'altra è RECV e contiene le informazioni calcolate dall'algoritmo della tecnica sul server.
Come primo elemento abbiamo la creazione dello spazio mappa che gestisce tutto il resto del lavoro, il secondo comando permette di selezionare il tipo di tiles che vogliamo visualizzare, ne esistono di vario tipo tra cui: rete ferroviaria, rete urbana, rete ciclabile, ecc. in base alle esigenze si può estrapolare la cartina che ci mostri le caratteristiche di interesse.

Dopodiché applichiamo i controlli tra cui la possibilità di zoommare sia con i tasti che con le dita tramite il touch screen

```
map = findViewById(R.id.map);
map.setTileSource(TileSourceFactory.CYCLEMAP);
mapController = map.getController();
mapController.setZoom(18);
map.setBuiltInZoomControls(true);
map.setMultiTouchControls(true);
```

Groundtruth

- Il grountruth centra la mappa nella location calcolata dalla geolocalizzazione di Google e

```
now = new GeoPoint(latitude, longitude, altitude);
mapController.setCenter(now);
```

- Compare sullo schermo il tasto "LANDMARK 0" per inserire i punti di riferimento

```
btnground.setVisibility(View.VISIBLE);
```

- L'utente interagisce col groundtruth inserendo nuovi punti di riferimento grazie al metodo OnClick invocato dal metodo SetOnClickListener.

Geolocalizzazione google

```
private static final long UPDATE_INTERVAL_IN_MILLISECONDS = 2000;
```

```
private static final long FASTEST_UPDATE_INTERVAL_IN_MILLISECONDS = 5000;
```

Le API mettono a disposizione queste due variabili per gestire la velocità con cui aggiornare la posizione, in questo caso i valori sono predisposti per permettere una buona tolleranza per percorsi a piedi.

Una volta impostati questi valori si passa a controllare se i servizi e i permessi sono attivi e quindi è possibile cominciare la geolocalizzazione, se così non fosse viene mostrato un errore sul display.

In caso contrario si passa alla localizzazione vera e propria:

- Per prima cosa viene salvata l'ultima location trovata

```
mFusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
mSettingsClient = LocationServices.getSettingsClient(this);

mLocationCallback = new LocationCallback() {
    @Override
    public void onLocationResult(LocationResult locationResult) {
        super.onLocationResult(locationResult);
        // location is received
        mCurrentLocation = locationResult.getLastLocation();
```

- In seguito la location viene aggiornata

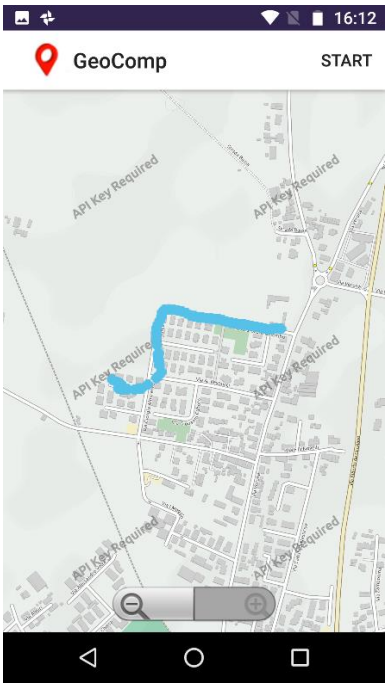
```
mLocationRequest = new LocationRequest();
mLocationRequest.setInterval(UPDATE_INTERVAL_IN_MILLISECONDS);
mLocationRequest.setFastestInterval(FASTEST_UPDATE_INTERVAL_IN_MILLISECONDS);
mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

LocationSettingsRequest.Builder builder = new
LocationSettingsRequest.Builder();
builder.addLocationRequest(mLocationRequest);
mLocationSettingsRequest = builder.build();
```

- Infine il punto corrispondente alla location trovata viene rappresentato sulla mappa e aggiunto alla lista delle location Google

```
now = new GeoPoint(lat, lon, alt);
mapController.setCenter(now);
Marker startMarker = new Marker(map);
startMarker.setPosition(now);
startMarker.setIcon(ResourcesCompat.getDrawable(getResources(),
R.drawable.google, null));
map.getOverlays().add(startMarker);
lstartMarker.add(startMarker);
```

Ripetendo i 3 passaggi soprastanti la mappa viene popolata dalle posizioni calcolate che nell'insieme rappresentano il percorso che stiamo facendo.



