Capitolo 1

Progettazione ed Implementazione

Il progetto di tesi è composto da due parti distinte:

- Un'applicazione Android in grado di rilevare uno spostamento del mezzo, di registrare dati dai sensori inerziali e di inviare quest'ultimi al server tramite una connessione Wi-Fi.

- Un server sviluppato in Python in grado di ricevere dati dall'applicazione Android, interpretarli usando le API di OpenStreetMap e calcolare la posizione del mezzo mediante diversi algoritmi.

DISEGNO CHE SPIEGA IL FUNZIONAMENTO DEL SISTEMA

1.1 Applicazione Android

L'applicazione è stata scritta in Java, mediante Android Studio. La versione minima di SKD supportata è la 19 (KitKat) garantendo il corretto funzionamento dell'applicazione su circa l'86,3 % dei device in uso.

1.1.1 Concept

L'idea di base del progetto è quella di implementare un'antifurto inerziale per mezzi di trasporto. L'applicazione deve essere installata su un device, il quale deve essere posizionato in maniera salda al mezzo.

Si presuppone che il device sia posizionato in maniera tale che l'Azimuth del device sia uguale all'Azimuth del mezzo di trasporto. In un possibile sviluppo futuro si potrebbe calcolare l'orientamento al momento dell'inizializzazione e considerare questo offset nel momento in cui si salvano i dati.

L'applicazione una volta avviata si mette in attesa che il mezzo venga rubato, per riconoscere il furto e quindi lo spostamento del mezzo, comincia a rilevare i dati provenienti dall'accelerometro, ne calcola la magnitudo e nel caso quest'ultima sia maggiore di una certa costante significa che il device ha subito un'accelerazione di una certa intensità e quindi un possibile furto.

La fase di inizializzazione consiste nell'ottenere un unico dato, ossia le coordinate geografiche del mezzo parcheggiato. Siccome il mezzo sul quale è posizionato il dispositivo è parcheggiato, la velocità è pari a 0 e siccome presupponiamo che l'Azimuth del device sia pari all'Azimuth del mezzo non è nemmeno importante capire come è orientato il device all'interno del veicolo.

Dopo che l'applicazione ha rilevato un furto comincia a registrare i dati generati dai sensori e prova a collegarsi ad una qualche rete Wi-Fi, in caso di avvenuta connessione invia i dati ad un server remoto.

1.1.2 Implementazione

L'applicazione è composta da due Activity, ognuna delle quali corrisponde ad una diversa modalità di utilizzo dell'applicazione. Una volta lanciata l'applicazione l'utente potrà cambiare la modalità di utilizzo mediante un pulsante posto in alto a destra sulla AppBar.

- **Anti-Theft Mode**
  E' la modalità di utilizzo in cui l'applicazione funge da antifurto.

L'utente dovrà posizionare il device e premere il pulsante "Activate" che attiverà il motion detection. Questa modalità implementa il funzionamento descritto nella sezione riguardante al concept dell'applicazione.

- **Test Mode**
  Questa modalità mi è servita per costruirmi un dataset utile allo sviluppo della parte server. In questa modalità l'utente dovrà posizionare l'applicazione nel punto desiderato e quando vuole far partire la registrazione dovrà premere sul pulsante "Start", mentre dovrà premere il pulsante "Stop" per interrompere la registrazione e salvare il file contenente i dati generati dai sensori in locale sul device. Inoltre l'utente clickando sulle label corrispondenti alle coordinate potrà visualizzare il punto corrispondente su Google Maps, potrà vedere il nome del file appena esportato ed il contenuto di quest'ultimo.

L'applicazione presenta diverse classi, ora andremo a presentarle nel dettaglio.

MotionDetection

La classe MotionDetection si occupa di rilevare un movimento del mezzo. Viene invocata solamente nella Anti-Theft Mode quando l'utente preme su "Activate".
MotionDetection implementa l'interfaccia SensorEventListener e chiede al SystemService tramite il SensorManager la possibilità di leggere i dati generati dall'accelerometro ad una frequenza pari a 5Hz. Ogni volta che l'applicazione rileva una minima accelerazione ne calcola la magnitudo dalla quale ci sottraiamo la forza di gravità

$$magnitude = \sqrt{x^2 + y^2 + z^2} - gravity$$

Se la magnitudo è maggiore rispetto ad una certa costante, l'applicazione capisce che il mezzo su cui è posizionato il device ha subito un cambiamento di velocità e quindi è stato spostato dal punto originario. Dopo diverse prove ho

impostato la sensibilità del MotionDetection ad un valore pari ad 1 in maniera che l'applicazione sia in grado di minimizzare sia i falsi positivi che i falsi negativi, quindi, che riesca ad interpretare come un furto lo spostamento del mezzo ma che non riesca ad interpretare come tale, ad esempio, le vibrazioni prodotte dal passaggio di mezzi pesanti nelle immediate vicinanze.