Geolocalizzazione nativa

Molto simile il funzionamento della localizzazione nativa. Anche in questo caso è possibile definire la bontà del segnale che vogliamo ricevere.

```
criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_MEDIUM);    //default

criteria.setCostAllowed(false);
```

- All'inizio viene salvato l'ultima location trovata.

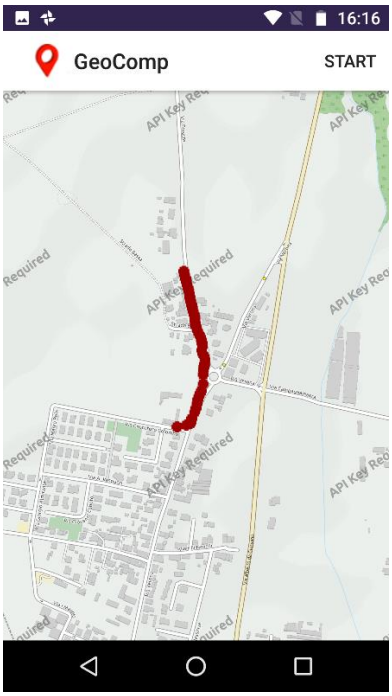
```
locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
provider = locationManager.getBestProvider(criteria, false);
Location location = locationManager.getLastKnownLocation(provider);
```

- Una volta controllato che i permessi sono tutti soddisfatti si imposta la velocità di aggiornamento della localizzazione come per Google.

```
locationManager.requestLocationUpdates(provider,5000, 1, mylistener );
```

- Infine il punto corrispondente alla location trovata viene rappresentato sulla mappa e aggiunto alla lista delle location Native.

```
nowN = new GeoPoint(lat, lon, alt);
mapController.setCenter(nowN);
Marker startMarkerN = new Marker(map);
startMarkerN.setPosition(nowN);
startMarkerN.setIcon(ResourcesCompat.getDrawable(getResources(),
R.drawable.inside, null));
map.getOverlays().add(startMarkerN);
lstartMarkerN.add(startMarkerN);
```



Particle Filtering

- SEND: all'invocazione del Particle Filtering, a seconda dell'opzione scelta, viene avviata la localizzazione Google oppure Native. La location trovata viene inviata al server che, tramite l'algoritmo di Particle Filter, calcola una nuova location più precisa.
- RECV: il server risponde alla chiamata del client e ritorna al client le coordinate di input.

Finestre di dialogo

Quando viene premuto il tasto "START" per avviare una nuova sessione di geolocalizzazione compare la finestra di dialogo principale dell'app che permette di scegliere una o più tecniche che di geolocalizzazione.

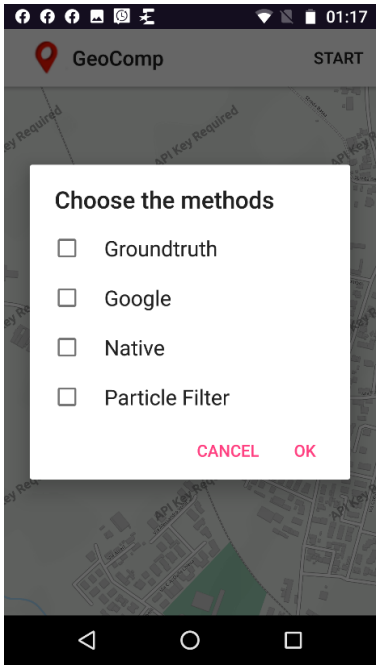
- Si assegna un titolo alla finestra e i nomi delle tecniche tra cui scegliere:

```
builder.setTitle("Choose the methods");
final String[] options = {"Groundtruth", "Google", "Native", "Particle Filter"};
```

- I nomi delle tecniche vengono fatti corrispondere agli item della finestra e ques'ultima viene costruita come MultiChoice per poter consentire più scelte.

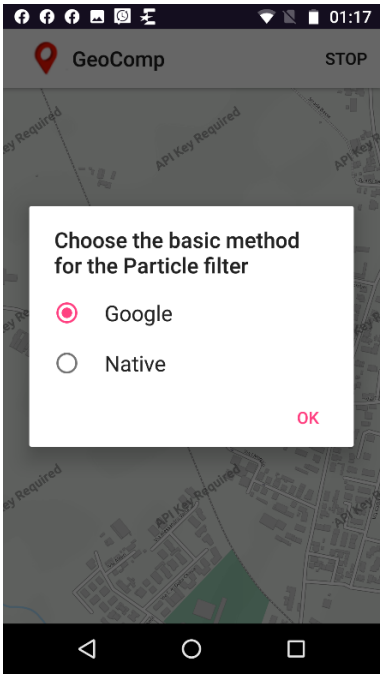
```
builder.setMultiChoiceItems(options, checked,
        new DialogInterface.OnMultiChoiceClickListener() // Item click listener
        {
            @Override
            public void onClick(DialogInterface dialogInterface, int i, boolean
isChecked) {
                checked[i] = isCheck
            }
        });
```

- Vengono poi inseriti nella finestra un tasto “OK” col metodo setPositiveButton per confermare le scelte e un tasto “CANCEL” col metodo setNegativeButton per tornare alla mappa e negare le scelte.



- Dopo aver confermato quali tecniche utilizzare col tasto “OK” compare, per ogni tecnica di geolocalizzazione avanzata selezionata ,una nuova finestra di dialogo che permette di scegliere quale tecnica utilizzare.

.



Queste nuove finestre di dialogo presentano solo il tasto “OK”(inserito col metodo setPositiveButton) per confermare la scelta riguardo la tecnica di base e le opzioni Google e Native. La finestra viene costruita come SingleChoice visto che sono una delle 2 tecniche di base può essere selezionata.

```
builder1.setTitle("Choose the basic method for the Particle filter");
final String[] options1 = {"Google", "Native"};
```

L’utente interagisce con i vari item delle finestre di dialogo grazie ai metodi OnClick invocati dai corrispondenti metodi SetOnClickListener.

Scrittura su file

Per ogni tecnica attivata ad ogni rilevazione viene inserita una nuova riga con Tipo, Tempo e Coordinate.

Le tecniche avanzate scrivono 2 righe: SEND per l’invio della location della tecnica di base al server e RECV che contiene il risultato dell’algoritmo della tecnica avanzata.

Ogni volta che si avvia una sessione di geolocalizzazione, se non esiste, viene creata la cartella rilevazioni_geocomp e un file di testo denominato come “data orario.csv” corrispondente alla sessione di geolocalizzazione.

Ad ogni tecnica di geolocalizzazione corrisponde un array di lunghezza 5. Se una tecnica viene selezionata ad ogni invocazione il suo array viene riempito con i valori corrispondenti e i suoi elementi vengono stampati sul file di sessione corrispondente.

- Nel caso del groundtruth la scrittura avviene ogni volta che viene premuto il tasto “LANDMARK I” con TIPO Groundtruth, TEMPO il tempo di pressione del tasto e COORDINATE gt_lat_I, gt_long_I, gt_alt_I.