Nel momento in cui l'applicazione rileva lo spostamento del mezzo, vengono lanciati in esecuzione i due Service, il primo relativo alla registrazione dei dati provenienti dai sensori, il secondo relativo all'invio dei dati al server.

gpsPosition

gpsPosition è la classe adibita alla geolocalizzazione del device. Mediante il LocationManager ottiene dal SystemService la possibilità di ottenere informazioni dal Location_Service.

jsonUtils

Questa classe gestisce il jsonObject contenente tutti i valori ottenuti dai sensori. Nel caso in cui l'applicazione sia in esecuzione in modalità Test Mode, il jsonObject viene salvato in un file .json nella memoria locale del device. Nel caso in cui l'applicazione sia in esecuzione in modalità Anti-Theft Mode, invece, il jsonObject viene passato al Service adibito all'invio dei dati al server.

parallelIntentService

ParallelIntentService estende la classe Service ed implementa un comportamento simile agli IntentService. E' stata utile per gestire due service parallelamente in due thread differenti e non entrambi nell'UI thread. SPIEGARE UN PO IL FUNZIONAMENTO E METTERE IL LINK

httpRequest

Per gestire le richieste HTTP ho usato "android-async-http", una comoda libreria per trasmettere informazioni ad un server sia in maniera sincrona che asincrona. La richiesta HTTP avviene nello stesso thread in cui avviene la chiamata ad essa e fino a che la richiesta non è conclusa, il thread si mette in wait, facendo quindi una comunicazione di tipo sincrono. SPIEGARE MEGLIO IL FUNZIONAMENTO E METTERE IL LINK

logData

logData è un Service il cui scopo è quello di registrare tutti i dati provenienti dai sensori e salvarli mediante i metodi forniti da jsonUtils in un unico jsonObject.

Tramite il SensorManager l'applicazione ottiene dal SystemService l'accesso al SENSOR_SERVICE. Il SensorManager si mette in ascolto dai dati provenienti da diversi sensori: accelerometro, magnetometro e giroscopio .

Oltre a questi sensori, l'applicazione calcola la matrice rotazionale del device tramite i valori raccolti dal magnetometro e dall'accelerometro. Una volta ottenuta la matrice rotazionale tramite il getRotationMatrix, otteniamo un'array di valori i quali indicano l'orientamento del device tramite getOrientation. I valori ottenuti saranno ¡Azimuth, Pith, Roll¿. Il calcolo dell'orientamento tramite matrice rotazionale è indispensabile per i dispositivi che non dispongono del giroscopio e per ottenere un valore di Azimuth corretto esente da errori relativi all'orientamento del dispositivo.

L'analisi dei dati generati dal sensore avviene ad una frequenza di 20Hz.

checkSend

checkSend è un Service il quale si occupa dell'inviare i dati al server. Ogni volta che il Service è in esecuzione controlla se il Wi-Fi è abilitato ed attivo. In caso di risposta affermativa, controlla se il dispositivo è attualmente connesso ad una rete Wi-Fi mediante il metodo checkConnection().

checkConnection() mediante il ConnectivityManager ottiene le informazioni relative alla network attiva, se è presente una network attiva ed è di tipo Wi-Fi, controlla se il device è connesso, scrive sul Json le informazioni relative alla network e invia i dati. Se il dispositivo non è collegato ad alcuna rete Wi-Fi, tramite il metodo searchFreeWifi() ottengo una lista delle rete Wi-Fi disponibili, le filtro escludendo le reti che hanno politiche di accesso e restituisco la network con il maggior valore di RSSI, ossia quella in cui il segnale è maggiore. Il service proverà a collegarsi, in caso di avvenuta connessione, salverà nel Json le informazioni relative alla network e invierà i dati.

L'invio dei dati è gestito dal metodo sendData(), il quale farà una richiesta HTTP di tipo POST al server mediante la classe httpRequest.

Bibliografia