- Nel caso delle altre tecniche di base e il SEND delle tecniche avanzate la scrittura avviene nel metodo corrispondente alla geolocalizzazione Google oppure Native a seconda della tecnica di base corrispondente. Per ognuna di queste tecniche viene assegnato un booleano che viene posto a TRUE nel caso la tecnica sia stata selezionata. Se il booleano vale true nel metodo corrispondente alla tecnica di base viene stampato su file la riga con valori TIPO la tecnica corrispondente, TEMPO il tempo in cui è stata calcolata la location della corrispondente tecnica di base e COORDINATE le coordinate corrispondenti alla location della tecnica di base.

- Nel caso della RECV delle tecniche avanzate la scrittura su file avviene solamente se il server risulta attivo e TIME corrisponde al tempo di ricezione della risposta del server. PF_RECV stampa su file le stesse coordinate inviate dal client tramite PF_SEND.

Content_main.xml:

Questa pagina si occupa di contenere tag per l'aspetto grafico dell'applicazione, come detto in precedenza non è molto complessa infatti è composta quasi solamente da linee di testo.

Il principale e di fondamentale importanza è il tag messo a disposizione da OSM (OpenStreetMap) che permette di definire uno spazio dove visualizzare le mappe, ogni tag mette a disposizione varie opzioni di stile e dimensione per adattarsi meglio allo schermo o alle esigenze della nostra app.

```
org.osmdroid.views.MapView
    android:id="@+id/map"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
</org.osmdroid.views.MapView>
```

Esso riempie completamente lo schermo in modo da mostrare la mappa usando più spazio possibile.

L'unico altro tag presente è il bottone corrispondente all'inserimento dei punti di riferimento del groundtruth che viene inizializzato con testo "LANDMARK 0" e aggiornata dalla classe MainActivity ogni volta che viene premuto.

```
<Button style="@style/Widget.AppCompat.Button.Colored"
    android:id="@+id/ground"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="350dp"
    android:layout_marginLeft="250dp"
    android:layout_centerHorizontal="true"
    android:enabled="true"
    android:text="LANDMARK 0"
    android:onClick="onClickBtn"/>
```

AndroidMainfest.xml:

Una qualsiasi applicazione di geolocalizzazione su un dispositivo mobile ha bisogno dei permessi da parte dell'utente di usufruire dell'antenna GPS integrata nel telefono. Questi tipi di permessi vengono richiesti nel file *AndroidManifest.xml* qui di seguito verranno spiegati tutti quelli utilizzati da questa applicazione.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"  />
```

Questi due permessi consentono l'uso della connessione ad internet a all'eventuale stato della rete, di fondamentale importanza in quanto le mappe usate di OpenStreetMap vengono richieste volta per volta ai server.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Il seguente permesso consente la memorizzazione di dati della memoria esterna del telefono (SD,

MSD), in questo programma è necessario memorizzare piccole porzioni di mappa, una volta definita la posizione dove ci troviamo verrà scaricata la tile di OpenStreetMap. In questo modo anche se dovesse venire a mancare la connessione ad internet si potrà comunque continuare ad usare il GPS in quanto la mappa è salvata sul dispositivo.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

I seguenti permessi permettono l'utilizzo dei sensori GPS. Il primo permette la localizzazione usando i sensori installati sul proprio dispositivo, il secondo invece permette di localizzarci usando il wifi, la precisione è ridotta rispetto hai sensori ma può essere utile in ambienti chiusi dove il segnale GPS ha più difficoltà ad arrivare.

Activity_main_actions.xml:

Di seguito un'altra pagina che ha la funziona di apparire nella schermata ma con le proprietà di azione, ovvero un menù composto dai pulsanti "START" e "STOP" ognuno con una determinata funzione spiegata in precedenza.

```
<item android:id="@+id/inizio"
    android:title="START"
    app:showAsAction="ifRoom" />

<item android:id="@+id/stop"
    android:title="STOP"
    app:showAsAction="ifRoom" />
```

Ogni azione è definita da un item che una volta premuto scatenerà una reazione all'interno del codice. Ognuno di essi è definito da un *id* il quale serve da riferimento per sapere che azione eseguire e da un' intestazione che compare nel menu.

Osservazioni:

Sono stati eseguiti molti test in varie circostanze per definire quale dei due metodi fosse il migliore.

La localizzazione di Google è molto più precisa quando siamo in ambienti chiusi o dove il segnale ha difficoltà ad arrivare, invece per quanto riguarda gli spazi aperti la differenza tra i due metodi è impercettibile sia per quanto riguarda l'errore temporale sia per quello spaziale.

Server:

L'idea è quella di immaginare le strade come zone in cui è altamente probabile trovarsi e più ci allontaniamo più la probabilità si abbassa.

Il lato server è sviluppato in modo da restituire una lista di gaussiane che descrivono dove sono localizzate le strade.

Per aprire il progetto, utilizzando eclipse, importare la cartella *spring-server* come Existing Maven Projects.

Sviluppo:

La classe che ci interessa è CalculatorApiController.java dove sono state create 5 funzioni per lo svolgimento delle operazioni di estrazione:

- **Download:** si occupa di scaricare una porzione di mappa geografica;
- **ExecuteMaperitive:** avvia un programma esterno per convertire la mappa in file SVG;
- **Extract:** estrae le informazioni che ci interessano dal file;
- **Calcolo:** va a calcolare tutti i parametri necessari per creare le gaussiane;
- **GaussCreate:** crea una lista di Gaussiane che descrivono la mappa.

Download mappa (download):

Il server prende in ingresso i seguenti parametri: latitudine, longitudine, altitudine, mezzo di trasporto e latitudine, longitudine, altitudine del punto precedente e vengono inviati dal client.

La latitudine e longitudine attuali vengono considerati come centro della mappa da scaricare. Per definire la dimensione totale della mappa usiamo 0.002 gradi, ovvero alla latitudine e longitudine aggiungiamo 0.002 gradi per definire i limiti superiore e destro e togliamo 0.002 per definire i limiti inferiore e sinistro.

La misura molto piccola è stata scelta per una questione di velocità nell'elaborazione dei dati essendo particolarmente onerosa.

La mappa viene scaricata da OSM(OpenStreetMap) attraverso un URL componibile, le quattro variabili bottom, left, top e right contengono le misure sopra citate.

```
"https://api.openstreetmap.org/api/0.6/map?bbox="+bottom+","+left+","+top+","+right;
```

Conversione (executeMaperitive):

affinché la mappa di OSM possa darci informazioni sulle strade abbiamo bisogni di convertirla in SVG, questo formato mette a disposizione oggetti geometrici per descrivere gli oggetti reali.

Per fare questo usiamo un'applicazione esterna Maperitive che permette l'elaborazione delle mappe di OSM, sito web: http://maperitive.net/.

Non essendo molto pesante come applicazione è stata aggiunta al progetto in modo da farne parte completamente, essa prende in ingresso una mappa OSM e restituisce la versione SVG.

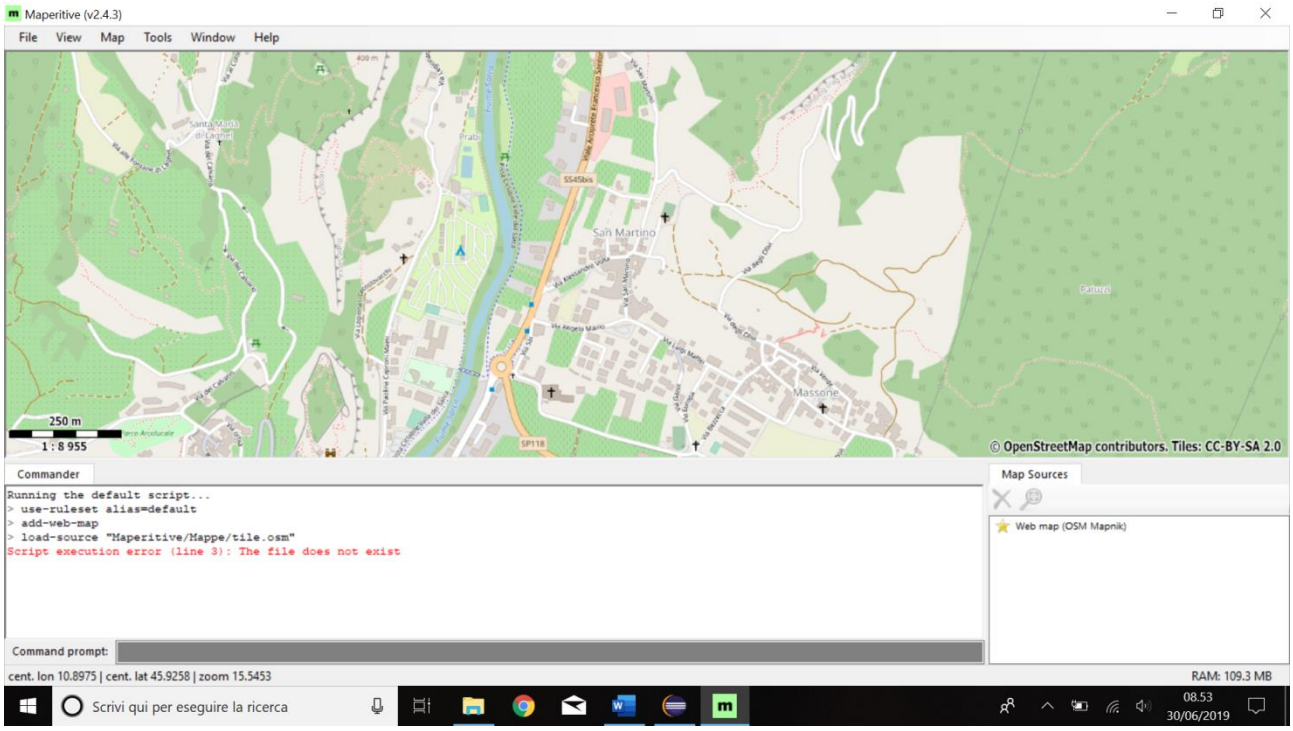
Per avviare questa applicazione dal programma avviamo un processo esterno come mostrato in seguito:

```
rt= Runtime.getRuntime().exec("Maperitive/Maperitive.exe");
rt.waitFor();
```

Il primo comando permette di avviare l'applicazione Maperitive e salvare il suo stato nella variabile rt.

Il secondo invece attende che la conversione sia terminata per proseguire con le operazioni.

Ovviamente avviare l'applicazione con convertirà automaticamente la mappa, infatti Maperitive possiede un'interfaccia grafica dove elaborare le mappe sia in modo interattivo sia in formato di script.



In alto a sinistra abbiamo i comandi per uno svolgimento manuale, invece nella console in basso abbiamo l'interfaccia per l'esecuzione degli script, uno script di base è già implementato e viene avviato all'accensione del programma.

Affinché la procedura si possa svolgere in completa autonomia è stato modificato lo script di default cosicché venga eseguito all'accensione:

use-ruleset alias=default
add-web-map
load-source "Maperitive/Mappe/tile.osm"
export-svg output/output.svg
exit

Lo script si trova nella cartella script dell'applicazione e si compone di soli 5 comandi:

Il primo comando serve a definire le regole di visualizzazione, teniamo quelle di default in quanto non è fondamentale l'aspetto grafico della mappa in questo caso;

Il secondo aggiunge la mappa l'interfaccia;

Il terzo prende in ingresso la mappa "tile.osm" che è quella scaricata in precedenza;

Il quarto con export-svg la converte e la salva;

E l'ultimo chiude l'applicazione.

Individuazione dati di interesse (extract):

Il file SVG è composto da vari tag che descrivono le primitive geometriche, per la nostra applicazione abbiamo bisogno dei dati relativi alle strade, esse sono rappresentate come segmenti adiacenti e descrivono l'andamento delle strade posizionandoci al centro di esse.

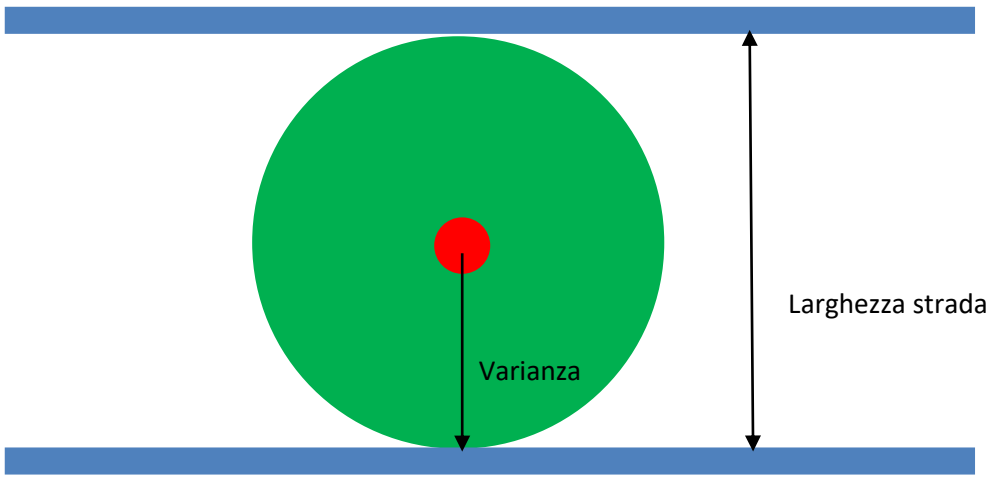
C'è usa sezione dedicata alla descrizione delle linee identificata da `<g id=\"Line_artwork"`, quindi viene scorso il file fino a trovare questo tag a sua volta al suo interno abbiamo i descrittori per le linee di qualunque oggetto parchi, edifici, strade ecc.

Quindi andiamo a cercare il tag `"<g id=\"highway"` che descrive tutte le strade questo descrittore nei parametri porta la larghezza della strada che viene salvata per essere usata successivamente come varianza della gaussiana.

Infine, si va ad analizzare il path delle strade che è specificato come punti cartesiani di inizio e di fine dei segmenti.

Calcolo dei parametri (calcolo):

L'idea fondamentale per creare i parametri delle gaussiane è quella di posizionare al centro della carreggiata la media della nostra distribuzione e come deviazione standard la larghezza della carreggiata.

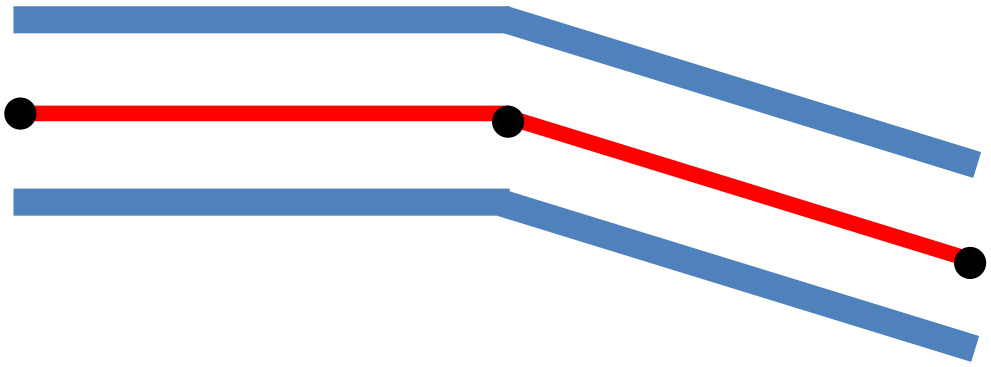


Come mostrato nell'immagine le linee blu indicano i lati della strada,il cerchio verde invece la gaussiana posizionata al centro della carreggiata che ha come deviazione standard metà della larghezza della strada.

Per realizzare queste gaussiane si va a prendere il path delle strade composto dai punti cartesiani e si salvano in una lista.

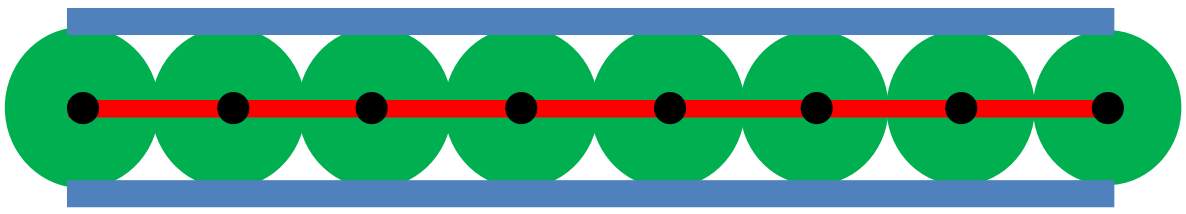
Affinché questi punti siano utilizzabile per analizzare i segmenti vanno presi a gruppi di 4, in questo modo è possibile calcolare la lunghezza del segmento che descrivono usando la distanza euclidea dato che ogni x,y rappresenta l'inizio di un segmento e la fine di un altro.

Di seguito l'esempio grafico di come sono memorizzate le strade:



Come detto in precedenza, le strade sono memorizzate come segmenti adiacenti al centro della strada, nel nostro caso i segmenti rossi, ogni punto nero sta ad indicare l'inizio e la fine di un determinato segmento ed è descritto come una x e una y nel piano cartesiano.

A elaborazione finita il risultato è una collezione di gaussiane che percorrono le strade della mappa:



Ora vediamo in dettaglio come avviene il calcolo, i file .svg oltre a elencare i punti cartesiani inserisce alcuni caratteri speciali M, h, v, Z :

M indica l'inizio del percorso;

h indica una linea orizzontale;

v indica una linea verticale;

Z indica che i punti definiscono una linea chiusa;

Quindi per ognuno di questi caratteri di vanno ad aggiungere punti o modificarli affinché mantengano la descrizione di partenza.

Dopodiché si prendono i punti a gruppi di quattro e si calcolano i segmenti, ogni segmento viene diviso in n parti, dove n è metà della larghezza della strada.

In questo modo si vanno a creare n gaussiane che si sovrappongono nelle code e sono adiacenti per varianza, il punto iniziale di ogni sottosegmento e metà della larghezza della strada vengono passati come parametri alla funzione che si occupa di creare la distribuzione.

Creazione della funzione di probabilità (GaussCreate):

quest'ultima funzione si occupa di creare la lista di gaussiane che descrivono la mappa presa in esame, per semplificare il processo vengono utilizzate le API Java (MultivariateGaussianDistribution) queste permettono di creare oggetti di tipo gaussiane multi variabili, indispensabili per descrivere una gaussiana in un mondo 3D.

```
mnd=new MultivariateGaussianDistribution(means, varianza);
```

La media, means, è definita come vettore di due componenti che contiene l'inizio dei sottosegmenti x,y invece varianza continente la larghezza della carreggiata.

Viene creato un oggetto di questo tipo per ogni sottosegmento e aggiunto ad una lista.

Tutte le gaussiane insieme vanno a formare la distribuzione di probabilità della mappa, da usare successivamente in un particle filter.